Native Configuration Manager API for Windows Operating Systems

Library Reference

August 2006

05-1903-008

Publication Date: August 2006

Document Number: 05-1903-008

Intel
1515 Route 10
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support

For **Products and Services Information**, visit the Intel Telecom and Compute Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:
http://www.intel.com/buy/networking/telecom.htm

Contents

Revision History 5

About This Publication 7

Purpose 7

Applicability 7

Intended Audience 7

How to Use This Publication 8

Related Information 8

1 Function Summary by Category 9

1.1 Query Configuration Functions 9

1.2 Modify Configuration Functions 10

1.3 System Functions 11

1.4 System Administration Functions 11

1.5 TDM Bus Functions 12

2 Function Information 15

2.1 Function Syntax Conventions 15

NCM_AddDevice( ) – instantiate a device 16

NCM_ApplyTrunkConfiguration( ) – configure DMV/B trunks 19

NCM_Dealloc( ) – deallocate memory 23

NCM_DeallocValue( ) – deallocate memory 25

NCM_DeleteEntry( ) – remove configuration information 27

NCM_DetectBoards( ) – detect auto-detectable boards 30

NCM_DetectBoardsEx( ) – initiate board auto-detection 32

NCM_EnableBoard( ) – enable or disable device initialization 35

NCM_GetAllDevices( ) – get a list of installable device models 38

NCM_GetAllFamilies( ) – get a list of installable families 40

NCM_GetAUID( ) – get the AUID of a family or device 42

NCM_GetBridgeDeviceHMPClockMasterFallbackList( ) – retrieve the bridge device clock master fallback list 44

NCM_GetClockMasterFallbackList( ) – get the clock master fallback list 48

NCM_GetCSPCountries( ) – get a list of supported countries 50

NCM_GetCspCountryCode( ) – get a country code 52

NCM_GetCspCountryName( ) – get a country name 54

NCM_GetCspFeaturesValue( ) – get a country-specific parameter value 56

NCM_GetCspFeaturesValueRange( ) – get the value range of a parameter 59

NCM_GetCspFeaturesVariables( ) – get the values for a parameter 61

NCM_GetDialogicDir( ) – get a given Intel Dialogic directory 63

NCM_GetDlgSrvStartupMode( ) – get the startup mode of the system service 65

NCM_GetDlgSrvState( ) – get the state of the system service 67

NCM_GetDlgSrvStateEx( ) – get the state of the system service 69

NCM_GetErrorMsg( ) – get the error message for an error code 71

NCM_GetFamilyDeviceByAUID( ) – get the family or device from an AUID 73
NCM_GetInstalledDevices( ) – get all instantiated devices for a family 75
NCM_GetInstalledFamilies( ) – get all instantiated families 77
NCM_GetProperties( ) – get the installable properties of a device 79
NCM_GetPropertyAttributes( ) – get a property's attributes 82
NCM_GetSystemState( ) – return the system service state 84
NCM_GetTDMBusValue( ) – get a TDM bus parameter value 86
NCM_GetValue( ) – get an instantiated value 88
NCM_GetValueEx( ) – get an instantiated value 91
NCM_GetValueRange( ) – get the value range for a parameter 94
NCM_GetValueRangeEx( ) – get the value range for a parameter 97
NCM_GetVariableAttributes( ) – get a parameter's attributes 100
NCM_GetVariables( ) – get the parameters for a property section 102
NCM_GetVersionInfo( ) – get OS and system software versions 105
NCM_IsBoardEnabled( ) – determine if a device is enabled or disabled 107
NCM_IsEditable( ) – determine if a parameter can be edited 109
NCM_ReconfigBoard( ) – reconfigure a DM3architecture board 112
NCM_RemoveBoard( ) – remove a board from the NCM database 114
NCM_SetClockMasterFallbackList( ) – set the clock master fallback list 116
NCM_SetDlgSrvStartupMode( ) – set the system service startup mode. 118
NCM_SetTDMBusValue( ) – set the TDM bus values. 120
NCM_SetValue( ) – sets a configuration parameter value 122
NCM_SetValueEx( ) – instantiate a configuration parameter value 125
NCM_StartBoard( ) – start an individual board 128
NCM_StartDlgSrv( ) – initiate the system service 130
NCM_StartSystem( ) – start all boards 132
NCM_StopBoard( ) – stop an individual board 134
NCM_StopDlgSrv( ) – stop the system service 136
NCM_StopSystem( ) – stop all boards in a system 138

3 Events 141

4 Data Structures 143
NCM_DETECTION_DETAILS – board detection details 144
NCM_DETECTION_INFO – board detection callback functions. 145
NCM_DETECTION_RESULT – board detection results 146
NCMString – defines NCM variables. 147
NCMSysVersion – system software\operating system version information 148
NCMValueEx – defines parameter values. 149
NCMTrunkConfig – information needed for trunk configuration 150
NCMVariableAttributes – defines attributes for a variable. 151

5 Error Codes 153

Index 157

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-1903-008	August 2006	NCM_ApplyTrunkConfiguration( ) section: Added pMediaLoad and pErrorMsg parameters. Replaced example code.
05-1903-007	October 2005	NCM_RemoveBoard( ) section: Added this section.
05-1903-006	October 2005	Global change: Minor editorial updates for Intel NetStructure® Host Media Processing release. NCM_ApplyTrunConfiguration( ) section: Modified first two parameter descriptions. The parameters are data structures, not pointers to data structures. NCM_ApplyTrunConfiguration( ) section: Added information about trunk protocol groupings for Intel® Host Media Processing Interface Boards NCM_GetBridgeDeviceHMPClockMasterFallbackLIst( ) section: Added this section for Intel NetStructure Host Media Processing release.
05-1903-005	April 2005	NCM_StartSystem( ) section: Updated example code for this function. NCM_StopSystem( ) section: Updated example code for this function.
05-1903-004	April 2005	Global change: Removed support for third party device functions NCM_AddThirdPartyDevice( ), NCM_AllocateTimeslots( ), NCM_DeallocateTimeslots( ), NCM_GetThirdPartyDeviceBusCaps( ), NCM_QueryTimeslots( ), and NCM_RemoveThirdPartyDevice( ) functions have been removed. They are not supported by the current release.
05-1903-003	September 2004	NCM_GetValueEx( ): Added second note under Description (PTR 32385).

Document No.	Publication Date	Description of Revisions

05-1903-002	November 2003	Global changes: Added a note to all the functions that have an Ex function saying that the Ex functions should be used. Added a note wherever applicable about not parsing the unique device name from an application.
		Query Configuration Functions section: Changed “read configuration functions” to “query configuration functions”.
		New reference pages for new functions:
		NCM_AddThirdPartyDevice( )
		NCM_AllocateTimeslots( )
		NCM_ApplyTrunkConfiguration( )
		NCM_DeallocateTimeslots( )
		NCM_GetCspCountries( )
		NCM_GetCspCountryCode( )
		NCM_GetCspCountryName( )
		NCM_GetCspFeaturesValue( )
		NCM_GetCspFeatuesValueRange( )
		NCM_GetCspFeaturesVariables( )
		NCM_GetSystemState( )
		NCM_GetThirdPartyDeviceBusCaps( )
		NCM_QueryTimeslots( )
		NCM_ReconfigBoard( )
		NCM_RemoveThirdPartyDevice( )
		NCM_StartSystem( )
		NCM_StopSystem( )
		NCM_AddDevice( ) reference page: Updated description. Expanded See Also section.
		NCM_DetectBoardsEx( ) reference page: rUpdated description
		NCM_GetDialogicDir( ) reference page: Updated description.
		NCM_GetValue( ) reference page: Updated description.
		NCM_GetValueEx( ) reference page: Updated description.
		NCM_GetVersionInfo( ) reference page: Expanded See Also section.
		NCM_StartBoard( ) reference page: Updated description.
		NCM_StartDlgSrv( ) reference page: Updated description. Added new sample code.
		NCM_StopBoard( ) reference page: Updated description.
		NCM_StopDlgSrv( ) reference page: Updated description.
		Events chapter: Updated description.
		NCMTrunkConfig reference page: New data structure page.

05-1903-001	November 2002	Initial version of document. Much of the information contained in this document was previously published in the *Customization Tools for Installation and Configuration for Windows*, document number 05-1103-007.

About This Publication

The following topics provide information about this publication:

- Purpose
- Applicability
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This publication provides a reference to the functions, data structures and error codes of the Native Configuration Manager (NCM) library for Intel® telecom products.

This publication is a companion document to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*, which provides guidelines for developing applications with the NCM API.

Applicability

This document version is published for Intel NetStructure® Host Media Processing Software Release 3.0 for Windows*.

This document may also be applicable to later Intel NetStructure Host Media Processing (HMP) software releases as well as Intel Dialogic system releases. Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This publication is intended for the following customer types:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

How to Use This Publication

This document assumes that you are familiar with the C programming language and the Windows* operating system.

Throughout this publication, the term "installable" indicates that the configuration data element to which it applies is supported by the DCM catalog. For example, an installable device is a device that is supported in the DCM catalog. The terms "instantiate" and "instantiated" refer to the process of creating system configuration data. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for more information about the distinction between the DCM catalog and the system configuration.

This publication is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the various categories of NCM library functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to all NCM library functions.

  Note: The **Ex** versions of functions should be used where available (for example, **NCM_GetValueEx(**) instead of **NCM_GetValue(**)). The non-Ex functions are provided for backwards compatibility.
- Chapter 3, "Events" contains information about events that are generated by certain NCM library functions.
- Chapter 4, "Data Structures" provides an alphabetical reference to the data structures used by the NCM library functions.
- Chapter 5, "Error Codes" presents a list of error codes that may be returned by the NCM library functions.

Related Information

Refer to the following publications and Web sites for more information:

- *Native Configuration Manager API for Windows Operating Systems Programming Guide*
- *Event Service API for Windows Operating Systems Library Reference*
- *Event Service API for Windows Operating Systems Programming Guide*
- *Intel NetStructure® Host Media Processing Interface Boards Configuration Guide*
- *Intel® DM3 Architecture Products Configuration Guide*
- *http://developer.intel.com/design/telecom/support/* for technical support
- *http://www.intel.com/design/network/products/telecom/index.htm* for product information

Function Summary by Category 1

This chapter describes the categories into which the NCM library functions can be logically grouped. Functions are divided into the following categories:

- Query Configuration Functions 9
- Modify Configuration Functions 10
- System Functions. 11
- System Administration Functions 11
- TDM Bus Functions 12

1.1 Query Configuration Functions

Query configuration functions query either the DCM catalog or your current system configuration settings for information about device families and individual devices. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for information about the distinction between the DCM catalog and the system configuration.

The query configuration functions are as follows:

NCM_GetAllDevices( )
returns a list of installable device models

NCM_GetAllFamilies( )
gets a list of installable device families

NCM_GetCSPCountries( )
gets a list of supported countries

NCM_GetCspCountryCode( )
gets the country code for a country

NCM_GetCspCountryName( )
gets the country named for a country code

NCM_GetCspFeaturesValue( )
gets a country-specific parameter value

NCM_GetCspFeaturesValueRange( )
gets the value range

NCM_GetCspFeaturesVariables( )
gets values

NCM_GetInstalledDevices( )
gets all instantiated devices for a family

NCM_GetInstalledFamilies( )
returns all instantiated device families

NCM_GetProperties( )
gets the installable properties for a device

NCM_GetPropertyAttributes( )
gets a properties attributes

NCM_GetValue( )
returns the value of a parameter

NCM_GetValueEx( )
extended function that behaves identically to **NCM_GetValue( )**, but returns the value of a parameter using the NCMValueEx data structure

NCM_GetValueRange( )
gets the range of valid values for a parameter

NCM_GetValueRangeEx( )
extended function that behaves identically to **NCM_GetValueRange( )**, but returns the value of a parameter using the NCMValueEx data structure

NCM_GetVariableAttributes( )
returns a parameter's attributes

NCM_GetVariables( )
gets the parameters for a property section

NCM_IsBoardEnabled( )
returns information about whether or not a device is to be initialized when the Intel Dialogic system is started

NCM_IsEditable( )
returns information about whether or not a given parameter can be modified

1.2 Modify Configuration Functions

The modify configuration functions allow you to add, modify, and delete configuration information for device families and individual devices in your system.

Note: You can modify configuration data only for the Intel® Dialogic or Intel NetStructure® products supported by the system software release. Refer to the *Release Guide* for list of supported hardware.

The modify configuration parameters include the following:

NCM_AddDevice( )
instantiates a device in your system configuration

NCM_ApplyTrunkConfiguration( )
creates trunk configuration files for DMV/B boards

NCM_DeleteEntry( )
deletes configuration information

NCM_EnableBoard( )
determines whether or not a device is to be initialized when the Intel Dialogic system is started

NCM_ReconfigBoard( )
reconfigures an individual DM3 architecture board

NCM_RemoveBoard( )
removes a board from the NCM database

NCM_SetValue( )
sets the value of a configuration parameter

NCM_SetValueEx( )
extended function that behaves identically to **NCM_SetValue( )**, but uses the NCMValueEx data structure as input

1.3 System Functions

The system functions allow you to interface with the Intel Dialogic or HMP system. You can set the system startup mode and query its current status. Refer to the "System Administration Functions" section for functions that start and stop the Intel Dialogic or HMP system.

The system functions are as follows:

NCM_GetDlgSrvStartupMode( )
returns the startup mode of the Intel Dialogic or HMP system

NCM_GetDlgSrvState( )
returns the state of the Intel Dialogic or HMP system

NCM_GetDlgSrvStateEx( )
extended function that returns the state of the Intel Dialogic or HMP system using the Win32 SERVICE_STATUS data structure

NCM_SetDlgSrvStartupMode( )
sets the startup mode of the Intel Dialogic or HMP system

NCM_GetSystemState( )
returns the status of the Intel Dialogic or HMP system service

1.4 System Administration Functions

The system administration functions allow you to manage the various components of your Intel Dialogic system (hardware, memory, NCM library error messages, system service state etc.).

System administration functions are as follows:

NCM_Dealloc( )
deallocates memory occupied by an NCMString data structure

NCM_DeallocValue( )
deallocates memory occupied by an NCMValueEx data structure

NCM_DetectBoards( )
initiates auto-detection of instantiated boards in your system

NCM_DetectBoardsEx( )
extended function that behaves identically to **NCM_DetectBoards( )**, but uses the NCM_DETECTION_RESULT data structure to return detailed information about the status of the auto-detection process

NCM_GetAUID( )
returns the Addressable Unit Identifier (AUID) for an instantiated device

NCM_GetDialogicDir( )
returns directory information about specific Intel Dialogic files

NCM_GetErrorMsg( )
gets the error message text string for a given error code

NCM_GetFamilyDeviceByAUID( )
returns a the family name for a device that has been assigned a given Addressable Unit Identifier (AUID)

NCM_GetVersionInfo( )
gets operating system and Intel Dialogic system or HMP software version information

NCM_StartBoard( )
starts an individual board

NCM_StartDlgSrv( )
initiates the system service

NCM_StartSystem( )
starts all boards in a system

NCM_StopBoard( )
stops an individual board

NCM_StopDlgSrv( )
stops the system service

NCM_StopSystem( )
stops all boards in a system

1.5 TDM Bus Functions

TDM bus functions allow you to manage the TDM bus within your Intel Dialogic or HMP system. You can set a clock master fallback list and/or set the value of individual TDM bus parameters. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for information about clock master fallback and bridge device HMP fallback.

The TDM bus functions are as follows:

NCM_GetBridgeDeviceHMPClockMasterFallbackList( )
retrieves the bridge device clock master fallback list

Note: This function is only supported for Intel NetStructure® Host Media Processing releases.

NCM_GetClockMasterFallbackList( )
returns the system's list of clock master fallback devices

NCM_SetTDMBusValue( )
gets a parameter value for the TDM bus

NCM_SetClockMasterFallbackList( )
sets a user-defined list of clock master fallback devices

NCM_SetTDMBusValue( )
sets a parameter value or the TDM bus

Function Information 2

This chapter provides an alphabetical reference to the functions in the NCM library.

2.1 Function Syntax Conventions

The NCM API functions use the following syntax:

```
NCMRetCode NCM_functionName(parameter1,...parameterN)
```

where:

NCMRetCode
: refers to the return field for the function. NCMRetCode is defined in the *NCMTypes.h* file.

NCM_functionName
: indicates the name of the function

parameter1
: represents the first parameter

parameterN
: represents the last parameter

NCM_AddDevice( )

Name:	NCMRetCode NCM_AddDevice(pncmFamily, pncmDeviceModel, pncmDeviceUnique)	
Inputs:	NCMFamily* pncmFamily	• pointer to a data structure containing a device family name
	NCMDevice* pncmDeviceModel	• pointer to a data structure containing a device model name
	NCMDevice* pncmDeviceUnique	• pointer to a data structure containing a unique device name
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

■ Description

The **NCM_AddDevice( )** function instantiates a device in the system configuration. Upon adding the device, this function will establish default settings for all configuration parameters pertaining to the device.

Parameter	Description
pncmFamily	points to an NCMString data structure containing the family name. The value of the data structure must be an installable family (i.e one that is supported in the DCM catalog). All instantiated families of devices in your current system configuration can be retrieved by **NCM_GetInstalledFamilies( )**.
pncmDeviceModel	points to an NCMString data structure containing the device's model name. The value of the structure must be an installable device.
pncmDeviceUnique	points to an NCMString data structure containing the device's unique name. This name can be any string that sufficiently distinguishes multiple instantiations of the same device model. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

■ Cautions

- Because devices are instantiated in the system configuration according to their unique device name, it is impossible to correlate an instantiated device with a device model name. Therefore, it is recommended that you embed the device model name within the unique device name when you instantiate a device with the **NCM_AddDevice( )** function.
- The **pncmFamily** and **pncmDeviceModel** pointers must reference information that is valid in the current DCM catalog. For information about how to determine which families, devices and

configuration parameters are valid in the current DCM catalog, refer the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

- This function adds to the information instantiated in the current system configuration. It has no effect on the installable families, devices and configuration parameters defined in the DCM catalog. For more information about the distinction between the system configuration and the DCM catalog, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Errors

Possible errors for this function include:

NCME_NO_RESOURCES
: there are no more system resources available for the device to use (memory, IRQ or ports).

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_BAD_INF
: there was an error parsing the DCM catalog

NCME_INVALID_FAMILY
: family name is invalid

NCME_INVALID_DEVICE
: device name is invalid

NCME_DUP_DEVICE
: the device could not be added because a device of the same device model name and unique device name is already instantiated in the system configuration

Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice model;
model.name = "D/41D";
model.next = NULL;

NCMDeviceUniqueName;
UniqueName.name = "D/41D at ID 0";
uniqueName.next = NULL;
```

```
//
// Execute
//

NCMRetCode ncmRc = NCM_AddDevice( &family, &model, &uniqueName );

if ( ncmRc == NCM_SUCCESS )
{
     ...
}
else
{
// Process error
     ...
}
...
```

See Also

- **NCM_DeleteEntry( )**
- **NCM_EnableBoard( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**
- **NCM_DetectBoards( )**
- **NCM_DetectBoardsEx( )**

NCM_ApplyTrunkConfiguration( )

Name: NCMRetCode NCM_ApplyTrunkConfiguration (NCMFamily Family, NCMDevice Device, NCMTrunkConfig* pTrunkConfig, NCMFeatureType* pMediaLoad, BYTE* pErrorMsg)

Inputs:

NCMFamily* pncmFamily	• data structure containing a device family name
NCMDevice* pncmDeviceUnique	• data structure containing a unique device name
NCMTrunkConfig* pTrunkConfig	• pointer to a data structure containing a trunk configuration
NCMFeatureType* pMediaLoad	• pre-defined sets of features
BYTE* pErrorMsg	• pointer to BYTE that the API fills with an error message on return. BYTE is defined as an unsigned character.

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Category: Modify configuration

Mode: Synchronous

■ Description

The **NCM_ApplyTrunkConfiguration( )** function is for trunk configuration. This function takes the Media load for the board and protocols for the trunks and then creates the configuration files such as PCD, FCD, and CONFIG. If the function call is successful, the newly generated configuration file names are set in the DCM's data storage (Registry) for the next download.

Note: As of System Release 6.0 on PCI for Windows, only the Intel® Dialogic® DMV1200BTEP board supports trunk configuration. As of System releases 6.0 and 6.1 on cPCI for Windows, the following boards support trunk configuration: DMV1200BTEC, DMV600BTEC, DMT160TEC, DMN160TEC. As of Intel NetStructure® HMP 2.0 for Windows, the Host Media Processing interface boards support trunk configuration.

Refer to the *Intel® DM3 Architecture Products Configuration Guide* for complete information about configuring DMV/B boards.

Refer to the *Intel NetStructure® Host Media Processing Software Configuration Guide* for information about configuring HMP interface boards.

Parameter	Description
pncmFamily	data structure containing the family name. The value of the data structure must be an installable family (that is, one that is supported in the DCM catalog). All instantiated families of devices in your current system configuration can be retrieved by **NCM_GetInstalledFamilies( )**.
pncmDeviceUnique	data structure containing the device's unique name. This name can be any string that sufficiently distinguishes multiple instantiations of the same device model. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.
pTrunkConfig	points to a list of trunk details. The value of TrunkName should be "Trunk1", "Trunk2", etc., and the value of the TrunkValue field should be a supported protocol value.
pMediaLoad	pre-defined sets of supported features. A media load consists of a configuration file set and associated firmware. Universal media loads support voice, fax, and conferencing resources simultaneously.
pErrorMsg	API provides as needed.

For DMV/B boards there are five protocol groupings available. You may assign the same protocol or different protocols for each trunk on the board, but all of the protocols must belong to the same group. For example, if you select a protocol from Group 1 for Trunk 1 on the board, then you must select a protocol from the same group for the remaining trunks on the board.

Group 1
4ess(T1), 5ess(T1), dms(T1), ni2(T1), ntt(T1), qsigt1(T1), qsige1(E1), net5(E1), cas(T1), r2mf(E1), t1cc(T1), e1cc(E1)

Group 2
dass2(E1) and dpnss(E1)

For Host Media Processing interface boards, there is one protocol grouping available. You may assign the same protocol or different protocols for each trunk on the board, but all of the protocols must belong to the same group. For example, if you select a protocol from Group 1 for Trunk 1 on the board, then you must select a protocol from the same group for the remaining trunks on the board.

Group 1
4ess(T1), 5ess(T1), ntt(T1), ni2(T1), dms(T1), qsigt1(T1), qsige1(E1), net5(E1), t1cc(T1), cas(T1), e1cc(E1), r2mf(E1)

Note: r2MF is supported on DN1601TEPHMP boards only.

■ Cautions

All the trunks must have protocols from the same group.

Errors

Possible errors for this function include:

NCME_TRUNK_CONFIG_FILE_NOT_FOUND
The CONFIG file that corresponds to the board's PCD file cannot be found.

NCME_TRUNK_CONFIG_FILE_PARSE
Trunk configuration entries were not found in the board's CONFIG file.

NCME_TRUNK_CONFIG_PROCESS_CREATION
Process required to create trunk configuration files failed.

NCME_TRUNK_CONFIG_SPECIFIC
Trunk configuration file creation failed.

NCME_TRUNK_CONFIG_FILE_ACTIVITY
System could not create or delete a required temporary file during trunk configuration.

NCME_TRUNK_CONFIG_INVALID_PROTOCOL
An invalid protocol has been passed in the trunk protocol list.

NCME_TRUNK_CONFIG_PROTOCOL_MISMATCH
Protocol mismatch. Refer to the Intel Dialogic documentation or DCM online help for the list of protocols that can be grouped together.

Example

The following example code is for a DMV/B series quad span board (i.e., DMV1200BTEP).

```
#include <stdio.h>
#include "ncmapi.h"
void main()
{
NCMRetCode ncmRetCode;
char buffer[300] = {0};

         NCMFamily family;
         family.name = "DM3";
         family.next = NULL;

         NCMDevice UniqueName;
         UniqueName.name = "DMV1200BTEP #1 in slot 2/10";
         UniqueName.next = NULL;

         NCMTrunkConfig ncmTruckConfig[4] = {0};

         NCMFeatureType ncmFeatureType = {0};


         ncmTruckConfig[0].TrunkName  = "Trunk1";
         ncmTruckConfig[0].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[0].next      = &(ncmTruckConfig[1]);

         ncmTruckConfig[1].TrunkName  = "Trunk2";
         ncmTruckConfig[1].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[1].next      = &(ncmTruckConfig[2]);

         ncmTruckConfig[2].TrunkName  = "Trunk3";
         ncmTruckConfig[2].TrunkValue = "5ESS(T1, Group 1)";
         ncmTruckConfig[2].next      = &(ncmTruckConfig[3]);
         ncmTruckConfig[3].TrunkName  = "Trunk4";
         ncmTruckConfig[3].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[3].next = NULL;

         strncpy(ncmFeatureType.MediaLoad, "ML10", MEDIA_LOAD_LENGTH);

         ncmRetCode = NCM_ApplyTrunkConfiguration(family,UniqueName , ncmTruckConfig,
&ncmFeatureType, reinterpret_cast<unsigned char*>(buffer));
         if (ncmRetCode != NCM_SUCCESS)
         {
           printf("Error calling NCM_ApplyTrunkConfiguration(). It returned: %d \n", ncmRetCode;
           printf( " Error Msg:  %s \n", buffer);
         }
              else
              {
                    printf("SUccessful calling NCM_ApplyTrunkConfiguration\n");
              }
              printf("press any key to exit\n");
              getchar();
}
...
```

■ See Also

- **NCM_DetectBoards( )**
- **NCM_DetectBoardsEx( )**
- **NCM_EnableBoard( )**
- **NCM_StartBoard( )**
- **NCM_StartDlgSrv( )**

NCM_Dealloc( )

Name:	NCMRetCode NCM_Dealloc(pncmString)	
Inputs:	NCMString* pncmString	• pointer to an NCMString data structure
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

Description

The **NCM_Dealloc( )** function deallocates memory allocated for NCMString data structures. For more information about memory allocation, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Parameter	Description
pncmString	points to the NCMString data structure occupying the memory to be freed. If the data structure is the first in a linked list, this function deallocates the memory occupied by all data structures in the list.

Cautions

To release memory that was allocated for one or more NCMValueEx data structure, use **NCM_DeallocValue( )** instead of **NCM_Dealloc( )**.

Errors

None.

Example

```
#include "NCMApi.h"

...

NCMFamily *pFamilies = NULL;

//get family list
NCMRetCode ncmRc = NCM_GetAllFamilies( &pFamilies);

if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
   ... //process error
}
```

```
//
// Execute
//

//Deallocate memory for family list
NCM_Dealloc(pFamilies);
...
```

See Also

- **NCM_DeallocValue( )**

NCM_DeallocValue( )

Name:	NCMRetCode NCM_DeallocValue(pncmValueEx)	
Inputs:	NCMValueEx *pncmValueEx	• pointer to an NCMValueEx data structure
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_DeallocValue( )** function deallocates memory allotted for NCMValueEx data structures. For more information about memory allocation, refer to the *Native Configuration Manager API Programming Guide*.

Parameter	Description
pncmValueEx	points to the NCMValueEx data structure occupying the memory to be freed. If the data structure is the first in a linked list, this function deallocates the memory occupied by all data structures in the list.

■ Cautions

None.

■ Errors

None.

■ Example

```
#include "NCMAPI.h"

...
//
//Prepare inputs
//

NCMFamily.family;
family.name = "DM3";
family.next = NULL;

NCMDevice device;
device.name = "VOIP-T1-1";
device.next = NULL;

NCMVariable variable;
variable.name = "PciID";
variable.next = NULL;
```

```
NCMValueEx *pValueEx = NULL;

NCMRetCode ncmRc = NCM_GetValueEx(&family, &device, & variable, *pValueEx);

if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
   ... //proecess error
}

...

//
// Execute
//

//Deallocate memory when through with it
NCM_DeallocValue(pValueEx);

...
```

See Also

- NCM_Dealloc( )

NCM_DeleteEntry( )

Name:	NCMRetCode NCM_DeleteEntry(pncmFamily, pncmDeviceUnique)	
Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a family name
	NCMDevice *pncmDeviceUnique	• pointer to a structure containing a unique device name
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

Description

The **NCM_DeleteEntry( )** function removes configuration information from the system configuration.

This function's scope depends upon which values are passed to the NCMFamily and NCMDevice pointers, as follows:

- To remove configuration information for an individual device: NCMFamily and NCMDevice should point to an instantiated device.
- To remove configuration information for a family: NCMFamily should point to a valid family and NCMDevice should point to NULL.
- To remove all configuration information: NCMFamily and NCMDevice should both point to NULL.

Parameter	Description
pncmFamily	points to an NCMString data structure containing the family name. The value of the data structure must be an instantiated family (i.e one that exists in your current system configuration).
pncmDeviceUnique	points to an NCMString data structure containing the device's unique name. The unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

Cautions

This function removes configuration information instantiated in the current system configuration. It has no effect on the installable families, devices and configuration parameters defined in the DCM catalog. For more information about the distinction between the system configuration and the

DCM catalog, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

NCME_GENERAL
: a problem occurred retrieving the data

Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

//
//execute
//

//delete a single device
NCMRetCode ncmRc = NCM_DeleteEntry(&family, &device);

if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
    ... //process error
{

// delete a family of devices
ncmRc = NCM_DeleteEntry( &family, NULL);

if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
   ... //process error
{

//delete all devices
ncmRc = NCM_DeleteEntry(NULL, NULL);
if (ncmRc == NCM_SUCCESS)
{
   ...
```

```
}
else
{
   ... //process error
{
```

■ See Also

None.

NCM_DetectBoards( )

Name:	NCMRetCode NCM_DetectBoards(pCallBackFunc, pnNumBrdsFound)	
Inputs:	GL_PROG_FUNC *pCallBackFunc	• pointer to a callback function
	int *pnNumBrdsFound	• pointer to a variable indicating the number of boards found through the auto-detect process
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_DetectBoards( )** function initiates a process that detects auto-detectable boards installed in the system.

Note: The **Ex** functions should be used where available (for example, **NCM_DetectBoardsEx( )** instead of **NCM_DetectBoards( )**). The non-Ex function is provided for backwards compatibility.

Parameter	Description
pCallbackFunc	points to a function that is defined as part of the client application and will be called by the NCM API to provide updates on the status of the auto-detection process (See **Cautions** below.)
pnNumBrdsFound	points to a variable where the number of boards found through the auto-detection process will be stored *Note:* The client application must declare the variable referenced by this pointer prior to calling **NCM_DetectBoards( )**.

■ Cautions

- The **NCM_DetectBoards( )** function is intended for non-DM3 architecture boards only. Use the **NCM_DetectBoardsEx( )** function for DM3 architecture boards.
- The callback function defined within the client application that is referenced by the **pCallBackFunc** pointer must follow the **int func_name (UINT percentageCompleted, const char *message)** format, where:
  - **percentageCompleted** is the address of an unsigned integer variable that the **NCM_DetectBoards( )** function will fill to indicate the progress of the detection process as a percentage of overall time required for detection.
  - **message** is a NULL-terminated character string containing a message to indicate detection progress status, such as “Detected Board #5”.
- If **pCallBackFunc** is NULL, then no message is sent to the client application from the detection process.

Errors

Possible errors for this function include:

NCME_REG_CALLBK
: callback function cannot be registered with initialization process

NCME_BRD_DETECT
: auto-detect failed

NCME_SP
: invalid state transition

NCME_CTBB_DEVICE_DETECTED
: error configuring the TDM bus

NCME_GENERAL
: a problem occurred retrieving the data

NCME_DETECTOR_LIB_NOT_FOUND
: there was an error loading the detector library

NCME_DETECTOR_FCN_NOT_FOUND
: there was an error getting the detector function

Example

```
#include "NCMApi.h"

int CallbackFunc(UINT uipercent, const char *message)
{
    //use the percentage and message to show status of the auto-detection process
    return TRUE;
}

...

int nNumBoardsFound = 0;

//
// Execute
//

NCMRetCode ncmRc = NCM_DetectBoards(CallBackFunc, &nNumBoardsFound);
...
```

See Also

- **NCM_DetectBoardsEx( )**
- **NCM_GetAUID( )**
- **NCM_StartBoard( )**
- **NCM_StopBoard( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

NCM_DetectBoardsEx( )

Name:	NCMRetCode NCM_DetectBoardsEx(pdetectInfo, pdetectResult)	
Inputs:	NCM_DETECTION_INFO *pdetectInfo	• pointer to a NCM_DETECTION_INFO data structure
	NCM_DETECTION_RESULT *pdetectResult	• pointer to a NCM_DETECTION_RESULT data structure
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

Description

The **NCM_DetectBoardsEx( )** function initiates a process that detects any auto-detectable boards installed in the system.

Parameter	Description
pdetectInfo	points to the NCM_DETECTION_INFO data structure
pdetectResult	points to a NCM_DETECTION_RESULT data structure

Note: **NCM_DetectBoardsEx( )** does not give list of .pcd files for DM3 boards. One of the data types in *NCMTypes.h* (NCM_MAX_FILEDESC) has changed from 81 characters to 2 * MAX_PATH (512). You must recompile your application if you use *NCMTypes.h* (directly or via other dependencies).

Cautions

The Intel® Dialogic® framework always performs a full system detection upon reboot. Any subsequent calls to this **NCM_DetectBoardsEx( )** function will complete the detection more quickly and return. In most of the cases, callback functions (NCM_CALLBACK_FCN and NCM_PCDFILE_SELECTION_FCN) might not be invoked.

Recommendations are as follows:

- Use NULL for both of these parameters: NCM_CALLBACK_FCN and NCM_PCDFILE_SELECTION_FCN. This will ensure that **NCM_DetectBoardsEx( )** behaves consistently, callback functions will be ignored, and default configurations will be selected.
- If your application is always running (for example, if it is running as a Windows service and set to automatic mode) then the application can listen to event service events generated during detection and act accordingly. Refer to the Event Service documentation for events that are generated during detection.

- To reconfigure the boards use the **NCM_ReconfigBoard( )** function.

Errors

Possible errors for this function include:

NCME_REG_CALLBK	callback function cannot be registered with initialization process
NCME_BRD_DETECT	auto-detect failed
NCME_SP	invalid state transition
NCME_CTBB_DEVICE_DETECTED	error configuring the TDM bus
NCME_GENERAL	a problem occurred retrieving the data
NCME_DETECTOR_LIB_NOT_FOUND	there was an error loading the detector library
NCME_DETECTOR_FCN_NOT_FOUND	there was an error getting the detector function
NCME_PCD_SELECTION	no PCD callback function present for DM3 boards

Example

```
#include "NCMAPI.h"

int CallBackFunc(UINT uipercent, const char *message)
{
   printf("%d percent complete \n Status message: %s \n", uipercent, message);
   return TRUE;
}

int GetPCDFile(NCMFileInfo *fileList, int NumFiles, NCMDevInfo devInfo, int *index)
{
   //if necessary, print out the devInfo, it contains information about the device for
   // (int i=0; I<numFiles; I++) displays the file index and file name

   printf ("index %d, file name = %s\n", i, fileList[i]);

   printf("please select file index");

   scanf("%d", index);
   return *index;
}

bool DetectBoardsEx( )
{
   NCMRetCode ncmRc = NCM_SUCCESS;
   NCM_DETECTION_INFO detectionInfo;
   NCM_DETECTION_RESULT detectionResult;
   detectionInfo.structSize = sizeof(NCM_DETECTION_INFO);
   detectionInfo.callbackFcn = (NCM_CALLBACK_FCN*) CallBackFunc;
   detectionInfo.pcdFileSelectionFcn = (NCM_PCDFILE_SELECTION_FCN*) GetPCDFile;
   ncmRc = NCM_DetectBoardsEx(detectionInfo, detectionResult);
```

```
        if (ncmRc ! = NCM_SUCCESS)
        {
           NCMErrorMsg *pncmErrorMsg = NULL;
           ncmRc = NCM_GetErrorMsg(ncmRc, *pncmErrorMsg);
           if (ncmRc== NCM_SUCCESS)
              printf ("NCM_DetectBoardsEx( ) returns error: %s \n", pncmErrorMsg->name);
           else
           {
              printf("NCM_DetectBoardsEx( ) returns unknown error \n";
              NCM_Dealloc(pncmErrorMsg);
              return false;
           }
        else
        {
           printf("NCM_DetectBoardsEx( ) success, detected %d boards \n",
                                                   detectionResult.totalDetectedBoards);
           return true;
        }
     }
```

■ See Also

- **NCM_DetectBoards( )**
- **NCM_GetErrorMsg( )**
- **NCM_StartBoard( )**
- **NCM_StopBoard( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

NCM_EnableBoard( )

Name:	NCMRetCode NCM_EnableBoard(pncmFamily, pncmDeviceUnique, bEnable)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a unique device name
	BOOL bEnable	• boolean indicator of enable or disable option
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

■ Description

The **NCM_EnableBoard( )** function enables or disables device initialization when the Intel Dialogic system is started. Any board that is disabled will not start when the system is started using **NCM_StartDlgSrv( )**.

The effect of this function depends on how you set the **bEnable** parameter:

- To allow device initialization: Set **bEnable** to TRUE.
- To prevent device initialization: Set **bEnable** to FALSE.

This function's scope depends upon what values are passed for the NCMFamily and NCMDevice pointers:

- To affect initialization for a device: **pncmFamily** and **pncmDeviceUnique** should point to an instantiated device.
- To affect initialization for a family: **pncmFamily** should point to a valid family and **pncmDeviceUnique** should be NULL.
- To affect initialization for all devices: **pncmFamily** and **pncmDeviceUnique** should both be NULL.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name. The value of the data structure must be an instantiated family (i.e one that exists in your current system configuration)
pncmDeviceUnique	points to the NCMString data structure containing the device's unique name. The unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.
bEnable	specifies whether devices should be enabled (TRUE) or disabled (FALSE)

■ Cautions

This function only affects devices instantiated in the current system configuration. It has no effect on the installable families, devices, and configuration parameters defined in the DCM catalog.

■ Errors

Possible errors for this function include:

NCME_SP
    invalid state transition

NCME_BAD_DATA_LOC
    the data destination is invalid or indeterminate

NCME_GENERAL
    a problem occurred retrieving the data

NCME_INVALID_INPUTS
    the values of the parameters supplied are invalid

NCME_DATA_NOT_FOUND
    requested data not found in NCM data storage

■ Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMFamily.family;
family.name = "D/x1D";
family.next = NULL;
```

```
NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

//
//execute
//

//enable a single device
NCMRetCode ncmRc = NCM_EnableBoard(&family, &device, TRUE);

if (ncmRc == NCM_SUCCESS)
{
 ...
}
else
{
   //process error
}

//enable a family of devices
NCMRetCode ncmRc = NCM_EnableBoard(&family, NULL, TRUE);

if (ncmRc == NCM_SUCCESS)
{
 ...
}
else
{
   //process error
}

...
```

See Also

- **NCM_AddDevice( )**
- **NCM_IsBoardEnabled( )**
- **NCM_DeleteEntry( )**

NCM_GetAllDevices( )

Name:	NCMRetCode NCM_GetAllDevices(pncmFamily, ppncmDeviceModel)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice **ppncmDeviceModel	• address of pointer where installable device model names will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetAllDevices( )** function gets a list of installable device models for a specific family. For information about using this function to fill all the data structures you need to instantiate and modify configuration parameter values, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Note: This function provides a list of installable device models from the DCM catalog. This function does not return a list of installed devices for your current system configuration. See the **NCM_GetInstalledDevices( )** for that functionality.

Parameter	Description
pncmFamily	points to an NCMString data structure containing the family name. The value of the data structure must be an installable family (i.e one that is supported in the DCM catalog)
ppncmDeviceModel	indicates the address of the pointer to the list to be filled with NCMString data structures containing installable device model names

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
  The DCM catalog could not be found

NCME_MEM_ALLOC
  memory could not be allocated to perform the function

NCME_GENERAL
    a problem occurred retrieving the data

NCME_INVALID_FAMILY
    the family name is invalid

NCME_INVALID_INPUTS
    the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice *pDevices = NULL;

//
//Execute
//

ncmRc = NCM_GetAllDevices(&family, &pDevices);

if (ncmRc == NCM_SUCCESS)
{
   NCMDevice * pModels = pDevices;
   while (pModels != NULL)
   {
      //process list
      ...
      pModels = pModels ->next;
   }
}

else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pDevices);
...
```

See Also

- **NCM_GetAllFamilies( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_AddDevice( )**

NCM_GetAllFamilies( )

Name:	NCMRetCode NCM_GetAllFamilies(ppncmFamily)	
Inputs:	NCMFamily **ppncmFamily	• address of a pointer in which the list of installable family names will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetAllFamilies( )** function gets a list of installable family names. For information about using this function to fill all the data structures you need to instantiate and modify configuration parameter values, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Note: This function provides a list of installable family names from the DCM catalog. This function does not return a list of family names for your current system configuration. See the **NCM_GetInstalledFamilies( )** for that functionality.

Parameter	Description
ppncmFamily	indicates the address of the pointer to the list to be filled with NCMString data structures containing installable family names

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
: The DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

NCMFamily *pFamilies = NULL;

//
//Execute
//

NCMRetCode ncmRc = NCM_GetAllFamilies(&pFamilies);

if (ncmRc == NCM_SUCCESS)
{
   NCMFamily * pCurrFamilies = pFamilies;
   while (pCurrFamilies != NULL)
   {
      //process list
      ...
      pCurrFamilies = pCurrFamilies ->next;
   }
}

else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pFamilies);
...
```

See Also

- **NCM_GetAllDevices( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_AddDevice( )**

NCM_GetAUID( )

Name:	NCMRetCode NCM_GetAUID(pncmFamily, pncmDevice, pNumAuid, pAuidList)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	int *pNumAuid	• pointer to the number of AUIDs to be returned by the function
	int *pAuidList	• pointer to the list of AUIDs to be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h devmap.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_GetAUID( )** function returns the Addressable Unit Identfier (AUID) for a given family or device. An AUID is a unique string of numbers that the Intel Dialogic system software assigns to each component with which communications can be initiated. In the context of the NCM API, devices are assigned AUIDs. You can use this function to get the AUID of an individual device or a list of AUIDs from all instantiated devices in a given family.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name
pnumAuid	points to the number of AUIDs to be returned by the function
pAuidList	points to where the list of AUIDs will be output

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
    invalid inputs supplied to the function

NCME_GENERAL
    a problem occurred retrieving the data

Example

```
#include "NCMApi.h"
#include "devmap.h"

//1--get AUID for an individual device
//prepare inputs

int fruAuid=0;
int numAuid=1;
NCMFamily ncmFamily = {"DM3", NULL};
NCMDevice ncmDevice = {"QS_T1", NULL};

//execute

ncmRc = NCM_GetAUID(&ncmFamily, &ncmDevice, &numAuid, &fruAuid);
if (ncmRc ==NCM_SUCCESS)
{
   //pring out AUID
   printf("The AUID for %s/%s is %d: \n", ncmFamily.name, ncmDevice.name, fruAuid);
}
else
   //process error

//2---get AUIDs for a family
//prepare inputs

int *pfruAuid = (int*)malloc(sizeof(int)*1);
int numAuid=1;
NCMFamily ncmFamily = {“DM3”, NULL};

//execute

do
{
   pfruAuid= (int*)realloc(pfruAuid, sizeof(int)*(numAuid));
   ncmRc = NCMGetAUID(&ncmFamily, NULL, &numAuid, pfruAuid);
} While (ncmRc == NCME_BUFFER_TOO_SMALL)

if (ncmRc == NCM_SUCCESS)
{
   //print out the AUIDs
   printf("The AUIDs for family %s are \n", ncmFamily.name);
   for (int I=0; I<numAuid; I++)
   {
      printf("AUID %d = %d \n", i, *pfruAuid++);
   }
}
else
   //process error
}
```

See Also

- NCM_GetFamilyDeviceByAUID( )

NCM_GetBridgeDeviceHMPClockMasterFallbackList( )

Name:	NCMRetCode NCM_GetBridgeDeviceHMPClockMasterFallbackList(ppFamily, ppDevice)	
Inputs:	NCMFamily **ppFamily	• address of the pointer to the family name of the bridge device clock master fallback list
	NCMDevice **ppDevice	• address of the pointer to the device name of the bridge device clock master fallback list
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	TDM bus	
Mode:	synchronous	

■ Description

Intel NetStructure® Host Media Processing (HMP) digital network interface boards have a bridge device that enables communication and media streaming between the HMP host and the boards on the CT Bus. The **NCM_GetBridgeDeviceHMPClockMasterFallbackList( )** function retrieves the bridge device HMP clock master fallback list. The bridge device HMP clock master fallback list has no relation to the CT Bus clock master fallback list. The bridge device HMP clock master fallback list is used to specify which bridge device provides the streaming synchronization to all Intel NetStructure Host Media Processing digital network interface boards in the system.

To set the bridge device HMP clock master fallback list, the application must call the **NCM_SetValueEx( )** function for each board that contains a bridge device. This function call should set the "BridgeDeviceHMPClockMasterFallbackNbrUserDefined" parameter to a number between 0 and 15, where 0 indicates the primary HMP clock master fallback bridge device, 1 indicates the secondary HMP clock master fallback bridge device, 2 indicates the third HMP clock master fallback bridge device and so on.

Refer to the *Intel NetStructure® Host Media Processing Interface Boards Configuration Guide* for more information about bridge devices.

Notes: *1.* The **NCM_GetBridgeDeviceHMPClockMasterFallbackList( )** is supported on Intel NetStructure HMP systems only.

2. **NCM_GetBridgeDeviceHMPClockMasterFallbackList( )** is a convenience function. The information returned by this function can also be obtained by calling the **NCM_GetValueEx( )** function for the **BridgeDeviceHMPClockMasterFallbackNbrResolved** parameter. The **NCM_GetValueEx( )** would need to be called for each board in the system that contains a bridge device.

3. Multiple bridge devices can have the same HMP clock master fallback number. In this case, the internal bridge controller selects the order of the bridge device fallback list.

Parameter	Description
ppfamily	address of the pointer to the family name of the bridge device clock master fallback list. The memory for this parameter is allocated by the NCM and needs to be deallocated by the application. This is typically done by calling the **NCM_Dealloc( )** function.
ppdevice	address of the pointer to the device name of the bridge device clock master fallback list. The memory for this parameter is allocated by the NCM and needs to be deallocated by the application. This is typically done by calling the **NCM_Dealloc( )** function.

Cautions

This function is only supported for Intel NetStructure Host Media Processing releases.

Errors

Possible errors for this function include:

NCME_MEM_ALLOC
memory could not be allocated to perform the function call

NCME_GENERAL
a problem occurred retrieving the data

Completion Event

Since the **NCM_GetBridgeDeviceHMPClockMasterFallbackList( )** runs in synchronous mode, a DLGC_EVT_BRIDGE_DEVICE_HMP_CLOCK_MASTER event issued on the BRIDGING_CHANNEL of the event service framework indicates function completion. Refer to the *Event Service API for Windows Operating Systems Library Reference* for information about the event service framework.

Example A

The example below shows how to get the bridge device HMP clock master fallback list:

```
# include "NCMApi.h"

...
//
//Prepare inputs
//

NCMFamily *pfamily = NULL;
NCMFamily *pdevice = NULL;
NCMVariable bridgeStatusVariable = {"BridgeDeviceStatus", NULL};
NCMValueEx *pbridgeStatusValueEx = NULL;

//
// Execute
//

NCMRetCode ncmRc = NCMGetBridgeDeviceHMPClockMasterFallbackList(&pfamily, &pdevice);
```

```
if(ncmRc == NCM_SUCCESS)
{
   NCMValue *pcurrentFamily = pfamily;
   NCMValue *pcurrentDevice = pdevice;
   while ((pcurrentFamily != NULL) && (pcurrentDevice != NULL))
   {
      //Process list
      //Check the bridge status of the bridge devices
      NCM_GetValueEx(pcurrentfamily,
                     pcurrentdevice,
                     &bridgeStatusVariable,
                     &bridgeStatusValueEx);

      NCM_DeallocValue(pbridgeStatusValueEx);
      pcurrentFamily = pcurrentFamily ->next;
      pcurrentDevice = pcurrentDevice ->next;
   }
}
else
{
   //process error
}

//deallocate memory
NCM_Dealloc(pfamily);
NCM_Dealloc(pdevice);

...
```

■ Example B

The example below shows how to set the bridge device HMP clock master fallback list:

```
#include "NCMTypes.h"
#include "NCMApi.h"

int main(void)
{
   NCMRetCode  ncmRc         = NCM_SUCCESS;
   NCMFamily   *pFamily      = {"DM3", NULL}
   NCMDevice   *pDevice0     = {T1E1_HEB 0"};
   NCMDevice   *pDevice1     = {T1E1_HEB 1};
   NCMVariable *pVariable    = {"BridgeDeviceHMPClockMasterFallbackNbrUserDefined", NULL);
   NCMValueEx  *pValueEx     = new NCMValueEx;
   int         intdatavalue0 = 0;
   int         intdatavalue1 = 1;

   pValueEx->structSize = sizeof(NCMValueEx);
   pValueEx->dataType   = NUMERIC;
   pValueEx->dataSize   = sizeof(pValueEx->dataValue);
   pValueEx->next       = NULL;
   pValueEx->dataValue  = &intdatavalue0;
   ncmrc                = NCM_SetValueEx(pFamily, pDevice0,
                                         pVariable, pValueEx);

   if (ncmRc != NCM_SUCCESS)
   {
      //process error
   }
   else
   {
      pValueEx->dataValue = &intdatavalue1;
      ncmRc               = NCM_SetValueEx(pFamily, pDevice1,
                                         pVariable, pValueEx);
   if (ncmRc != NCM_SUCCESS)
   {
      //process error
   }
   else
   {
      //process success
   }
   delete pValueEx;
   return 0;
}
```

■ See Also

None.

NCM_GetClockMasterFallbackList( )

Name:	NCMRetCode NCM_GetClockMasterFallbackList(pncmBus, pnumInList, ppfallbackList)	
Inputs:	NCMDevice *pncmBus	• pointer to a specific bus name
	int *pnumInList	• pointer to the total number of boards in the list to be returned
	NCMDevice **ppfallbackList	• address of a pointer to the list of devices in the clock master fallback list to be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	TDM bus	
Mode:	synchronous	

■ Description

The **NCM_GetClockMasterFallbackList( )** function returns the clock master fallback list. This function fills a pointer to a pointer with the beginning address of a list of devices in the clock master fallback list. In addition, this function also returns the total number of devices in the list. For more information about the clock master fallback list, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Parameter	Description
pncmBus	points to an NCMString data structure containing the specific bus name ("Bus-0")
pnumInList	points to the total number of devices in the clock master fallback list
ppfallbackList	address of a pointer to the list of devices to be returned

■ Cautions

- The current system software release supports a single TDM bus. Therefore, the bus name for the **pncmBus** parameter should always be "Bus-0".
- The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMDevice bus;
device.name = "Bus-0";
device.next = NULL;

NCMValue * pfallbackList = NULL;
int total;

//
//Execute
//

NCMRetCode ncmRc = NCMGetClockMasterFallbackList(&bus, &total, &pfallbackList);

if (ncmRc == NCM_SUCCESS)
{
   NCMValue * pCurrList = pfallbackList;
   while (pCurrList !=NULL)
   {
      //process list
      ...
      pCurrList = pCurrList->next;
   }
}
else
{
   //process error
   ...
}

//deallocate memory
NCM_Dealloc(pfallbackList);
```

■ See Also

- **NCM_GetTDMBusValue( )**
- **NCM_SetClockMasterFallbackList( )**
- **NCM_SetTDMBusValue( )**

NCM_GetCSPCountries( )

Name:	NCMRetCode NCM_GetCspCountries(ppncmCountries)
Inputs:	NCMValue **ppncmCountries • address of pointer where countries will be output
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	Query configuration
Mode:	synchronous

■ Description

The **NCM_GetCspCountries( )** function gets a list of supported countries from the DCM catalog.

This function fills a pointer to a pointer with the beginning address of a list of countries. The list represents those countries for which Intel telecom devices may be configured.

Parameter	Description
ppncmCountries	indicates the address of the pointer to the list to be filled with structures containing countries

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
the DCM catalog could not be found

NCME_MEM_ALLOC
memory could not be allocated to perform the function

NCME_GENERAL
a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

NCME_INVALID_INPUTS
values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

NCMValue *pCountries = NULL;

//
//Execute
//

NCMRetCode ncmRc = NCM_GetCspCountries(&pCountries);

if (ncmRc == NCM_SUCCESS)
{
   NCMValue * pCurrCountries = pCountries;
   while (pCurrCountries != NULL)
   {
      //process list
      ...
      pCurrCountries = pCurrCountries ->next;
   }
}

else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pCountries);
...
```

See Also

- NCM_GetCspCountryCode( )
- NCM_GetCspCountryName( )
- NCM_GetCspFeaturesValue( )
- NCM_GetCspFeaturesValueRange( )
- NCM_GetCspFeaturesVariables( )

NCM_GetCspCountryCode( )

Name:	NCMRetCode NCM_GetCspCountryCode(szCountryName, ppncmCode)	
Inputs:	char *szCountryName	• pointer to a country name
	NCMValue **ppncmCode	• address of pointer where the ISO country code will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Query configuration	
Mode:	synchronous	

■ Description

The **NCM_GetCspCountryCode( )** function gets the country code for a country from the DCM catalog. This function returns the ISO country code for a specified country via the address of a pointer that is passed to it.

Parameter	Description
szCountryName	points to an ASCII-Z string containing the country name you would like to get the code for
ppncmCode	indicates the address of the pointer that will point to the ISO country code

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
    the DCM catalog could not be found

NCME_MEM_ALLOC
    memory could not be allocated to perform the function

NCME_GENERAL
    a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
    requested data not found in NCM data storage

NCME_INVALID_INPUTS
values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

NCMValue *pCode = NULL;

//
//execute
//

NCMRetCode ncmRc = NCM_GetCspCountryCode("United States", &pCode);

if (ncmRc== NCM_SUCCESS)

{
   ...
}
else
{
   //process error
   ...
{

//deallocate memory when through with it
NCM_Dealloc(pCode);
...
```

See Also

- NCM_GetCSPCountries( )
- NCM_GetCspCountryName( )
- NCM_GetCspFeaturesValue( )
- NCM_GetCspFeaturesValueRange( )
- NCM_GetCspFeaturesVariables( )

NCM_GetCspCountryName( )

Name:	NCMRetCode NCM_GetCspCountryName(szCountryCode, ppncmName)	
Inputs:	char *szCountryCode	• pointer to a country code
	NCMValue **ppncmName	• address of pointer where the country name will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Query configuration	
Mode:	synchronous	

■ Description

The **NCM_GetCspCountryName( )** function gets the country name for a given ISO country code from the DCM catalog.

Parameter	Description
szCountryCode	points to an ASCII-Z string containing the country code you would like to get the name for
ppncmName	indicates the address of the pointer that will point to the country name

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
  the DCM catalog could not be found

NCME_MEM_ALLOC
  memory could not be allocated to perform the function

NCME_GENERAL
  a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
  requested data not found in NCM data storage

NCME_INVALID_INPUTS
  values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

NCMValue *pName = NULL;

//
//execute
//

NCMRetCode ncmRc = NCM_GetCspCountryName("US", &pName);

if (ncmRc== NCM_SUCCESS)

{
   ...
   //process error
}
else
{
   //process error
   ...
{

//deallocate memory when through with it
NCM_Dealloc(pName);
...
```

See Also

- **NCM_GetCSPCountries( )**
- **NCM_GetCspCountryCode( )**
- **NCM_GetCspFeaturesValue( )**
- **NCM_GetCspFeaturesValueRange( )**
- **NCM_GetCspFeaturesVariables( )**

NCM_GetCspFeaturesValue( )

Name:	NCMRetCode NCM_GetCspFeaturesValue(szCountryCode, szFeatures, pncmVariable, ppncmValue)	
Inputs:	char *szCountryCode	• pointer to a country code
	char *szFeatures	• pointer to a comma-separated list of country specific configuration parameters
	NCMVariable *pncmVariable	• pointer to a country specific configuration parameter listed in the **szFeatures** parameter
	NCMValue **ppncmValue	• address of the pointer to the value of the variable specified in the **pncmVariable** parameter
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Query configuration	
Mode:	synchronous	

■ Description

The **NCM_GetCspFeaturesValue**( ) function gets a country-specific parameter value from within the pointer to a comma-separated list of country-specific configuration parameters. A list of country-specific configuration parameters is contained in the **Features** configuration parameter. The value is either extracted from the string of values passed in the **szFeatures** parameter or found in the current system configuration.

Whether the country-specific configuration parameter value this function gets is from the DCM catalog or the system configuration depends on the value of the **szFeatures** parameter, as follows:

- To retrieve the default country-specific configuration parameter value from the DCM catalog, set **szFeatures** to an ASCII-Z string containing a pointer to a comma-separated list of country-specific configuration parameters.
- To retrieve the country-specific configuration parameter value from the system configuration, set the **szFeatures** to NULL.

Parameter	Description
szCountryCode	points to an ASCII-Z string containing an ISO country code. This value determines the set of country-specific parameters that are returned by the function. Each country has a unique set of configurable parameters.
szFeatures	points to an ASCII-Z string that contains either a pointer to a comma-separated list of country-specific configuration parameters or NULL

Parameter	Description
pncmVariable	points to the country-specific configuration parameter for which a value will be returned
ppncmValue	indicates the address of the pointer that will contain the value of the country-specific configuration parameter

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
the DCM catalog could not be found

NCME_MEM_ALLOC
memory could not be allocated to perform the function

NCME_GENERAL
a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

NCME_INVALID_INPUTS
values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMVariable variable;
variable.name = "Receive Gain";
variable.next = NULL;

NCMValue *pValue = NULL;

//
//Execute
//

NCMRetCode ncmRc = NCM_GetCspFeaturesValue("US", "RXGAIN_0, FREQRES_HIGH", &variable, &pValue);
```

```
if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pValue);
...
```

■ See Also

- **NCM_GetCSPCountries( )**
- **NCM_GetCspCountryCode( )**
- **NCM_GetCspCountryName( )**
- **NCM_GetCspFeaturesValueRange( )**
- **NCM_GetCspFeaturesVariables( )**

NCM_GetCspFeaturesValueRange( )

Name:	NCMRetCode NCM_GetCspFeaturesValueRange(szCountryCode, pncmVariable, ppncmRange)	
Inputs:	char *szCountryCode	• pointer to a country code
	NCMVariable *pncmVariable	• pointer to a country-specific configuration parameter
	NCMValue **ppncmRange	• address of a pointer where the range will be output
Returns:	NCM_SUCCESS if success NCM_error code if failure	
Includes:	NCMApi.h	
Category:	Query configuration	
Mode:	synchronous	

Description

The **NCM_GetCspFeaturesValueRange( )** function gets the value range for a country-specific configuration parameter.

Parameter	Description
szCountryCode	points to an ASCII-Z string containing an ISO country code
pncmVariable	points to the country-specific configuration parameter for which a range will be returned
ppncmRange	indicates the address of the pointer that will contain the value range of the country specific configuration parameter

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
 the DCM catalog could not be found

NCME_MEM_ALLOC
 memory could not be allocated to perform the function

NCME_GENERAL
 a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
 requested data not found in NCM data storage

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMVariable variable;
variable.name = "Receive Gain";
variable.next = NULL;

NCMValue *pRange=NULL;

//
//execute
//

NCMRetCode ncmRc = NCM_GetCspFeaturesValueRange("US", &variable, &pRange);

if (ncmRc== NCM_SUCCESS)
{
  NCMValue *pCurrRange = pRange;
  while (pCurrRange !=NULL)
  {
     ...
     pCurrRange = pCurrRange->next
   }
}
else
{
   //process error
   ...
{

//deallocate memory when through with it
NCM_Dealloc(pRange);
...
```

■ See Also

- NCM_GetCSPCountries( )
- NCM_GetCspCountryCode( )
- NCM_GetCspCountryName( )
- NCM_GetCspFeaturesValue( )
- NCM_GetCspFeaturesVariables( )

NCM_GetCspFeaturesVariables( )

Name:	NCM_RetCode NCM_GetCspFeaturesVariables(szCountryCode, ppncmVariables)	
Inputs:	char *szCountryCode	• pointer to a country code
	NCMVariable **ppncmVariables	• address of a pointer where the pointer to a comma separated list of country-specific configuration parameters will be stored
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Query configuration	
Mode:	synchronous	

■ Description

The **NCM_GetCspFeaturesVariables( )** function gets values for country-specific configuration parameters for a specific country. The country must be supported by the DCM catalog. Use the **NCM_GetCSPCountries( )** function to get a list of countries that are supported by the DCM catalog.

Parameter	Description
szCountryCode	points to an ASCII-Z string containing an ISO country code
ppncmVariables	indicates the address of the pointer that will point to the returned “Features” components

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
    the DCM catalog could not be found

NCME_MEM_ALLOC
    memory could not be allocated to perform the function

NCME_GENERAL
    a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
    requested data not found in NCM data storage

NCME_INVALID_INPUTS
values of the parameters supplied are invalid

Example

```
#include “NCMApi.h”

...

NCMVariable *pVariables = NULL;

//
//execute
//

NCMRetCode ncmRc = NCM_GetCspFeaturesVariables(“US”, &pVariables);

if (ncmRc== NCM_SUCCESS)
{
  NCMVariable *pCurrVariables = pVariables;
  while (pCurrVariables !=NULL)
  {
     ...
     pCurrVariables = pCurrVariables->next
   }
}
else
{
   //process error
   ...
{

//deallocate memory when through with it
NCM_Dealloc(pVariables);
...
```

See Also

- NCM_GetCSPCountries( )
- NCM_GetCspCountryCode( )
- NCM_GetCspCountryName( )
- NCM_GetCspFeaturesValue( )
- NCM_GetCspFeaturesValueRange( )

NCM_GetDialogicDir( )

Name:	NCMRetCode NCM_GetDialogicDir(szKey, size, pDlgcDir)	
Inputs:	char *szKey	• pointer to the Intel Dialogic path key
	int *size	• pointer to the returned buffer size
	char *pDlgcDir	• pointer to the directory to be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

Description

The **NCM_GetDialogicDir( )** function returns the corresponding Intel Dialogic directory. The function queries the Intel Dialogic software for the specified path key. For example, if the supplied path key value is "DLFWLPATH", then "...\Dialogic\data" would be returned.

Parameter	Description
szKey	points to an ASCII-Z string containing the specific path key value. The possible key values along with their corresponding path are defined as follows: • DIALOGICDIR – "...Dialogic\" • DLCFGPATH – "...\Dialogic\cfg" • DLFWLPATH – "...\Dialogic\data" • DLINFPATH – "...\Dialogic\inf" • DNASDKDIR – "...lDialogic\bin" • GFAX – "...\Dialogic\cpfax" (if Gammalink fax is installed)
size	indicates the buffer size allocated for the returned **pDlgcDir** parameter
pDlgDir	points to the directory to be returned

Cautions

The application needs to allocate memory for the directory to be returned and needs to provide the size of the buffer. If the buffer size is too small, the function will return a NCME_BUFFER_TOO_SMALL error.

Errors

Possible errors for this function include:

NCME_BUFFER_TOO_SMALL
buffer is too small

NCME_DATA_NOT_FOUND
: requested data not found in NCM data storage

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...

NCMRetCode ncmRc = NCM_SUCCESS;

//get directory
char *pDlg_data_path= NULL;
int bufferSize = MAX_PATH;
char pathKey[MAX_PATH] = {0};

strcpy(pathKey, "DLFWLPATH");

//
//execute
//

do
{
  pDlg_data_path = (char*)realloc(pDlg_data_path, bufferSize *sizeof(char));
  memset (pDlg_data_path, '/0', bufferSize);

  ncmRc= NCM_GetDialogicDir(pathKey, &buffersize, pDlg_data_path);
  bufferSize *=2;
}

  while (ncmRc ==NCME_BUFFER_TOO_SMALL);

if (ncmRc != NCM_SUCCESS && ncmRc !=NCME_BUFFER_TOO_SMALL)
  {
   //process error
   ...
  }

else if (pDlg_data_path != NULL)
{
  printf("dialogic dir path is %s\n", pDlg_data_path);
}

//clean up
if(pDlg_data_path)
  free(pDlg_data_path);

...
```

■ See Also

None.

NCM_GetDlgSrvStartupMode( )

Name:	NCMRetCode NCM_GetDlgSrvStartupMode(pncmStartupMode)	
Inputs:	NCMDlgSrvStartupMode *pncmStartupMode	• pointer to where the startup mode will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System	
Mode:	synchronous	

Description

The **NCM_GetDlgSrvStartupMode( )** function gets the startup mode of the Intel Dialogic or HMP system. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for more information about the system.

Parameter	Description
pncmStartupMode	points to the current startup mode setting for the Intel Dialogic or HMP system. Possible modes are as follows: • NCM_DLGSRV_AUTO – The Intel Dialogic or HMP system starts automatically whenever the system is re-started • NCM_DLGSRV_MANUAL – The Intel Dialogic or HMP system must be started manually • NCM_DLGSRV_DISABLED – The Intel Dialogic or HMP system is currently disabled • NCM_DLGSRV_STARTUP_UNDEFINED – The Intel Dialogic or HMP system startup mode is undefined

Cautions

None.

Errors

Possible errors for this function include:

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

NCME_OPENING_SCM
an error occurred while opening the service control manager

NCME_OPENING_DLGC_SVC
an error occurred while opening the Intel Dialogic system

NCME_QUERY_SVC_STATUS
: an error occurred while querying the status of the Intel Dialogic system

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

NCMDlgSrvStartupMode startupMode = NCM_DLGRV_AUTO;

//
// execute
//

NCMRetCode ncmRc = NCM_GetDlgSrvStartupMode(&startupMode);
if (ncmRc == NCM_SUCCESS)
{
   switch (startupMode)
   {
      case NCM_DLGSRV_AUTO;
         printf("Startup mode is set to Auto\n");
         break;
      case NCM_DLGSRV_MANUAL;
         printf("Startup mode is set to Manual\n");
         break;
      case NCM_DLGSRV_DISABLED;
         printf("Startup mode is set to Disabled\n");
         break;
      default:
         printf("Startup mode is undefined\n");
         break;
      } //endswitch

   }
else
{
  //process error
  ...
}
...
```

See Also

- **NCM_GetDlgSrvState( )**
- **NCM_GetDlgSrvStateEx( )**
- **NCM_SetDlgSrvStartupMode( )**

NCM_GetDlgSrvState( )

Name:	NCMRetCode NCM_GetDlgSrvState(pncmSrvState)
Inputs:	NCMDlgSrvState *pncmSrvState • pointer to where the system state will be output
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	System
Mode:	synchronous

■ Description

The **NCM_GetDlgSrvState( )** function gets the current state of the Intel Dialogic system. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for more information about the system. The **NCM_GetDlgSrvState( )** function outputs the value in the parameter passed to the values of SERVICE_STATUS structure. For details about SERVICE_STATUS structure, refer to the Microsoft Windows* documentation (MSDN Library at *http://msdn.microsoft.com/library*.)

Note: The **Ex** functions should be used where available (for example, **NCM_GetDlgSrvStateEx( )** instead of **NCM_GetDlgSrvStates**). The non-Ex function is provided for backwards compatibility.

Parameter	Description
pncmSrvState	points to the current state of the Intel Dialogic System. Refer to the Windows documentation for possible states of the system.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_OPENING_SCM
: an error occurred while opening the service control manager

NCME_OPENING_DLGC_SVC
: an error occurred while opening the Intel Dialogic system

NCME_QUERY_SVC_STATUS
: an error occurred while querying the status of the Intel Dialogic system

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...
//
// Execute
//

NCMDlgSrvState           serviceState = 0;
NCMRetCode     ncmRc = NCM_GetDlgSrvState( &serviceState );

if ( ncmRc == NCM_SUCCESS )
{
     if ( serviceState == SERVICE_CONTINUE_PENDING )
     {
          printf( "Continue Pending\n" );
     }
     else if ( serviceState == SERVICE_PAUSE_PENDING )
     {
          printf( "Pause Pending\n" );
     }
     else if ( serviceState == SERVICE_STOP_PENDING )
     {
          printf( "Stop Pending\n" );
     }
     else if ( serviceState == SERVICE_START_PENDING )
     {
          printf( "Start Pending\n" );
     }
     else if ( serviceState == SERVICE_RUNNING )
     {
          printf( "Running\n" );
     }
     else if ( serviceState == SERVICE_STOPPED )
     {
          printf( "Stopped\n" );
     }
     else if ( serviceState == SERVICE_PAUSED )
     {
          printf( "Paused\n" );
     }
     else
     {
          printf( "Unknown\n" );
     }
}
else
{     // process error
     ...
}
...
```

■ See Also

- NCM_GetDlgSrvStartupMode( )
- NCM_GetDlgSrvStateEx( )
- NCM_SetDlgSrvStartupMode( )

NCM_GetDlgSrvStateEx( )

Name:	NCMRetCode NCMGetDlgSrvStateEx(pncmSrvState)	
Inputs:	SERVICE_STATUS *pncmSrvState	• pointer to the Win32 SERVICE_STATUS data structure
Returns:	NCM_SUCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System	
Mode:	synchronous	

■ Description

The **NCM_GetDlgSrvStateEx( )** function gets the current state of the Intel Dialogic or HMP system. Refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide* for more information about the system. The **NCM_GetDlgSrvStateEx( )** function outputs the value in the parameter passed to the values of SERVICE_STATUS structure. For details about SERVICE_STATUS structure, refer to the Microsoft Windows documentation (MSDN Library at *http://msdn.microsoft.com/library*.)

This function returns the state of the system by filling the passed pointer.

Parameter	Description
pncmSrvState	points to the current state of the Intel Dialogic or HMP system. Refer to the Windows documentation for possible states of a service.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_OPENING_SCM
: an error occurred while opening the service control manager

NCME_OPENING_DLGC_SVC
: an error occurred while opening the Intel Dialogic or HMP system

NCME_QUERY_SVC_STATUS
: an error occurred while querying the status of the Intel Dialogic or HMP system

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...
//
// Execute
//

SERVICE_STATUS srvcStatus;

NCMRetCode ncmRc = NCM_GetDlgSrvStateEx( &srvcStatus );

if ( ncmRc == NCM_SUCCESS )
{
     if ( srvcStatus.dwCurrentState == SERVICE_CONTINUE_PENDING )
     {
          printf( "Continue Pending\n" );
     }
     else if ( srvcStatus.dwCurrentState == SERVICE_PAUSE_PENDING )
     {
          printf( "Pause Pending\n" );
     }
     else if ( srvcStatus.dwCurrentState == SERVICE_STOP_PENDING )
     {
          printf( "Stop Pending\n" );
     }
     else if ( srvcStatus.dwCurrentState == SERVICE_START_PENDING )
     {
          printf( "Start Pending\n" );
     }
     else if ( srvc.dwCurrentState == SERVICE_RUNNING )
     {
          printf( "Running\n" );
     }
     else if ( srvcStatus.dwCurrentState == SERVICE_STOPPED )
     {
          printf( "Stopped\n" );
     }
     else if ( srvcStatus.dwCurrentState == SERVICE_PAUSED )
     {
          printf( "Paused\n" );
     }
     else
     {
          printf( "Unknown\n" );
     }
}
else
{     // process error
     ...
}
...
```

■ See Also

- NCM_GetDlgSrvStartupMode( )
- NCM_GetDlgSrvState( )
- NCM_SetDlgSrvStartupMode( )

NCM_GetErrorMsg( )

Name:	NCMRetCode NCM_GetErrorMsg(ncmRcIn, ppncmErrorMsg)	
Inputs:	NCMRetCode ncmRcIn	• NCM return code
	NCMErrorMsg **ppncmErrorMsg	• address of a pointer where the error message will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

Description

The **NCM_GetErrorMsg( )** function gets the error message for a given NCM error code. Each function in the NCM API returns an error code indicating the success or failure of the function. The **NCM_GetErrorMsg( )** function accepts any one of the error codes and returns its associated text string.

Notes: ***1.*** Refer to the System Log of the Windows Event Viewer for a detailed explanation of generated NCM API error messages.

2. All NCM error codes are defined in the *NCMTypes.h* file.

Parameter	Description
ncmRcIn	specifies the return code whose error message should be returned
ppncmErrorMsg	indicates a pointer to a pointer to be filled with the error message

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
the values of the parameters supplied are invalid

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

■ Example

```
#include "NCMApi.h"

...

NCMFamily *pFamilies= NULL;

NCMRetCode ncmRc= NCM_GetAllFamilies(&pFamilies);

if (ncmRc == NCM_SUCCESS)
{
   ...
}
else
{
   //process error

   //execute
   ncmErrorMsg *pErrorMsg = NULL;
   ncmRc = NCM_GetErrorMsg(ncmRc, &pErrorMsg);
   if (ncmRc = NCM_SUCCESS)
    {
      printf("Failed to get families: %s\n", pErrorMsg->name);
    }

    //deallocate memory
    NCM_Dealloc(pErrorMsg);
}

//deallocate memory when through with it
NCM_Dealloc(pFamilies);
...
```

■ See Also

None.

NCM_GetFamilyDeviceByAUID( )

Name:	NCMRetCode NCM_GetAUID(fruAuid, ppncmFamily, ppncmDevice)	
Inputs:	int fruAuid	• AUID
	NCMFamily **pncmFamily	• pointer to the address where the family name will be returned
	NCMDevice **ppncmDevice	• pointer to the address where the device name will be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h devmap.h	
Category:	System administration	
Mode:	synchronous	

Description

The **NCM_GetFamilyDeviceByAUID( )** function returns device name that has been assigned a given Addressable Unit Identifier (AUID). The family name that the device belongs to is also returned by this function. An AUID is a unique string of numbers that the Intel Dialogic system software assigns to each component with which communications can be initiated. In the context of the NCM API, devices are assigned AUIDs.

Parameter	Description
fruAuid	indicates the AUID
pncmFamily	points to the address where the family name will be returned
pncmDevice	points to the address where the device name will be returned

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
    invalid inputs supplied to the function

NCME_GENERAL
    a problem occurred retrieving the data

■ Example

```
#include "NCMApi.h"
#include "devmap.h"

//prepare inputs

int fruAuid = 1223456;
NCMFamily *pncmFamily=NULL;
NCMDevice *pncmDevice=NULL’

//execute

ncmRc = NCM_GetFamilyDeviceByAuid(fruAuid, &pncmFamily, pncmDevice);

if (ncmRc ==NCM_SUCCESS)
{
   //print out the family/device name
   if (pncmFamily && pncmFamily ->neme && pncmDevice && pncmDevice->name)
   {
      printf("The family & device name for AUID %d are %s---%s: \n", fruAuid, pncmFamily->name
                                                           pncmDevice->name);
   }
   NCM_Dealloc(pncmFamily);
   NCM_Dealloc(pncmDevice);
}

else
  //process error
}
```

■ See Also

- **NCM_GetAUID( )**

NCM_GetInstalledDevices( )

Name:	NCMRetCode NCM_GetInstalledDevices(pncmFamily, ppncmDeviceUnique)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice **ppncmDeviceUnique	• address of pointer where unique names of installed devices will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetInstalledDevices( )** function queries your system configuration for all instantiated devices in a given family.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name. The value of the data structure must be a family that is included in your current system configuration.
ppncmDeviceUnique	indicates the address of the pointer to the list to be filled with NCMString data structures containing unique device names. The unique device names will be the same names used to add the devices to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
the values of the parameters supplied are invalid

NCME_INVALID_FAMILY
: the family name is invalid

Example

```
#include "NCMApi.h"

...

//
//prepare inputs
//

NCMFamily family;
family.name= "D/x1D";
family.next= NULL;

//
//Execute
//

NCMRetCode ncmRc = NCM_GetInstalledDevices(&family, &pDevices);

if (ncmRc == NCM_SUCCESS)
{
   NCMDevice *pCurrDevices = pDevices;
   while(pCurrDevices !=NULL)
   {
     //process list
     ...
     pCurrDevices = pCurrDevices->next;
    }

}

else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pDevices);
...
```

See Also

- NCM_GetAllDevices( )
- NCM_GetAllFamilies( )
- NCM_GetInstalledFamilies( )

NCM_GetInstalledFamilies( )

Name:	NCMRetCode NCM_GetInstalledFamilies(ppncmFamily)	
Inputs:	NCMFamily *ppncmFamily	• address of the pointer where the instantiated device families will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

Description

The **NCM_GetInstalledFamilies( )** function gets all instantiated families of devices in your current system configuration. This function fills a pointer to a pointer with the beginning address of a list of instantiated device families.

Parameter	Description
ppncmFamily	indicates the address of the pointer to the list to be filled with NCMString data structures containing family names from your current system configuration

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
    the values of the parameters supplied are invalid

Example

```
#nclude "NCMApi.h"

...

NCMFamily *pFamilies = NULL;

//
// execute
//

NCMRetCode ncmRc = NCM_GetInstalledFamilies(&pFamilies);
```

```
if (ncmRc == NCM_SUCCESS)
{
   NCMFamily *pCurrFamilies = pFamilies;
   while (pCurrFamilies !=NULL)
   {
      //process list
      ...
      pCurrFamilies = pCurrFamilies->next;
   }
}

else
{
   //process error
   ...
}

//deallocate memory when through with it
NCM_Dealloc(pFamilies);
...
```

■ See Also

- **NCM_GetAllDevices( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetInstalledDevices( )**

NCM_GetProperties( )

Name:	NCMRetCode NCM_GetProperties(pncmFamily, pncmDevice, ppncmProperties)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty **ppncmProperties	• address of the pointer where the device's properties will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetProperties( )** function gets the installable properties for a device. For information about using this function to fill all the structures you ned to instantiate and modify configuration parameter values, refer to the *Native Configuration Manager API for Windows Operating Systems Programming Guide*.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
ppncmProperties	specifies the address of the pointer to the list to be filled with NCMString data structures that contain the properties of a device

■ Cautions

- The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.
- The **pncmFamily** and **pncmDevice** parameters must reference information that is valid in the current DCM catalog.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty *     pProperties = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetProperties( &family, &device, &pProperties );

if ( ncmRc == NCM_SUCCESS )
{
     NCMProperty * pCurrProperties = pProperties;
     while ( pCurrProperties != NULL )
     {
          // Process list
          ...
          pCurrProperties = pCurrProperties ->next;
     }
}
else
{     // Process error
...
}

// Deallocate memory
NCM_Dealloc( pProperties );
...
```

See Also

- **NCM_GetAllDevices( )**

- **NCM_GetAllFamilies( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetPropertyAttributes( )**

NCM_GetPropertyAttributes( )

Name:	NCMRetCode NCM_GetPropertyAttributes(pncmFamily, pncmDevice, pncmProperty, pncmPropAttribs)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property section
	NCMPropertyAttributes *pncmPropAttribs	• pointer to the property's attributes
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetPropertyAttributes( )** function gets a property's attributes. The function queries the system configuration to determine whether an attribute is HIDDEN, VISIBLE or UNDEFINED.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmProperties	points to the NCMString data structure containing the property name
pncmPropAttribs	points to the property's attributes to be returned

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
the inputs to the function were invalid

NCME_DATA_NOT_FOUND
    requested data not found in NCM data storage

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty propert;
property.name = "Misc";
property.next = NULL;

NCMPropertyAttributes pPropAttribs;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetPropertyAttributes( &family, &device, &property,
                                                           &pPropAttribs );

if ( ncmRc == NCM_SUCCESS )
{
     //Process attributes
       ...
}
else
{     //Process error
    ...
}
...
```

See Also

- **NCM_GetAllDevices( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetProperties( )**

NCM_GetSystemState( )

Name:	NCMRetCode NCM_GetSystemState(pncmSystemState)	
Inputs:	NCMSystemState *pncmSystemState	• pointer to where the system service state will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System	
Mode:	Synchronous	

Description

The **NCM_GetSystemState( )** function returns the state of the Intel Dialogic or HMP system service by filling in the pointer that is passed to the function.

The function parameters are defined as follows:

Parameter	Description
pncmSystemState	pointer to where the system service state will be returned. Possible system service states are as follows: • NCM_SYSTEM_START_PENDING • NCM_SYSTEM_STOP_PENDING • NCM_SYSTEM_STOPPED • NCM_SYSTEM_RUNNING • NCM_SYSTEM_HALTED • NCM_SYSTEM_STATE_UNDEFINED

Cautions

None.

Errors

Possible errors for this function include:

NCME_OPENING_SCM
: an error occurred while opening the service control manager

NCME_OPENING_DLGC_SVC
: an error occurred while opening the Intel Dialogic or HMP system service

NCME_QUERY_SVC_STATUS
: an error occurred while querying the status of the Intel Dialogic or HMP system service

NCME_INVALID_INPUTS
invalid inputs

Example

```
#include "NCMApi.h"

...
NCMRetCode rc = NCM_GetSystemState(&state);
if (rc == NCM_SUCCESS)
{
   switch (state)
   {
      case NCM_SYSTEM_START_PENDING:
         printf("Intel Dialogic System in <Start_Pending> \n");
         break;
      case NCM_SYSTEM_STOP_PENDING:
         printf("Intel Dialogic System in <Stop_Pending> \n");
         break;
      case NCM_SYSTEM_STOPPED:
         printf("Intel Dialogic System is <Stopped> \n");
         break;
      case NCM_SYSTEM_RUNNING:
         printf("Intel Dialogic System is <Running> \n");
         break;
      case NCM_SYSTEM_HALTED:
         printf("Intel Dialogic System is <Halted> \n");
         break;
      case NCM_SYSTEM_STATE_UNDEFINED:
         printf("Intel Dialogic System is <Undefined> \n");
         break;
      default:
         printf("Intel Dialogic System is <Unknown> \n");
         break;
   }
}
else
{
   printf("Fail to get the Intel Dialogic system state \n");
}
...
```

See Also

- NCM_StartSystem( )
- NCM_StopSystem( )

NCM_GetTDMBusValue( )

Name:	NCMRetCode NCM_GetTDMBusValue(pncmBus, pvariable, ppvalue)	
Inputs:	NCMDevice *pncmBus	• pointer to a data structure containing a specific bus name
	NCMVariable *pvariable	• pointer to a data structure containing the variable name to be returned
	NCMValue **pvalue	• address of the pointer to the value to be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	TDM bus	
Mode:	synchronous	

■ Description

The **NCM_GetTDMBusValue( )** function gets the parameter value of the TDM bus. This function also allows you to retrieve the value of user defined and resolved variables in the TDM Bus family.

Parameter	Description
pncmBus	points to the NCMString data structure containing a specific bus name, for example "Bus-0"
pvariable	points to the variable name to be returned
ppvalue	specifies the address of the pointer where the variable value will be returned

■ Cautions

- The variable must be a valid parameter under the TDM bus configuration, otherwise the function returns an NCME_INVALID_INPUTS error.
- The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.
- The current system software release supports a single TDM bus. Therefore, the bus name for the **pncmBus** parameter should always be "Bus-0".

■ Errors

Possible errors for this function include:

NCME_CTBB_LIB
: a failure to load the CTBB library occurred

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMDevice bus;
device.name = "Bus-0";
device.next = NULL;

NCMVariable variable1;
variable.name = "Primary Master FRU (Resolved)";
variable.next = NULL;

NCMVariable variable2;
variable.name = "NETREF One FRU (Resolved)";
variable.next = NULL;

NCMValue *     pValue1 = NULL;
NCMValue *     pValue2 = NULL;

//
// Execute
//

//Get current Primary Master FRU
NCMRetCode     ncmRc = NCM_GetValue( &bus, &variable1, &pValue1 );

if ( ncmRc != NCM_SUCCESS )
{     // Process error
     ...
}

//Get current Net Ref FRU
NCMRetCode     ncmRc = NCM_GetValue( &bus, &variable2, &pValue2 );

if ( ncmRc != NCM_SUCCESS )
{     // Process error
     ...
}

// Deallocate memory
NCM_Dealloc( pValue1 );
NCM_Dealloc( pValue2 );
```

See Also

- NCM_GetClockMasterFallbackList( )
- NCM_SetClockMasterFallbackList( )
- NCM_SetTDMBusValue( )

NCM_GetValue( )

Name:	NCMRetCode NCM_GetValue(pncmFamily, pncmDevice, pncmProperty, pncmVariable, ppncmValue)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	NCMValue **ppncmValue	• address of a pointer where the parameter value will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetValue( )** function gets an instantiated value. This function enables you to determine the instantiated value of a configuration parameter in your current system configuration.

Note: The **Ex** functions should be used where available (for example, **NCM_GetValueEx( )** instead of **NCM_GetValue( )**). The non-Ex function is provided for backwards compatibility.

To get an instantiated configuration parameter value: **pncmDevice** must point to a unique device name. The unique device must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmProperty	points to an NCMString data structure containing the property name
pncmVariable	points to an NCMString data structure containing the configuration parameter name
ppncmValue	indicates the address of the pointer to be filled with the configuration parameter value

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_SP
: invalid state transition

NCME_GENERAL
: a problem occurred retrieving the data

NCME_BAD_INF
: there was an error parsing the DCM catalog

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty property;
property.name = "System";
property.next = NULL;

NCMVariable variable;
variable.name = "D41DAddress";
variable.next = NULL;

NCMValue *     pValue = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetValue( &family, &device, &property, &variable, &pValue );
```

```
if ( ncmRc == NCM_SUCCESS )
{
     if (pValue != Null && pValue ->name ! = Null)
     //use the value
}
else
{     // Process error
     ...
}

// Deallocate memory
NCM_Dealloc( pValue );

...
```

■ See Also

- NCM_GetValueEx( )
- NCM_GetValueRange( )
- NCM_GetValueRangeEx( )

NCM_GetValueEx( )

Name:	NCMRetCode NCM_GetValueEx(pncmFamily, pncmDevice, pncmVariable, ppncmValueEx)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter property section
	NCMValueEx **ppncmValueEx	• address of a pointer where the parameter value will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetValueEx( )** function gets an instantiated value. The return format is t. This function enables you to determine the instantiated value of a configuration parameter in the current system configuration.

Notes: *1.* The **NCM_GetValueEx( )** function returns different value types (numeric versus alphanumeric) for PCI bus number and PCI slot number for Springware and DM3 architecture boards.

2. The **NCM_GetValueEx( )** function does not return physical slot information for Intel Netstructure® IPT boards. It works on DM3 architecture boards. The difference is that for IPT boards, the parameter name is "Physical Slot" whereas for DM3 architecture boards, the parameter name is "Physical Slot Number."

To get an instantiated configuration parameter value: **pncmDevice** must point to a unique device name. The unique device must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).

Parameter	Description
pncmVariable	points to an NCMString data structure containing the configuration parameter name
ppncmValueEx	specifies the address of the pointer to the NCMValueEx data structure to be filled with the configuration parameter value

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_SP
: invalid state transition

NCME_GENERAL
: a problem occurred retrieving the data

NCME_BAD_INF
: there was an error parsing the DCM catalog

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMFamily family;
family.name = "DM3";
family.next = NULL;

NCMDevice device;
device.name = "VOIP-T1-1";
device.next = NULL;

NCMVariable variable;
variable.name = "PciID";
variable.next = NULL;

NCMValueEx *     pValueEx = NULL;
```

```
//
// Execute
//

NCMRetCode     ncmRc = NCM_GetValueEx( &family, &device, &variable, &pValueEx );
if ( ncmRc == NCM_SUCCESS)
{
     if ( pValueEx != NULL && pValueEx->dataValue !=NULL)
     {
          Switch (pValueEx -> dataType)
          {
          case ALPHANUMERIC:
               cout << (char*) pValueEx -> dataValue >> endl;
               break;
          case NUMERIC:
               cout <<(*((unsigned long*)pValueEx->dataValue)) <<endl;
               break;
          default:
               cout << "*** Bad datatype!!! ***" << endl;
               break;
          }
     }
}
else
{
     // Process error
     ...
}

// Deallocate memory when through
// with it

NCM_DeallocValue( pValueEx );

...
```

See Also

- NCM_GetValue( )
- NCM_GetValueRange( )
- NCM_GetValueRangeEx( )

NCM_GetValueRange( )

Name:	NCMRetCode NCM_GetValueRange(pncmFamily, pncmDevice, pncmProperty, pncmVariable, ppncmValues)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	NCMValue **ppncmValues	• address of a pointer where the parameter value range will be output
Returns:	NCM_SUCCESS if success	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

Description

The **NCM_GetValueRange( )** function provides the range of values that can be set for an installable configuration parameter. To determine the value of a configuration parameter instantiated in your current system configuration, use the **NCM_GetValue( )** or **NCM_GetValueEx( )** function.

Note: The **Ex** functions should be used where available (for example, **NCM_GetValueRangeEx( )** instead of **NCM_GetValueRange**). The non-Ex function is provided for backwards compatibility.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmProperty	points to an NCMString data structure containing the property name
pncmVariable	points to an NCMString data structure containing the configuration parameter name
ppncmValues	indicates the address of the pointer to be filled with the configuration parameter values

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_BAD_INF
: there was an error parsing the DCM catalog

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty property;
property.name = "System";
property.next = NULL;

NCMVariable variable;
variable.name = "D41DAddress";
variable.next = NULL;

NCMValue *     pRange = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetValueRange( &family, &device, &property, &variable, &pRange );
```

```
if ( ncmRc == NCM_SUCCESS )
{
     NCMValue * pCurrRange = pRange;
     while ( pCurrRange != NULL )
     {
          // Process list
               ...
          pCurrRange = pCurrRange->next;
     }
}
else
{     // Process error
     ...
}

// Deallocate memory
NCM_Dealloc( pRange );
...
```

■ See Also

- NCM_GetValue( )
- NCM_GetValueEx( )
- NCM_GetValueRangeEx( )

NCM_GetValueRangeEx( )

Name:	NCMRetCode NCM_GetValueRangeEx(pncmFamily, pncmDevice, pncmVariable, ppncmRangeEx)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	NCMValueEx **ppncmRangeEx	• address of pointer where the value range will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetValueRangeEx( )** function provides the range of values that can be set for an installable configuration parameter. To determine the value of a configuration parameter instantiated in your current system configuration, use the **NCM_GetValue( )** or **NCM_GetValueEx( )** function.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmVariable	points to an NCMString data structure containing the configuration parameter name
ppncmRangeEx	specifies the address of the pointer to the NCMValueEx data structure to be filled with the configuration parameter value range

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_BAD_INF
: there was an error parsing the DCM catalog

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMFamily family;
family.name = "DM3";
family.next = NULL;

NCMDevice device;
device.name = "VOIP-T1-1";
device.next = NULL;

NCMVariable variable;
variable.name = "PciID";
variable.next = NULL;

NCMValueEx *     pRangeEx = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetValueRangeEx( &family, &device, &variable, &pRangeEx );
if ( ncmRc == NCM_SUCCESS)
{
     NCMValueEx * pCurrRangeEx = pRangeEx;
     while (pCurrRangeEx != NULL)
     {     // Process list
          ...
          pCurrRangeEx = pCurrRangeEx->next;
     }      // endwhile
}
else
{     // Process error
     ...
}
```

```
// Deallocate memory when through
// with it
NCM_DeallocValue( pRangeEx );

...
```

See Also

- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_GetValueRange( )**

NCM_GetVariableAttributes( )

Name:	NCMRetCode NCM_GetVariableAttributes(pncmFamily, pncmDevice, pncmVariable, pncmVariableAttribs)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMVariable *pncmVariable	• pointer to a data structure containing a property section
	NCMVariableAttributes *pncmVariableAttribs	• pointer to a data structure containing the variable's attributes
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetVariableAttributes( )** function returns a configuration parameter's attributes. The function fills a pointer to a pointer with the beginning address of a list of variables for a particular property section.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing the device name for which the variables should be returned
pncmVariable	points to an NCMString data structure containing an individual variable
pncmVariableAttribs	points to where the variable's attributes are returned

■ Cautions

Global or family-level calls with this function are not supported. Default values are returned with the NCM_SUCCESS return code.

■ Errors

Possible errors for this function include:

NCME_DATA_NOT_FOUND
requested data not found in NCM data storage

NCME_INVALID_INPUTS
    invalid inputs

■ Example

None.

■ See Also

None.

NCM_GetVariables( )

Name:	NCMRetCode NCM_GetVariables(pncmFamily, pncmDevice, pncmProperty, ppncmVariables)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property section
	NCMVariable **ppncmVariables	• address of pointer where configuration parameters will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_GetVariables( )** function gets the parameters for a property section. It fills a pointer to a pointer with the beginning address of a list of configuration parameters for a particular property section. This function can be used to retrieve a list of all global configuration parameters from the DCM catalog by setting both the **pncmFamily** and the **pncmDevice** parameters to NULL.

This function provides configuration parameters that can be set for a device as defined in the DCM catalog. To determine the value of a configuration parameter instantiated in your system configuration, use **NCM_GetValue( )** or **NCM_GetValueEx( )**.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmProperty	points to the NCMString data structure containing the property name
ppncmVariables	specifies the address of the pointer to the list to be filled with configuration parameter data structures

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty property;
property.name = "System";
property.next = NULL;

NCMVariable * pVariables = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetVariables( &family, &device, &property, &pVariables );
if ( ncmRc == NCM_SUCCESS )
{
     NCMVariable * pCurrVariables = pVariables;
     while ( pCurrVariables != NULL )
     {     // Process list
          ...
          pCurrVariables = pCurrVariables ->next;
     }
}
else
{     // Process error
     ...
}

// Deallocate memory
NCM_Dealloc( pVariables );
...
```

See Also

- **NCM_GetVariableAttributes( )**

NCM_GetVersionInfo( )

Name:	NCMRetCode NCM_GetVersionInfo(NCMSysVersion *psysver)	
Inputs:	NCMSysVersion *psysver	• pointer to a data structure where the system version information will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_GetVersionInfo( )** function returns operating system (OS) and Intel Dialogic or HMP system software version information for local and remote computers.

Parameter	Description
psysver	points to an NCMSysVersion data structure to be filled with the system version information

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_REMOTE_REG_ERROR
    error opening the registry key of remote computer

■ Example

```
#include "NCMApi.h"
#include "NCMTypes.h"
...

//
//Execute
//

NCMRetCode ncmRc = NCM_GetVersionInfo (*psysver);

if ( ncmRc != NCM_SUCCESS )
{       //process error
     ...
}
...
```

See Also

- NCM_GetDialogicDir( )

NCM_IsBoardEnabled( )

Name:	NCMRetCode NCM_IsBoardEnabled(pncmFamily, pncmDeviceUnique, pbEnabled)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a unique device name
	BOOL *pbEnabled	• pointer to a Boolean
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

Description

The **NCM_IsBoardEnabled( )** function determines if a device is to initialized when the Intel Dialogic system is started. If a device is enabled, the address referenced by the **pbEnabled** pointer is set to TRUE; otherwise it is set to FALSE.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDeviceUnique	points to an NCMString data structure containing the device's unique name. The unique name must be the same name you used to add the device with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.
pbEnabled	points to a Boolean variable indicating that the device is enabled (TRUE) or disabled (FALSE)

Cautions

None.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
: requested data not found in NCM data storage

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

BOOL     bEnabled = TRUE;

//
// Execute
//

NCMRetCode     ncmRc = NCM_IsBoardEnabled( &family, &device, &bEnabled );

if ( ncmRc == NCM_SUCCESS )
{
     if ( bEnabled == TRUE )
     {
          ...
     }
     else
     {
          ...
     }
}
else
{     // Process error
     ...
}
...
```

■ See Also

- **NCM_EnableBoard( )**

NCM_IsEditable( )

Name:	NCMRetCode NCM_IsEditable(pncmFamily, pncmDevice, pncmProperty, pncmVariable, pbEditable)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDevice	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property section
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	BOOL *pbEditable	• pointer to a Boolean where output is placed
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Read configuration	
Mode:	synchronous	

■ Description

The **NCM_IsEditable( )** function determines if a configuration parameter can be edited. This function queries the DCM catalog to determine if the passed configuration parameter can be edited. If the configuration parameter can be edited; the address referenced by the **pbEditable** pointer is set to TRUE, otherwise it is set to FALSE.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a device name. The device name can either be a device model name or a unique device name (the unique device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function).
pncmProperty	points to the NCMString data structure containing the property name
pncmVariable	points to the NCMString data structure containing the configuration parameter
pbEditable	points to a Boolean specifying that the configuration parameter can be edited (TRUE) or cannot be edited (FALSE)

■ Cautions

The **pncmFamily**, **pncmProperty** and **pncmVariable** pointers must reference information that is valid in the DCM catalog.

Errors

Possible errors for this function include:

NCME_NO_INF
: the DCM catalog could not be found

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty property;
property.name = "System";
property.next = NULL;

NCMVariable variable;
variable.name = "D41DAddress";
variable.next = NULL;

BOOL     bEditable = TRUE;

//
// Execute
//

NCMRetCode     ncmRc = NCM_IsEditable( &family, &device, &property,
                                            &variable, &bEditable );

if ( ncmRc == NCM_SUCCESS )
{
     if ( bEditable == TRUE )
     {
          ...
     }
     else
     {
          ...
     }
}
```

```
else
{     // Process error
     ...
}
...
```

■ See Also

- NCM_GetProperties( )
- NCM_GetPropertyAttributes( )
- NCM_GetValue( )
- NCM_GetValueEx( )
- NCM_GetValueRange( )
- NCM_GetValueRangeEx( )
- NCM_GetVariableAttributes( )
- NCM_GetVariables( )

NCM_ReconfigBoard( )

Name: NCMRetCode NCM_ReconfigBoard(pncmFamily, pncmDevice, NCMDetectInfo *pdetectInfo, NCMDevice **ppncmDevice)

Inputs:	NCMFamily *pncmFamily	• pointer to the NCMFamily structure which stores the family name
	NCMDevice *pncmDevice	• pointer to the NCMDevice structure which stores the device name
	NCMDetectInfo *pdetectInfo	• pointer to the detection info structure
Output:	NCMDevice **ppncmNewDevice	• pointer to the address of the NCMDevice structure which the new device name would be populated

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMTypes.h, NCMApi.h

Category: Modify configuration

Mode: Synchronous

Description

The **NCM_ReconfigBoard( )** function lets you reconfigure a single DM3 architecture device.

Cautions

None.

Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
invalid inputs

NCME_GENERAL
a problem occured while retrieving the data

Example

```
#include "NCMApi.h"

int CallBackFunc( UINT uipercent, const char *message )

{
    // use the percentage and message
    // to show status of the auto-detection process
    return TRUE;
}
```

```
int GetPCDFile(NCMFileInfo *fileList, int numFiles, NCMDevInfo devInfo, int *index)
{
    // Pick a PCD file from fileList
    // index = picked one
    return TRUE;
}
...

//
// Prepare inputs
//
NCMFamily ncmFamily = { "DM3", NULL };
NCMDevice ncmDevice = { "QS_T1 0", NULL );
NCMDevice * pncmNewDevice = NULL;
NCM_DETECTION_INFO detectionInfo;
detectionInfo.structSize = sizeof( NCM_DETECTION_INFO );
detectionInfo.callbackFcn = &CallBackFunc;
detectionInfo. pcdFileCallbackFcn = &GetPCDFile;

//
// Execute
//
NCMRetCode ncmRc = NCM_ReconfigBoard(&ncmFamily, &ncmDevice, &detectionInfo, &pncmNewDevice);

if ( ncmRc == ERROR_SUCCESS )
{
   ...
}
else
{
    // process error
   ...
}

//clean up
NCM_Dealloc( pncmNewDevice );
...
```

See Also

- **NCM_DeleteEntry( )**
- **NCM_DetectBoards( )**
- **NCM_DetectBoardsEx( )**
- **NCM_EnableBoard( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**

NCM_RemoveBoard( )

Name:	NCMRetCode NCM_RemoveBoard(pncmFamily, pncmDevice)	
Inputs:	NCMFamily *pncmFamily	• pointer to the NCMFamily structure which stores the family name
	NCMDevice *pncmDevice	• pointer to the NCMDevice structure which stores the device name
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

■ Description

The **NCM_RemoveBoard**( ) function removes a board from the NCM database.

Parameter	Description
pncmFamily	points to an NCMString data structure containing the family name. The value of the data structure must be an instantiated family (i.e one that exists in your current system configuration).
pncmDevice	points to an NCMString data structure containing a device name

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_INVALID_INPUTS
invalid inputs

NCME_GENERAL
a problem occured while retrieving the data

■ Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//
```

```
NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDeviceName;
UniqueName.name = "D/41D at ID 0";
uniqueName.next = NULL;

//
// Execute
//

NCMRetCode ncmRc = NCM_AddDevice( &family, &model, &uniqueName );

if ( ncmRc == NCM_SUCCESS )
{
     ...
}
else
{
// Process error
     ...
}
...

NCMRetCode ncmRc = NCM_RemoveBoard( &family, &DeviceName );

if ( ncmRc == NCM_SUCCESS )
{
     ...
}
else
{
// Process error
     ...
}
...
```

■ See Also

None.

NCM_SetClockMasterFallbackList( )

Name:	NCMRetCode NCM_SetClockMasterFallbackList(pncmBus, pfallbackList)	
Inputs:	NCMDevice *pncmBus	• pointer to a data structure containing a specific bus name
	NCMDevice *pfallbackList	• pointer to a list of clock master fallback devices to be set
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	TDM bus	
Mode:	synchronous	

■ Description

The **NCM_SetClockMasterFallbackList( )** function sets the clock master fallback list. The function will issue a CTBB_USER_APPLY message to validate changes. If the Computer Telephony Bus Broker (CTBB) returns an error, then the list will not be set and previous values will remain unchanged.

Note: If only one device is defined in the list, this device will be the Primary Clock Master and the system will select the Secondary Clock Master. If no devices are defined in the list, the system will choose both the Primary and Secondary Clock Master.

The clock master fallback list is created in order of the user's preference. The first device listed will be the Primary Clock Master, the second device will be the Secondary Clock Master, the third device will be the next fallback clock master and each subsequent device listed will be considered by the system as a clock fallback master. The list will end with an NCMString=NULL.

Parameter	Description
pncmBus	points to an NCMString data structure containing the specific bus name ("Bus-0")
pfallbackList	points to a device list that will be set as the clock master fallback list

■ Cautions

- The current system software release supports a single TDM bus. Therefore, the bus name for the **pncmBus** parameter should always be "Bus-0".
- The data structure that is passed to the function must be in single-link list form.

■ Errors

Possible errors for this function include:

NCME_DATA_NOT_FOUND
    requested data not found in NCM data storage

NCME_CTBB_LIB
: a failure to load the CTBB library occurred

NCME_MEM_ALLOC
: memory could not be allocated to perform the function

NCME_GENERAL
: a problem occurred retrieving the data

NCME_INVALID_INPUTS
: the values of the parameters supplied are invalid

NCME_CTBB_USERAPPLY
: error updating the TDM bus parameters

■ Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMDevice bus;
device.name = "Bus-0";
device.next = NULL;

NCMDevice * pfallbackList;
NCMDevice * pCurrList = pfallbackList;

//Populate List
while ( )
{
     // Populate List
     ...
     pCurrList = pCurrList->next;
     pCurrList->next = NULL;
}

//
// Execute
//

NCMRetCode     ncmRc = NCM_SetClockMasterFallbackList( &bus, pfallbackList );

if ( ncmRc != NCM_SUCCESS )
{     // Process error
     ...
}
```

■ See Also

- **NCM_GetClockMasterFallbackList( )**
- **NCM_GetTDMBusValue( )**
- **NCM_SetTDMBusValue( )**

NCM_SetDlgSrvStartupMode( )

Name:	NCMRetCode NCM_SetDlgSrvStartupMode(ncmStartupMode)	
Inputs:	NCMDlgSrvStartupMode ncmStartupMode	• specifies the system startup mode to be set
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System	
Mode:	synchronous	

■ Description

The **NCM_SetDlgSrvStartupMode( )** function sets the Intel Dialogic system startup mode.

Parameter	Description
ncmStartupMode	indicates the startup mode for the system. Possible values are as follows: • NCM_DLGSRV_AUTO – The Intel Dialogic system starts automatically when the system reboots • NCM_DLGCSRV_MANUAL – The Intel Dialogic system must be started manually • NCM_DLGSRV_DISABLED – disable the Intel Dialogic system

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_OPENING_SCM
: an error occurred while opening the service control manager

NCME_OPENING_DLGC_SVC
: an error occurred while opening the system

NCME_CHANGE_SVC_STATUS
: an error occurred while changing the system status

NCME_UNKNOWN_SERVICE_TYPE
: the current service type is unknown

■ Example

```
#include "NCMApi.h"

...
```

```
//
// Execute
//

// Set startup mode of Dialogic service to Automatic
NCMRetCode     ncmRc = NCM_SetDlgSrvStartupMode( NCM_DLGSRV_AUTO );

if ( ncmRc != NCM_SUCCESS )
{     // process error
     ...
}
...
```

See Also

- **NCM_GetDlgSrvStartupMode( )**
- **NCM_GetDlgSrvState( )**
- **NCM_GetDlgSrvStateEx( )**

NCM_SetTDMBusValue( )

Name:	NCMRetCode NCM_SetTDMBusValue(pncmBus, pvariable, pvalue)	
Inputs:	NCMDevice *pncmBus	• pointer to a specific bus name
	NCMVariable *pvariable	• pointer to a data structure containing a variable name
	NCMValue *pvalue	• pointer to the value to be set
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	TDM bus	
Mode:	synchronous	

■ Description

The **NCM_SetTDMBusValue( )** function sets the values of the TDM bus. Variables under the TDM bus family with “UserDefined” in the parameter name can be changed by the user. Variables with “Resolved” in the parameter name cannot be modified by the user.

Parameter	Description
pncmBus	points to an NCMString data structure containing the specific bus name (“Bus-0”)
pvariable	points to an NCMString data structure containing the name of a variable
pvalue	points to an NCMString data structure containing the name of the value to be set

■ Cautions

- If you pass in a variable that cannot be modified, the function will return an NCME_ACCESS_DENIED error message.
- The current system software release supports a single TDM bus. Therefore, the bus name for the **pncmBus** parameter should always be “Bus-0”.

■ Errors

Possible errors for this function include:

NCME_CTBB_LIB
  a failure to load the CTBB library occurred

NCME_MEM_ALLOC
  memory could not be allocated to perform the function

NCME_GENERAL
  a problem occurred retrieving the data

NCME_INVALID_INPUTS
the values of the parameters supplied are invalid

NCME_CTBB_USERAPPLY
error updating the TDM bus parameters

NCME_ACCESS_DENIED
variable is read-only or not modifiable

Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMDevice bus;
device.name = "Bus-0";
device.next = NULL;

NCMVariable variable;
variable.name = "Derive Primary Clock From (User Defined)";
variable.next = NULL;

NCMValue value;
value.name = "InternalOscillator";
value.next = NULL;

//
// Execute
//

//set Primary Master FRU clock to Internal Oscillator
NCMRetCode     ncmRc = NCM_SetTDMBusValue( &bus, &variable, &value );

if ( ncmRc != NCM_SUCCESS )
{     // Process error
     ...
}

...
```

See Also

- NCM_GetClockMasterFallbackList( )
- NCM_GetTDMBusValue( )
- NCM_SetClockMasterFallbackList( )

NCM_SetValue( )

Name:	NCMRetCode NCM_SetValue(pncmFamily, pncmDeviceUnique, pncmProperty, pncmVariable, pncmValue)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family name
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a device name
	NCMProperty *pncmProperty	• pointer to a data structure containing a property
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	NCMValue *pncmValue	• pointer to a data structure containing the new value to be set
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

■ Description

The **NCM_SetValue( )** function sets a configuration parameter value. This function enables you to set the value of a configuration parameter in the system configuration. It does not enable you to add configuration parameter values to the DCM catalog.

Note: The **Ex** functions should be used where available (for example, **NCM_SetValueEx( )** instead of **NCM_SetValue( )**). The non-Ex function is provided for backwards compatibility.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDevice	points to an NCMString data structure containing a unique device name. The device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function.
pncmProperty	points to the NCMString data structure containing the property name
pncmVariable	points to the NCMString data structure containing the configuration parameter name
pncmValue	points to an NCMString data structure containing the new value to be set

■ Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

Errors

Possible errors for this function include:

NCME_SP
    invalid state transition

NCME_BAD_DATA_TYPE
    the data type of the variable is incorrect or indeterminable

NCME_BAD_DATA_LOC
    the data destination is invalid or indeterminate

NCME_CTBB_DEVICE_DETECTED
    error configuring the TDM bus

NCME_INVALID_INPUTS
    values of the parameters supplied are invalid

Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1D";
family.next = NULL;

NCMDevice device;
device.name = "D/41D-1";
device.next = NULL;

NCMProperty property;
property.name = "System";
property.next = NULL;

NCMVariable variable;
variable.name = "D41DAddress";
variable.next = NULL;

NCMValue value;
value.name = "d0000";
value.next = NULL;

//
// Execute
//

NCMRetCode     ncmRc = NCM_SetValue( &family, &device, &property,
                                          &variable, &value );

if ( ncmRc != NCM_SUCCESS )
{      // Process error
     ...
}

...
```

■ See Also

- NCM_AddDevice( )
- NCM_DeleteEntry( )
- NCM_EnableBoard( )
- NCM_GetValue( )
- NCM_GetValueEx( )
- NCM_SetValueEx( )

NCM_SetValueEx( )

Name:	NCMRetCode NCM_SetValueEx(pncmFamily, pncmDeviceUnique, pncmVariable, pncmValueEx)	
Inputs:	NCMFamily *pncmFamily	• pointer to a data structure containing a family
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a unique device name
	NCMVariable *pncmVariable	• pointer to a data structure containing a configuration parameter
	NCMValueEx *pncmValueEx	• pointer to a data structure containing the value to be set
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	Modify configuration	
Mode:	synchronous	

Description

The **NCM_SetValueEx( )** function instantiates a configuration parameter value.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDeviceUnique	points to an NCMString data structure containing a unique device name. The device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.
pncmVariable	points to the NCMString data structure containing the configuration parameter name
pncmValueEx	points to the NCMValueEx data structure containing the variable to be set

Cautions

The NCM API allocates memory for the data returned by this function. To avoid memory leaks, the client application must deallocate this memory by calling the **NCM_Dealloc( )** or **NCM_DeallocValue( )** functions.

■ Errors

Possible errors for this function include:

NCME_SP
: invalid state transition

NCME_GENERAL
: a problem occurred retrieving the data

NCME_DATA_NOT_FOUND
: requested data not found in NCM data storage

NCME_CTBB_DEVICE_DETECTED
: error configuring the TDM bus

NCME_INVALID_INPUTS
: values of the parameters supplied are invalid

■ Example

```
#include "NCMApi.h"

//
// Prepare inputs
//

NCMFamily family;
family.name = "DM3";
family.next = NULL;

NCMDevice device;
device.name = "VOIP-T1-1";
device.next = NULL;

NCMVariable      variable;
variable.name = "NetworkTimeout";
variable.next = NULL;

unsigned long      netTimeOut = 2;
NCMValueEx           valueEx;

valueEx.structSize = sizeof( NCMValueEx );
valueEx.dataType = NUMERIC;
valueEx.dataValue = &netTimeOut;
valueEx.dataSize = sizeof( netTimeOut );
valueEx.next = NULL;

//
// Execute
//

ncmRC = NCM_SetValueEx( &family, &device, &variable, &valueEx );


    if ( ncmRc == NCM_SUCCESS)
{
     ...
}
else
{     // Process error
     ...
}
...
```

■ See Also

- **NCM_AddDevice( )**
- **NCM_DeleteEntry( )**
- **NCM_EnableBoard( )**
- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_SetValue( )**

NCM_StartBoard( )

Name:	NCMRetCode NCM_StartBoard(pncmFamily, pncmDeviceUnique)	
Inputs:	NCMFamily *pncmFamily	• pointer to the data structure containing a device family name
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a unique device name
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_StartBoard( )** function starts an individual board. To start the entire Intel Dialogic or HMP system, use **NCM_StartDlgSrv( )**.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDeviceUnique	points to an NCMString data structure containing a unique device name. The device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCM_GENERAL
a problem occurred retrieving the data

Example

```
#include "NCMApi.h"
...
//
// Prepare inputs
//

NCMFamily family;
Family.name = "DM3";
Family.next = NULL;

NCMDevice device;
device.name = "QS_T1-1";
device.next = NULL;

//
// Execute
//
NCMRetCode        ncmRc = NCM_StartBoard(&family, &device);

if ( ncmRc == NCM_SUCCESS )
{        // process related functions calls
        ...
}
else
{        // process error
        ...
}
```

See Also

- NCM_DetectBoards( )
- NCM_DetectBoardsEx( )
- NCM_GetAUID( )
- NCM_GetFamilyDeviceByAUID( )
- NCM_StopBoard( )
- NCM_StartDlgSrv( )
- NCM_StopDlgSrv( )

NCM_StartDlgSrv( )

Name:	NCMRetCode NCM_StartDlgSrv(void)
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	System administration
Mode:	synchronous

■ Description

The **NCM_StartDlgSrv( )** function initiates the Intel Dialogic or HMP system. To start only one board, use **NCM_StartBoard( )**.

Notes: ***1.*** A successful completion code for this function (NCM_SUCCESS) only indicates that a start message was sent to the Intel Dialogic or HMP system. Use **NCM_GetDlgSrvState( )** or **NCM_GetDlgSrvStateEx( )** to determine whether or not the system actually started.

2. The **NCM_StartSystem( )** function is intended to replace **NCM_StartDlgSrv( )**, which will be discontinued in a future release. The **NCM_StartSystem( )** function was created to support Semi-Automatic mode.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_OPENING_SCM
: an error occurred opening the service control manager

NCME_OPENING_DLGC_SVC
: an error occurred opening the Intel Dialogic or HMP system

NCME_STARTING_DLGC_SVC
: an error occurred starting the Intel Dialogic or HMP system

■ Example

```
NCMRetCode ncmRc = NCM_SUCCESS;
ncmRc = NCM_StartDlgSrv();
```

```
if ( ncmRc == NCM_SUCCESS )
{
     SERVICE_STATUS srvcStatus;
     DWORD dwMilSecs = 1000;
     // loop wait for status to change
     while ( 1 )
     {
          ncmRc = NCM_GetDlgSrvStateEx( &srvcStatus );

          if ( ncmRc == NCM_SUCCESS )
          {
               if ( srvcStatus.dwWin32ExitCode == NO_ERROR &&
                         srvcStatus.dwCurrentState != SERVICE_RUNNING )
               {
                    Sleep( dwMilSecs );
               }

               if ( desiredState == SERVICE_STOPPED &&
                         srvcStatus.dwCurrentState == SERVICE_RUNNING )
               {
                    ncmRc = NCME_STOPPING_DLGC_SVC; // Error stopping Dialogic Service
                    return NCMToHresult(ncmRc);
               }

               if ( srvcStatus.dwCurrentState == SERVICE_RUNNING )
               {
                    break;
               }

               if (desiredState == SERVICE_RUNNING &&
                         srvcStatus.dwCurrentState == SERVICE_STOPPED )
               {
                    ncmRc = NCME_STARTING_DLGC_SVC; // Error starting Dialogic Service
                    // Handle error ...
                    return ncmRc;
               }

          }
     } // end while
}
```

■ See Also

- **NCM_DetectBoards( )**
- **NCM_DetectBoardsEx( )**
- **NCM_StartBoard( )**
- **NCM_StopBoard( )**
- **NCM_StopDlgSrv( )**

NCM_StartSystem( )

Name:	NCMRetCode NCM_StartSystem( )
Inputs:	None
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	System administration
Mode:	synchronous

■ Description

The **NCM_StartSystem( )** function starts all Intel telecom boards in the system. If your system is running in Manual mode, the **NCM_StartSystem( )** function will start the Intel Dialogic system service and start all Intel telecom boards in the system. If your system is in Semi-Automatic mode, the Intel Dialogic system service will run uninterrupted and a call to the **NCM_StartSystem( )** function will start all Intel telecom boards. Use the **NCM_GetSystemState( )** function to determine whether or not the system service is running.

Note: The **NCM_StartSystem( )** function is intended to replace **NCM_StartDlgSrv( )**, which will be discontinued in a future release. The **NCM_StartSystem( )** function was created to support Semi-Automatic mode.

■ Cautions

None

■ Errors

Possible errors for this function include:

NCME_GENERAL
  a problem occurred while starting the system service

■ Example

```
#include <windows.h>
#include "stdio.h"

...

   // Initialization
   NCMSystemState systemState = NCM_SYSTEM_STARTED;
   int      nNcmRc = 0;
   int      nMilSecs = 2000;    // 2 Seconds
   int      nTimeElapsed = 0;
   int      nMaxTime = 1000*60*5;    // 5 minutes
```

```
// Attempt to start the Intel Dialogic system
nNcmRc = NCM_StartSystem();
if (nNcmRc == 0)
{
   while ((1) && (nTimeElapsed < nMaxTime))
      {
      if (nNcmRc == NCM_GetSystemState(&systemState))
         {
           // If the System Service started successfully
           if (systemState == NCM_SYSTEM_STARTED)
              {
                 //..successfully started system service
                       break;
              }
           else
              {
                Sleep(nMilSecs);
                nTimeElapsed += nMilSecs;
              }
        }
     } //end while
} //end if

else
   {
   //process error and return error code to the user
     printf("Starting the Intel(R) Dialogic(R) system failed with error. Please Contact your
     System Administrator\n");
   return nNcmRc;
   }
   // Timed out before starting the System
   if(nTimeElapsed >= nMaxTime)
    {
      printf("Timed out trying to start the Intel(R) Dialogic(R) system. Please Contact your
       System Administrator\n");
    }
 return 0;

...
```

See Also

- NCM_StopSystem( )

NCM_StopBoard( )

Name:	NCMRetCode NCM_StopBoard(pncmFamily, pncmDeviceUnique)	
Inputs:	NCMFamily *pncmFamily	• pointer to the data structure containing a device family name
	NCMDevice *pncmDeviceUnique	• pointer to a data structure containing a unique device name
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Category:	System administration	
Mode:	synchronous	

■ Description

The **NCM_StopBoard( )** function stops an individual board. To stop the system service, use **NCM_StopDlgSrv( )**.

Parameter	Description
pncmFamily	points to an NCMString data structure containing a family name
pncmDeviceUnique	points to an NCMString data structure containing a unique device name. The device name must be the same name you used to add the device to the system configuration with the **NCM_AddDevice( )** function. ***Note:*** You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCM_GENERAL
: a problem occurred retrieving the data

■ Example

```
#include "NCMApi.h"
...
```

```
//
// Prepare inputs
//

NCMFamily family;
Family.name = "DM3";
Family.next = NULL;

NCMDevice device;
device.name = "QS_T1-1";
device.next = NULL;

//
// Execute
//
NCMRetCode        ncmRc = NCM_StopBoard(&family, &device);

if ( ncmRc != NCM_SUCCESS )
{
        // process error code
        ...
}
```

■ See Also

- **NCM_DetectBoards( )**
- **NCM_DetectBoardsEx( )**
- **NCM_GetAUID( )**
- **NCM_GetFamilyDeviceByAUID( )**
- **NCM_StartBoard( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

NCM_StopDlgSrv( )

Name:	NCMRetCode NCM_StopDlgSrv(void)
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	System administration
Mode:	synchronous

■ Description

The **NCM_StopDlgSrv( )** function stops the Intel Dialogic system. To stop only one board, use **NCM_StopBoard( )**.

Notes: ***1.*** A successful completion code (NCM_SUCCESS) only indicates that this function attempted to stop the system. Use **NCM_GetDlgSrvState( )** or **NCM_GetDlgSrvStateEx( )** to determine whether or not the system was actually stopped.

2. The **NCM_StopSystem( )** function is intended to replace **NCM_StopDlgSrv( )**, which will be discontinued in a future release. The **NCM_StopSystem( )** function was created to support Semi-Automatic mode.

■ Cautions

None.

■ Errors

Possible errors for this function include:

NCME_OPENING_SCM
an error occurred opening the service control manager

NCME_OPENING_DLGC_SVC
an error occurred opening the Intel Dialogic system

NCME_STOPPING_DLGC_SVC
an error occurred stopping the Intel Dialogic system

■ Example

```
#include "NCMApi.h"

...

//
// Execute
//

NCMRetCode     ncmRc = NCM_StopDlgSrv(  );
```

```
if ( ncmRc != NCM_SUCCESS )
{     // process error
     ...
}
```

See Also

- **NCM_StartBoard( )**
- **NCM_StopBoard( )**
- **NCM_StartDlgSrv( )**

NCM_StopSystem( )

Name:	NCMRetCode NCM_StopSystem(void)
Inputs:	None
Returns:	NCM_SUCCESS if success NCM error code if failure
Includes:	NCMApi.h
Category:	System administration
Mode:	Synchronous

Description

The **NCM_StopSystem( )** function stops all Intel telecom boards in the system. If your system is running in Semi-Automatic mode, the **NCM_StopSystem( )** function will stop all Intel telecom boards in the system, but will not stop theIntel Dialogic or HMP system service. If your system is running in Automatic or Manual mode, the **NCM_StopSystem( )** function will stop all Intel telecom boards and stop the Intel Dialogic or HMP system service.

Note: The **NCM_StopSystem( )** function is intended to replace **NCM_StartDlgSrv( )**, which will be discontinued in a future release. The **NCM_StopSystem( )** function was created to support Semi-Automatic mode.

Cautions

None.

Errors

Possible errors for this function include:

NCME_GENERAL
a problem occurred while stopping the system service

Example

```
#include <windows.h>
#include "stdio.h"

...

   // Initialization
   NCMSystemState systemState = NCM_SYSTEM_STOPPED;
   int      nNcmRc = 0;
   int      nMilSecs = 2000;    // 2 Seconds
   int      nTimeElapsed = 0;
   int      nMaxTime = 1000*60*5;    // 5 minutes
```

```
// Attempt to stop the Intel Dialogic system
nNcmRc = NCM_StopSystem();
if (nNcmRc == 0)
{
   while ((1) && (nTimeElapsed < nMaxTime))
      {
      if (nNcmRc == NCM_GetSystemState(&systemState))
         {
           // If the System Service stopped successfully
           if (systemState == NCM_SYSTEM_STOPPED)
              {
                 //..successfully stopped system service
                       break;
              }
           else
              {
                Sleep(nMilSecs);
                nTimeElapsed += nMilSecs;
              }
        }
     } //end while
} //end if

else
   {
   //process error and return error code to the user
     printf("Stopping the Intel(R) Dialogic(R) system failed with error. Please Contact your
     System Administrator\n");
   return nNcmRc;
   }
   // Timed out before stopping the System
   if(nTimeElapsed >= nMaxTime)
    {
      printf("Timed out trying to stop the Intel(R) Dialogic(R) system. Please Contact your
       System Administrator\n");
    }
 return 0;

...
```

■ See Also

- **NCM_StartSystem( )**
- **NCM_GetSystemState( )**

Events 3

This chapter contains information about events generated by the functions in the NCM API.

All functions in the NCM API operate in synchronous mode, so the start/completion of each function is difficult to determine. However, certain functions generate events that are transmitted via the Intel® Dialogic® system event notification framework's ADMIN_CHANNEL. Refer to the *Event Service API for Windows Operating Systems Library Reference* and the *Event Service API for Windows Operating Systems Programming Guide* for information about registering your application to receive events generated by select NCM library functions.

The following NCM library functions generate events that are carried on the event notification framework's ADMIN_CHANNEL or BRIDGING_CHANNEL (for HMP bridge devices):

- **NCM_StartBoard( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopBoard( )**
- **NCM_StopDlgSrv( )**

Data Structures 4

This chapter provides an alphabetical reference to the data structures used by the NCM library functions. These data structures are defined in *NCMTypes.h*. (For your convenience, *NCMApi.h* already includes *NCMTypes.h*.)

The following data structures are discussed:

- NCM_DETECTION_DETAILS 144
- NCM_DETECTION_INFO 145
- NCM_DETECTION_RESULT 146
- NCMString 147
- NCMSysVersion 148
- NCMTrunkConfig 150
- NCMValueEx. 149
- NCMVariableAttributes 151

NCM_DETECTION_DETAILS

```
typedef struct _NCM_DETECTION_DETAILS
{
      int   structSize;
      int   numDetectors;
      int   numBoardsDetected[256];
      int   returnCode[256];
      char   returnMsg[64][256];
      char   detector[64][256];
} NCM_DETECTION_DETAILS;
```

■ Description

The NCM_DETECTION_DETAILS data structure provides detailed information about the board detection process when the **NCM_DetectBoardsEx( )** function is invoked. Refer to the description of the NCM_DETECTION_RESULT, on page 146 for more information about the NCM_DETECTION_DETAILS data structure.

■ Field Descriptions

The fields of the NCM_DETECTION_DETAILS data structure are described as follows:

structSize
    size of the NCM_DETECTION_DETAILS data structure

numDetectors
    number of board detectors

numBoardsDetected
    number of boards detected

returnCode
    detector return code

returnMsg
    detector returned message

detector
    board detector name

NCM_DETECTION_INFO

```
typedef struct _NCM_DETECTION_INFO
{
      int  structSize;
      NCM_CALLBACK_FCN *callbackFcn;
      NCM_PCDFILE_SELECTION_FCN * pcdFileSelectionFcn;
} NCM_DETECTION_INFO;
```

Description

The NCM_DETECTION_INFO data structure provides information for the **NCM_DetectBoardsEx( )** function. This data structure contains the structure size and the address of the following two callback functions, both of which are defined in the *NCMTypes.h* file:

- NCM_CALLBACK_FCN
- NCM_PCDFILE_SELECTION_FCN

Field Descriptions

The fields of the NCM_DETECTION_INFO data structure are described as follows:

structSize
: size of the NCM_DETECTION_INFO data structure

callbackFcn
: address of the callback function

pcdFileSelectionFcn
: address of the PCD file callback function

NCM_DETECTION_RESULT

```
typedef struct _NCM_DETECTION_RESULT
{
      int structSize;
      int totalDetectedBoards;
      NCM_DETECTION_DETAILS returnInfo;
} NCM_DETECTION_RESULT;
```

Description

The NCM_DETECTION_RESULT data structure returns the results of the board detection procedure after the **NCM_DetectBoardsEx( )** function has been invoked.

Field Descriptions

The fields of the NCM_DETECTION_RESULT data structure are described as follows:

structSize
: size of the NCM_DETECTION_RESULT data structure

totalDetectedBoards
: total number of boards detected

returnInfo
: returned information (NCM_DETECTION_DETAILS data structure)

NCMString

```
typedef struct NCMString
{
      char *name;
      struct NCMString *next;
} NCMString;
```

Description

The NCMString data structure defines most variables used by the NCM library functions. All of following are aliases for NCMString:

- NCMFamily
- NCMDevice
- NCMProperty
- NCMValue
- NCMVariable
- NCMErrorMsg

Field Descriptions

The fields of the NCMString data structure are described as follows:

name
: string that defines the name of a particular data type (for example "DM3" for NCMFamily, "QS_T1" for NCMDevice etc.)

next
: points the next NCMString data structure in a linked list (if applicable)

NCMSysVersion

```
typedef struct _NCMSysVersion
{
      char szOSName[MAX_PATH];
      char szOSVersion[MAX_PATH];
      char szOSBuild[MAX_PATH];
      char szOSType[MAX_PATH];
      char szOSSvcPack[MAX_PATH];
      char szDSSVersion[MAX_PATH];
      char szDSSRelease[MAX_PATH];
      char szDSSBuild[MAX_PATH];
      char szDSSSvcPack[MAX_PATH];
} NCMSysVersion;
```

■ Description

The NCMSysVersion data structure defines the Operating System and Intel® Dialogic® System Software version information. This data structure is when the **NCM_GetVersionInfo( )** is invoked.

■ Field Descriptions

The fields of the NCMSysVersion data structure are described as follows:

szOSName
: name of the operating system

szOSVersion
: version of the operating system

szOSBuild
: operating system build

szOSType
: type of operating system

szOSSvcPack
: installed operating system service packs installed

szDSSVersion
: Intel Dialogic system software version

szDSSRelease
: Intel Dialogic system software release

szDSSBuild
: Intel Dialogic system software build

szDSSSvcPack
: Intel Dialogic system software service packs installed

NCMValueEx

```
typedef struct _NCMValueEx
{
      int                 structSize;
      NCMDataType         dataType;
      void                *dataValue;
      int                 dataSize;
      struct _NCMValueEx  *next;
} NCMValueEx;
```

Description

The NCMValueEx data structure defines configuration parameter values for use by the NCM API extended functions (**NCM_GetValueEx( )**, **NCM_GetValueRangeEx( )**, etc.).

Field Descriptions

The fields of the NCMValueEx data structure are described as follows:

structSize
: size of the NCMValueEx data structure

dataType
: enumerated (enum) type to signify the type of data held by the variable. Possible data types, as defined in *NCMTypes.h*, are as follows:

- UNDEFINED
- NUMERIC
- ALPHANUMERIC
- NCMFILE (to be used by filenames)

dataValue
: a buffer that holds the variable

dataSize
: size of the buffer allocated to hold the data (dataValue field)

next
: points to the next NCMValueEx data structure in a linked list (if applicable)

NCMTrunkConfig

```
typedef struct _NCMTrunkConfig
{
     char * TrunkName;
     char * TrunkValue;
     struct _NCMTrunkConfig * next;
} NCMTrunkConfig;
```

■ Description

This structure is used to pass the information needed for trunk configuration such as Media Load information for the board and protocols for the trunks.

■ Field Descriptions

The fields of the NCMTrunkConfig data structure are described as follows:

TrunkName
: For passing the Media Load, the value of this field should be Media Load. For passing the protocols for the trunks, the value of this filed should be Trunk *n*, where *n* is the number of trunks supported for the board.

TrunkValue
: If MediaLoad is the value for the TrunkName field, this field should have a supported media load for the board. Otherwise, it should have a supported protocol for the board.

NCMVariableAttributes

```
typedef struct _NCMVariableAttributes
{
      unsigned int           structSize;
      NCMDataType            dataType;
      int                    radix;
      NCMVariableDomainType  domainType;
      NCMVariableVisibleType visibleType;
      NCMVariableEditType    editType;
} _NCMVariableAttributes;
```

Description

The NCMVariableAttributes data structure defines the attributes of a configuration parameter. This data structure is filled when the **NCM_GetVariableAttributes( )** function is invoked.

Field Descriptions

The fields of the NCMVariableAttributes data structure are described as follows:

structSize
: size of the NCMVariableAttributes data structure

dataType
: enumerated (enum) type to signify the type of data held by the variable attribute. Possible data types, as defined in *NCMTypes.h*, are as follows:

- UNDEFINED
- NUMERIC
- ALPHANUMERIC
- NCMFILE (to be used by filenames)

radix
: radix of the variable

domainType
: enumerated (enum) type to signify the domain/range of the variables valid settings. Possible domain/range types, as defined in *NCMTypes.h*, are as follows:

- NCM_DOMAIN_UNDEFINED
- NCM_DOMAIN_OPEN
- NCM_DOMAIN_CLOSE

visibleType
: enumerated (enum) type to signify whether or not the variable is visible. Possible visibility types, as defined in *NCMTypes.h*, are as follows:

- NCM_VIS_UNDEFINED
- NCM_VARIABLE_VISIBLE
- NCM_VARIABLE_HIDDEN

editType
: enumerated (enum) type to signify whether the variable is read-only (RO) or read-write (RW). Possible edit types, as defined in *NCMTypes.h*, are as follows:

- NCM_ACC_UNDEFINED
- NCM_VARIABLE_RW
- NCM_VARIABLE_RO

Error Codes 5

This chapter lists the error codes that may be returned by the NCM library functions.

If a library function fails, use the **NCM_GetErrorMsg( )** function to return the error message. The following errors can be returned by the **NCM_GetErrorMsg( )** function:

NCME_ACCESS_DENIED
: denied access error (configuration parameter may be read-only)

NCME_ADD_DEVICE
: attempt to add device failed

NCME_BAD_DATA_LOC
: destination of data (i.e., global, family, or device level) could not be determined

NCME_BAD_DATA_TYPE
: data type of variable is incorrect or indeterminable

NCME_BAD_INF
: error parsing the .inf file

NCME_BRD_DETECT
: error auto-detecting boards

NCME_BUFFER_TOO_SMALL
: allocated buffer is too small

NCME_CTBB_DEVICESDETECTED
: re-detection of devices failed

NCME_CTBB_LIB
: *CTBBFace.dll* file is either not in the system or is the incorrect version

NCME_CTBB_USERAPPLY
: error updating TDM bus settings

NCME_DATA_NOT_FOUND
: data not found

NCME_DETECTOR_LIB_NOT_FOUND
: error loading detector library

NCME_DETECTOR_FCN_NOT_FOUND
: error calling detector function

NCME_DUP_DEVICE
: attempt to add a duplicate device name

NCME_FAIL_TO_CONFIGURE_BUS
: failure to configure TDM bus

NCME_FAIL_TO_SET_PRIMARY
: device could not be set to primary clock master

NCME_FAIL_TO_SET_SECONDARY
: device could not be set to secondary clock master

NCME_GENERAL
: general error

NCME_INVALID_ARG
: invalid version

NCME_INVALID_BUFF
: received an invalid buffer

NCME_INVALID_DEVICE
: invalid device name

NCME_INVALID_FAMILY
: invalid family name

NCME_INVALID_INPUTS
: invalid function inputs

NCME_INVALID_THIRDPARTY_DEVICE
: specified third party device does not exist

NCME_MEM_ALLOC
: memory allocation error

NCME_MISSING_BUS_CAPABILITIES
: invalid TDM bus capabilities

NCME_MULTIPLE_PCDS
: multiple .pcd files exist

NCME_NO_INF
: .inf files could not be found

NCME_NO_RESOURCES
: no system resources available

NCME_NO_TIMESLOT
: specified time slots queried do not exist

NCME_OPENING_DLGC_SVC
: error opening the Intel Dialogic system

NCME_OPENING_SCM
: error opening the service control manager

NCME_PCD_SELECTION
: no .pcd file was selected for DM3 boards

NCME_QUERY_SVC_STATUS
: error querying the Intel Dialogic system status

NCME_REG_CALLBK
: error registering a callback function with GENLOAD library

NCME_RELEASE_TIMESLOT
: failed to release the specified timeslots

NCME_REMOTE_REG_ERROR
: error opening the Intel Dialogic key in the remote machine registry

NCME_SETTING_DEFAULTS
: error occurred while setting the default values

NCME_SP
: invalid state transition

NCME_STARTING_DLGC_SVC
: error occurred while starting the Intel Dialogic system

NCME_STOPPING_DLGC_SVC
: error occurred while stopping the Intel Dialogic system

NCME_SYSTEMERROR
: lack of system resources

NCME_UNAVAILABLE_TIMESLOT
: requested time slot is not available

NCME_UNKNOWN_SERVICE_TYPE
: software cannot determine the Intel Dialogic system type

Index

A
ADMIN_CHANNEL 141
aliases for NCMString 147
auto-detection of boards 30

B
Bus-0 86

C
clock master fallback list 116
CTBB 116
CTBB_USER_APPLY 116

D
DCM catalog 17, 28, 38
deallocating memory 23
detecting DM3 boards 30
devmap.h 42, 73
disable a device 35

E
enable a device 35
enum 149, 151
error messages 153
event notification framework 141
extended functions 149

F
function syntax 15

H
HIDDEN property attribute 82

I
installable devices 8
instantiated devices 8

M
message variable 30

N
NCM_CALLBACK_FCN 145
NCM_PCDFILE_SELECTION_FCN 145
NCMRetCode 15
NCMSysVersion 105
NCMTypes.h 15, 145, 151
NCMValueEx 97
number of TDM busses supported 86

O
Operating System version 105
order of clock master fallback list 116

P
PCD file 145
percentageCompleted variable 30
Primary Clock Master 116

R
Resolved TDM bus parameters 120

S
Secondary Clock Master 116
supported TDM busses 86
system service startup modes 118
system software version 105

T
TDM bus parameters
    Resolved 120
    UserDefined 120

U
UNDEFINED property attribute 82

unique device name 27
UserDefined TDM bus parameters 120

V

VISIBLE property attribute 82