Customization Tools for Installation and Configuration

for Windows* NT/2000/XP

05-1103-008

Publication Date: January 2003

Part Number: 05-1103-008

Intel Converged Communications
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 Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices**, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Table of Contents

List of Figures

List of Tables

1. Overview

1.1. Features

This document provides instructions for using the Intel® Dialogic Configuration Manager (DCM) API and the Intel® Dialogic OEM Installation Tool (DOIT). The features of these two products are outlined in the subsections that follow.

1.1.1. The DCM API

The DCM API consists of a library of functions for creating and manipulating the configuration data necessary to initialize Intel® Dialogic devices and to control their operation.

You can use the DCM API to develop your own customized configuration data management applications. For example, in conjunction with the Intel® Dialogic OEM Installation Tool (DOIT), you can create automated installation and configuration programs that require no user intervention.

The features available through the DCM API include:

- **Modifying Configuration Data:** you can set configuration parameter values.
- **Querying Configuration Data:** you can determine what devices can be installed and what configuration parameter values can be set in a given Intel® Dialogic System Software release.
- **Working with the System Service:** you can start and stop the system service, set its startup mode and check its status.
- **Single Board Stop/Start:** you can stop and start a single DM3 PCI (H.100) or CompactPCI (cPCI) (H.110) board in the system without affecting system operation. In the case of cPCI boards, this allows you to perform a Hot Swap.
- **Clock Master Fallback List:** you can set the clock master fallback list. The clock master fallback list defines master capable devices in a preferred order. If the primary clock master should fail, this list is consulted by the system and a new primary clock master is assigned.

- **Operating System and System Software Version Information:** you can determine which operating system and system software versions are installed on the host computer.
- **Adding Third Party Devices**: you can add one or more third party devices to the Intel® Dialogic System. Third party devices can also be configured as the system's primary clock master or secondary clock master.

NOTE: You can add third party devices to the Intel® Dialogic system, but third party devices must be configured according to the vendor's documentation.

- **Reserve TDM Bus Resources for Third Party Devices**: you can reserve TDM bus time slots for exclusive use by third party devices. This allows third party devices to share the TDM bus with Intel® Dialogic boards.

NOTE: You can add and modify configuration data only for those Intel® Dialogic hardware products supported by the software release of which the DCM API is a component. For more information about this limitation, see Section 2.1, "The DCM API Architecture", on page 5.

Full instructions on using the DCM API are contained in Chapter 2, "Using the DCM API".

1.1.2. The Intel® Dialogic OEM Installation Tool (DOIT)

The Intel® Dialogic OEM Installation Tool (DOIT) enables you to use command-line parameters to create installation routines that are:

- **Customized:** the installation contains only the components you specify.
- **Unattended:** the installation executes without any user intervention.
- **Silent:** the installation executes without producing any screen output.

1.2. Requirements

Both the DCM API and the Intel® Dialogic OEM Installation Tool require the base components included with a Intel® Dialogic system software release that supports these two features. They do not operate as stand-alone products.

NOTE: The configuration data you can manipulate with the DCM API is limited to those boards and parameters provided with the software release of which the DCM API is a component.

1.3. Restrictions and Limitations

1.3.1. The DCM API

The following restrictions and limitations apply to the DCM API:

- **Restrictions and Limitations for All Devices:**
  - You can add and modify configuration data only for those Intel® Dialogic hardware products supported by the software release of which the DCM API is a component. Refer to the *Release Guide* that accompanies each Intel® Dialogic System Software release for a list of supported products.
  - The code samples in Chapter 4, "Function Reference" use C syntax.
  - The DCM API function code samples are not compilable.
  - The **NCM_GetVariables( )** function can be used to retrieve a list of all global configuration parameters from the DCM Catalog by setting both the NCMFamily and NCMDevice structures to NULL.
  - 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. Intel® Dialogic strongly recommends that you embed the device model name within the unique device name when you instantiate a device with the **NCM_AddDevice( )** function.
  - All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )** function can be used to start the system service. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable Intel® Dialogic devices.
  - If you use the **NCM_SetDlgSrvStartupMode( )** function to set the system service startup mode to **Automatic**, the system does not auto-detect devices before starting the system service. Therefore, the following restrictions apply when the system is rebooted in **Automatic** mode:
    - You cannot add or remove SpringWare devices.

- You cannot add or remove DM3 PCI devices. However, you can replace a DM3 PCI device with an identical device. For example, you can physically remove a DM/V960-4T1-PCI from the system, but you must install another DM/V960-4T1-PCI in the same PCI slot as the original board.
- You can add or remove DM3 CompactPCI (cPCI) devices while the system is powered on. However, you must first use the **NCM_StopDlgSrv( )** function to stop the Dialogic System Service and then use the **NCM_DetectBoardsEx( )** function before rebooting the system. Alternatively, you can use the **NCM_StopBoard( )** and **NCM_StartBoard( )** functions to perform a Hot Swap of the DM3 cPCI device without stopping the system service or telephony application. Refer to Section 2.9, "Single Board Stop/Start (Hot Swap)", on page 23 for more information about Hot Swap.

1.3.2. The Dialogic OEM Installation Tool

The following restrictions and limitations apply to the Intel® Dialogic OEM Installation Tool:

- The DOIT command line input is limited to 64 characters.

2. Using the DCM API

2.1. The DCM API Architecture

The DCM API is one layer of a multi-layer configuration management architecture. This architecture provides a vehicle for managing configuration data for Intel® Dialogic products. The components of this architecture are:

- **The DCM Catalog:** Default configuration information originates in the DCM Catalog. This default information includes the devices that can be installed, the configuration parameters that can be applied to them, and the default configuration parameter values. The content of the DCM Catalog is determined by the software release of which the DCM API is a component. The DCM API therefore does not enable you to modify the DCM Catalog.

  Throughout this manual, the term "installable" indicates that the configuration data element to which it applies is contained in the DCM Catalog. For example, an installable device is a device that is defined in the DCM Catalog.
- **The DCM System Configuration:** The system configuration consists of the configuration data currently in use in your system.

  In this manual, the terms "instantiate" and "instantiation" refer to the process of creating system configuration data.

  The Intel® Dialogic devices and configuration parameter values you can instantiate are determined by the DCM Catalog. The DCM API enables you to instantiate and delete devices and to change the configuration parameter values in your system configuration. The system configuration is read by the system software through the DCM API when it initializes Intel® Dialogic devices.

CAUTION

In the current version of the DCM API, the system configuration is stored in the Windows* registry. Do not attempt to modify the DCM registry data through any means other than the DCM API. By adhering to this guideline, you ensure forward-compatibility between your application and future versions of the DCM API.

- **The DCM API:** The DCM API consists of a library of functions for instantiating Intel® Dialogic devices in the system configuration and for modifying configuration parameter values. It also enables you to start and stop the Intel® Dialogic System Service, stop and start individual DM3 PCI and CompactPCI (cPCI) boards, and to auto-detect devices. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable devices.
- **Client Application:** A client application makes the functionality of the DCM API available to end-users. A client application may be a GUI-based configuration tool, a customized automated silent configuration process, or some other type of application.

  An example of a Client Application is the DCM GUI client, which is included in every Intel® Dialogic System Software release that includes the DCM API. Access the DCM GUI client by clicking the **Configuration Manager-DCM** icon from the **Intel Dialogic System Software** folder in the Windows Start menu.

For an overview of the DCM API architecture, see Figure 1, "DCM API Architecture", on page 8.

2.2. The Data Model

2.2.1. Configuration Parameter Values and Device Instantiation

The basic unit of configuration data is the configuration parameter value. There are configuration parameter values for each Intel® Dialogic device in your system configuration. For example, hardware configurable boards, such as the D/41D, require values for the **D41DAddress** and **D41Dinterrupt** parameters. (These

particular parameters store the valid memory and interrupt information for each Hardware Configurable device in the system.) When the Intel® Dialogic System Service is initiated, it reads the configuration parameter values through the DCM API from the system configuration and uses them to initialize the devices.

For information about the function of each configuration parameter, consult the online help for the DCM GUI client. Access the DCM GUI client by clicking the **Configuration Manager-DCM** icon from the **Intel Dialogic System Software** folder. Access the online help by pressing **F1** at any window.

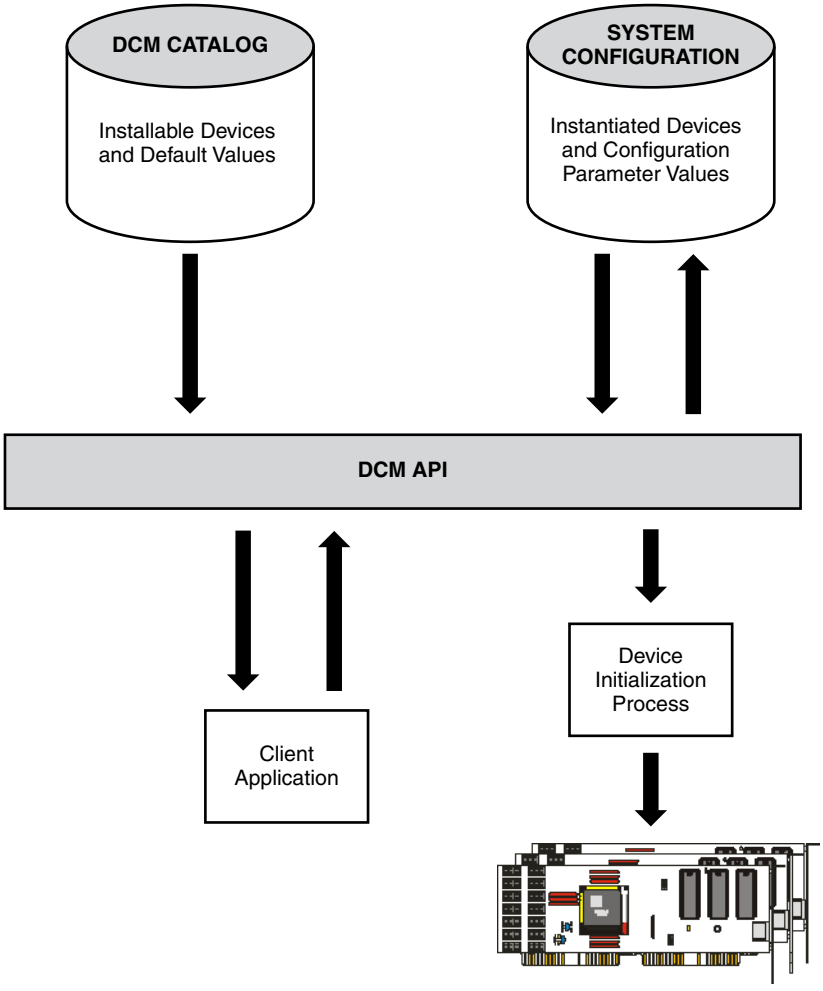

Figure 1. DCM API Architecture

A device is a Intel® Dialogic board, such as a D/240SC-2T1. The DCM API enables you to instantiate devices in your system configuration and to query the

DCM Catalog to determine which configuration parameter values can be set for a given device.

2.2.2. Configuration Parameter Property Groups

The DCM Catalog organizes configuration parameters into groupings called properties. For example, the parameters related to configuring interrupts, memory addresses, and bus slot assignments are grouped together under the **System** property. The **System** property may include the following parameters:

- **AntInterrupt**
- **BLTAddress**
- **BLTId**
- **BLTInterrupt**
- **D41DAddress**
- **D41DInterrupt**
- **ISABusWidth**
- **MasterBoard**
- **PciBusNumber**
- **PciID**
- **PciSlotNumber**

The parameters in any given property change depending on the Intel® Dialogic System Software release with which the DCM API is provided.

The properties to which configuration parameters belong are determined by the DCM Catalog. This relationship cannot be modified through the DCM API.

2.2.3. Configuration Parameter Scope

The DCM Catalog defines the scope of configuration parameters so that their value can apply either to one specific device or to a group of devices. There are three categories of parameter scope:

- **Device-Specific:** a device-specific configuration parameter value applies to only one device (board) in the system. When you edit a device specific configuration parameter, the parameter value applies only to that specific

device. For example, the **D41DAddress** parameter is defined uniquely for each Hardware Configurable board in the system.

- **Family-Level:** a family-level configuration parameter applies to a family of devices (boards) in the system, such as the DIALOG/HD family. At this level, each family of devices has configuration information that is pertinent to the entire family of devices.
- **Global:** global configuration parameter values apply to all "applicable" devices. An applicable device is a device for which the parameter has functional relevance. For example, the **D41DInterrupt** parameter applies only to boards that employ jumpers and DIP switches to configure their memory and interrupt settings. The value of the **D41DInterrupt** parameter therefore has no applicability to boards that employ Board Locator Technology (BLT). Similarly, the **BLTInterrupt** parameter applies only to BLT boards; such boards are unaffected by the value of the **D41DInterrupt** parameter.

Overridable Configuration Parameters

Overridable configuration parameters can either be treated as global configuration parameters, affecting all applicable devices, or restricted to the device level. For example, by default the **ParameterFile2** parameter affects all applicable devices. But it can also be modified as a device-level parameter. Once the configuration parameter has been modified at the device level, subsequent modifications to the global parameter have no effect at the device level.

The configuration parameter scope is determined by the DCM Catalog. It cannot be modified through the DCM API.

For information about controlling the scope of parameter read-write functions, see Section 2.3.3, "Reading and Writing Configuration Parameter Values", on page 13.

2.3. Working with the Data Model

In order to use the DCM API effectively, you will need to understand how to use the DCM API to navigate the data model. The DCM Catalog defines the following data elements:

- the devices available for installation
- the families to which the installable devices belong

- the configuration parameters that apply to each device
- the properties to which configuration parameters belong
- the scope (device, family, or global) of each parameter

The DCM Catalog provides data elements for only those devices that are supported by the Intel® Dialogic System Software release of which the DCM API is a component. The DCM API enables you to instantiate devices and to set the values of their parameters in your system configuration. It prevents you from adding to or modifying the data elements that constitute the DCM Catalog.

In order to instantiate devices and modify configuration parameter values, you will need to carry out the following preliminary steps:

1. Populate the required structures.
2. Determine configuration parameter scope.
3. Read and write configuration parameter values.

These steps are described in the subsections that follow.

2.3.1. Populating Required Structures

There are a number of structures that are necessary in order to perform basic DCM API operations. For example, the **NCM_SetValue( )** function, which enables you to change the value of a configuration parameter, requires structures to identify the family, device, and property to which the configuration parameter to be modified belongs.

The following steps illustrate how to populate these structures:

1. Use **NCM_GetAllFamilies( )** to return a list of NCMFamily structures; the structures in this list indicate the families for all installable devices defined by the current version of the DCM Catalog.
2. Use any one of the NCMFamily structures as input to **NCM_GetAllDevices( )** to return a list of NCMDevice structures for one of the families designated by the list of NCMFamily structures. Lists of installable devices can be retrieved only for one family at a time.
3. Use the NCMFamily and NCMDevice structures for a specific device as input to **NCM_GetProperties( )**; this function returns a list of NCMProperty

structures for the device identified by the NCMFamily and NCMDevice structures. Lists of properties can be retrieved only for one device at time.

4. Use the NCMProperty, NCMFamily, and NCMDevice structures as input to **NCM_GetVariables( )**; this function returns a list of NCMVariable structures for the installable device indicated by the specified input structures. The list of NCMVariable structures contains all the installable configuration parameters within the property for the specified device.

   NOTE: The contents of the NCMVariable structure list varies depending on the specific device being queried. For example, the **System** property for a D/41D device contains the **D41DAddress**, **D41DInterrupt**, and **ISABusWidth** parameters, whereas the same property for a D/240SC device contains **BLTAddress**, **BLTInterrupt**, **BLTId**, and **ISABusWidth**.

For more information about these structures, see Chapter 3, "DCM API Structures".

2.3.2. Determining Configuration Parameter Scope

A configuration parameter's scope determines whether its value applies to all devices, a family of devices, or one specific device (for more information, see Section 2.2.3, "Configuration Parameter Scope", on page 9).

In order to modify a configuration parameter, you must know its scope. To determine the scope of a configuration parameter, consult the online help accompanying the DCM GUI client by following these steps:

1. Click the **Configuration Manager-DCM** icon from the **Intel Dialogic System Software** folder in the Windows Start menu.
2. When DCM GUI client loads, press F1 to invoke the online help.
3. Click **Help Topics** to invoke the table of contents for the help file.
4. From the DCM online help, click the **Parameter Reference** book.
5. From the **Parameter Reference** book, click **Configuration Parameters**.
6. From the list of parameters, click the name of the parameter you wish to modify.
7. Consult the **Rules** section of the parameter description. The two elements of the **Rules** section relevant to a parameter's scope are:

- **Scope:** This element indicates whether the parameter's scope is global or device.
- **Override:** For global parameters, the letter **Y** indicates that the parameter can be overridden on a global basis; the letter **N** indicates that the parameter can not be overridden.

2.3.3. Reading and Writing Configuration Parameter Values

Functions such as **NCM_GetValue( )** and **NCM_SetValue( )** that enable you to read or write a configuration parameter value in the system configuration require that you specify the parameter's scope. Such functions accept the NCMFamily and NCMDevice structures as input, and it is through these structures that you specify the parameter's scope. Set these structures as follows depending on the type of parameter you wish to modify:

- **device-specific configuration parameters:** NCMDevice should be set to a valid address.
- **family level configuration parameters:** NCMFamily should be set to a valid address.
- **global configuration parameters:** NCMFamily and NCMDevice should be set to NULL.

  The **NCM_GetVariables( )** function can be used to retrieve a list of all global configuration parameters from the DCM Catalog by setting both the NCMFamily and NCMDevice structures to NULL.
- **overridable configuration parameters:** to treat the parameter globally, set the value of the NCMFamily and NCMDevice structures as you would a global parameter, both to NULL; to treat the parameter at the device level, set the value of the NCMFamily and NCMDevice structures as you would a device-level parameter, both set to a valid address.

  NOTE: The DCM Catalog maintains two copies of overridable configuration parameters, one at the global level, and another for each device. By default, the device-level copy is updated whenever the global-level copy is modified. But once the device-level copy is modified, its value is de-linked from the global-level copy. Changes to the global-level no longer affect that device.

This method for accessing different levels of data is summarized in Table 1, "Configuration Parameter Level Control", on page 14.

Table 1. Configuration Parameter Level Control

Parameter Scope	Value of NCMFamily	Value of NCMDevice
Device	Pointer to family	Pointer to device
Family	Pointer to a family	NULL
Global	NULL	NULL
Global overridable, treated as global	NULL	NULL
Global overridable, treated as device-specific	Pointer to family	Pointer to device

NOTE: There are several parameters for Antares products that are global in scope and are retrieved through the Antares Global Settings family. The parameters are: **Bulk_Data_Buffer**, **DspAskToken**, **DspWaitCall**, **Max_Bulk_Data_Buffer**, **Max_Bulk_Data**, **Max_Dpi**, **Max_Opened_Rcus**, **Max_Rcus**, **Message_Length**.

2.3.4. Device Model Names and Unique Device Names

The NCMDevice structure can be set to a "device model name" or a "unique device name". A device model name is the generic Intel® Dialogic name for a device, such as "D/41D" and "D/240SC". The device model name is necessary to retrieve a list of installable devices with the **NCM_GetAllDevices( )** function.

A unique device name is a unique identifier that you create when you instantiate a device in the system configuration using the **NCM_AddDevice( )** function. The unique device name is necessary to distinguish multiple instances of the same device model in the system.

DCM API functions that take a pointer to an NCMDevice structure as input differ with respect to the type of device name they require. These functions fall into the following categories:

- **Functions that write to or read from the system configuration and require a unique device name.** This category includes:
  - **NCM_AddDevice( )**
  - **NCM_DeleteEntry( )**
  - **NCM_EnableBoard( )**
  - **NCM_IsBoardEnabled( )**
  - **NCM_SetValue( )**
  - **NCM_SetValueEx( )**
  - **NCM_StartBoard( )**
  - **NCM_StopBoard( )**

  You create the unique device name with the **NCM_AddDevice( )** function. You can then retrieve the unique device name with the **NCM_GetInstalledDevices( )** function. The unique device name you retrieve can then be used with the other functions in this category.

  NOTE: 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. Intel® Dialogic strongly recommends that you embed the device model name within the unique device name when you instantiate a device with the **NCM_AddDevice( )** function.
- **Functions that read from the DCM Catalog and require either a device model name or a unique device name.** This category includes:
  - **NCM_GetProperties( )**
  - **NCM_GetValueRange( )**
  - **NCM_GetValueRangeEx( )**
  - **NCM_GetVariables( )**
  - **NCM_IsEditable( )**

The data provided by the **NCM_GetValue( )** and **NCM_GetValueEx( )** functions is affected by whether NCMDevice is a device model name or a unique device name. If NCMDevice is a device model name, these functions read the default

configuration parameter value from the DCM Catalog. If NCMDevice is a unique device name, they read the instantiated configuration parameter value from the system configuration.

In the DCM API functions in Chapter 4, “Function Reference”, the entries for functions that require an NCMDevice structure as input indicate whether the structure must contain a device model name or a unique device name.

2.3.5. Data Model Function Summary

In working with configuration data, you will encounter two families of functions, one for reading the contents of the DCM Catalog, another for reading and writing the contents of the system configuration. These two groups are as follows:

- **Functions for reading from the DCM Catalog:**
  - **NCM_GetAllDevices( )**
  - **NCM_GetAllFamilies( )**
  - **NCM_GetVariables( )**
  - **NCM_GetValueRange( )**
  - **NCM_GetValueRangeEx( )**
  - **NCM_GetProperties( )**
- **Functions for reading from and writing to the system configuration:**
  - **NCM_DeleteEntry( )**
  - **NCM_GetInstalledDevices( )**
  - **NCM_GetInstalledFamilies( )**
  - **NCM_AddDevice( )**
  - **NCM_SetValue( )**
  - **NCM_SetValueEx( )**
  - **NCM_EnableBoard( )**
  - **NCM_IsBoardEnabled( )**
  - **NCM_SetTDMBusValue( )**
  - **NCM_SetClockMasterFallbackList( )**

NOTE: The data source for the **NCM_GetValue( )**, **NCM_GetValueEx( )**, **NCM_GetTDMBusValue( )**, and **NCM_GetClockMasterFallbackList( )** functions changes depending on the value of NCMDevice. If NCMDevice is a device model name, these functions read the default configuration parameter value from the DCM Catalog. If NCMDevice is a unique device name, they read the instantiated configuration parameter value from the system configuration.

2.4. Dynamic Memory Allocation

Many DCM API functions return dynamic data in the form of linked lists, the last item in the list pointing to NULL. In this case, memory space must be allocated to accommodate the linked list. Functions of this type accept a pointer to an address at which to allocate the memory needed to return data to the client. The pointer is declared in the client application.

In order to avoid memory leaks in the application, this memory must be deallocated when it is no longer being used. The **NCM_Dealloc( )** and **NCM_DeallocValue( )** functions can be called by the client application to deallocate the memory dynamically allocated by another API function. Functions that require the use of **NCM_Dealloc( )** and **NCM_DeallocValue( )** are identified as such in Chapter 4, “Function Reference”.

2.5. Error Code Handling

Every API function returns a code that indicates the success or failure of the function. Any client application calling an API function should check the return codes. The **NCM_GetErrorMsg( )** function translates the function return code into a text message.

NOTE: Refer to the System Log in the Windows Event Viewer for a detailed explanation of DCM API error messages.

2.6. Board Detection and System Parameter Values

2.6.1. Auto Detection

The **NCM_DetectBoards( )** and **NCM_DetectBoardsEx( )** functions detect Intel® Dialogic devices that are installed in your system chassis. These functions detect Board Locator Technology (BLT), PCI, and cPCI type devices. They do not detect Hardware Configurable or Antares ISA devices.

NOTES: 1. Intel® Dialogic recommends using the **NCM_DetectBoards( )** function to detect non-DM3 boards only.

2. All auto-detectable Intel® Dialogic devices in the system must be detected using either **NCM_DetectBoards( )** or **NCM_DetectBoardsEx( )** before the **NCM_StartDlgSrv( )** function can be used to start the Intel® Dialogic System Service.

When the DCM API detects a device installed in the system chassis, it instantiates default configuration parameter values for the device. As explained in the following subsection, it also provides appropriate values for the device's system property parameters.

2.6.2. System Property Parameters

The parameters that store the memory, IRQ, and port values all belong to the **System** property. When you instantiate a device, the DCM API queries the Windows kernel for available resources and sets the values of these parameters appropriately. Please note the following details with regard to the different types of Intel® Dialogic boards:

- **BLT Boards:** the System property for BLT boards includes the **BLTAddress**, **BLTId**, and **BLTInterrupt** parameters. The **BLTId** parameter is set by the DCM API to the value of the rotary-switch setting of the board (if there are more than one BLT boards in the system, the rotary-switch setting must be unique for each one). The values for **BLTAddress** and **BLTInterrupt** are set according to the available memory and interrupt resources registered with the Windows kernel. All BLT boards share the same memory and interrupt, so the **BLTAddress** and **BLTInterrupt** configuration parameter values are the same for all BLT boards. The DCM API's ability to set these parameters, in

conjunction with auto-detection, make it possible to install a BLT board in the system chassis and then detect and configure the board's system configuration parameter values with no further user intervention.

- **Hardware Configurable Boards:** the System property for Hardware Configurable boards includes the **D41DAddress** and **D41DInterrupt** parameters. The values for **D41DAddress** and **D41DInterrupt** are set according to the available memory and interrupt resources registered with the Windows kernel. All Hardware Configurable boards share the same interrupt, so the **D41DInterrupt** configuration parameter value is the same for all Hardware Configurable boards. Because Hardware Configurable boards cannot be auto-detected, they should not be installed in the chassis prior to determining the value of **D41DAddress** and **D41DInterrupt**. Once these configuration parameter values are set, the user can configure the board's jumpers and switches and, after powering down the system, install the board in the system chassis.
- **Antares ISA:** the System property for Antares ISA boards includes the **AntInterrupt** parameter. The **Port** parameter, which is also necessary for basic board functioning, is included in the Board property. The values for **AntInterrupt** and **Port** are set according to the available memory and port resources registered with the Windows kernel. All Antares ISA boards share the same interrupt, so the **AntInterrupt** configuration parameter value is the same for all Antares ISA boards. Because Antares ISA boards cannot be auto-detected, they should not be installed in the chassis prior to determining the values of **AntInterrupt** and **Port** configuration parameter values. Once these configuration parameters are set, the user can configure the board's jumpers and switches and, after powering down the system, install the board in the system chassis.
- **PCI:** the System property for PCI boards includes the **PciBusNumber**, **PciID**, and **PciSlotNumber** parameters. For Antares PCI boards, the **PciBusNumber** and **PciSlotNumber** are included, along with **PciInterrupt**, in the Board property parameters. The values for these configuration parameters are set automatically by the operating system. The DCM API's ability to record these parameter values, in conjunction with auto detection, make it possible to install a PCI board in the system chassis and then detect and configure the board with no further user intervention.

NOTE: The DCM API relies on the Windows kernel for setting system configuration parameters. In so doing, it follows the standard procedure for requesting resources from Windows and for notifying the kernel that those resources are taken by Intel® Dialogic devices. If another device in your system does not comply with this procedure, a conflict may occur when the Intel® Dialogic System Service attempts to initialize the Intel® Dialogic devices.

2.7. Working with Country Specific Parameters

The country-specific configuration parameters behave differently from other configuration parameters. Rather than being stored in individual configuration parameters, the country-specific configuration parameters are concatenated together in a comma-separated string. This string is stored in the **Features** configuration parameter. The countries for which country-specific configuration parameters can be set are stored in the **Country** configuration parameter.

Working with country-specific configuration parameters differs from other configuration parameters in the following ways:

- **You can use special functions for reading country-specific configuration parameters.** A distinct set of functions enables you to read the country-specific configuration parameters contained in the **Features** parameter. These functions are:
  - **NCM_GetCspFeaturesValue( )**, to get the value of a particular country-specific configuration parameter from the system configuration (such as **Frequency Resolution** or **Analog Signaling**) from within **Features**.
  - **NCM_GetCspFeaturesValueRange( )**, to get the valid value range for a specific country-specific configuration parameter code from the DCM Catalog.
  - **NCM_GetCspFeaturesVariables( )**, to get all country-specific configuration parameters contained in the DCM Catalog.
- **You must set the Features configuration parameter using the normal DCM API write functions.** Although you can read the individual country-specific configuration parameters using the parameters outlined above, the **Feature** configuration parameter must be set using the functions described in Section 2.3.3, “Reading and Writing Configuration Parameter Values”, on page 13. To change the value of any particular country-specific configuration

parameter, your application must modify the parameter within the pointer to a comma-separated list of country-specific configuration parameters and write this string to the **Feature** configuration parameter using the **NCM_SetValue( )** function.

- **The country-specific configuration parameters retrieved for the Country property vary with the value of the Country parameter.** For all other properties, the configuration parameters are retrieved with the **NCM_GetVariables( )** function; the configuration parameters retrieved in this way are those that are applicable to the device name input to this function. For example, the **System** property for a D/41D device contains the **D41DAddress**, **D41DInterrupt**, and **ISABusWidth** parameters, whereas the same property for a D/240SC device contains **BLTAddress**, **BLTInterrupt**, **BLTId**, and **ISABusWidth**. In the case of country-specific configuration parameters, however, it is the value of the **Country** configuration parameter that determines which country-specific configuration parameters belong to the property. You can retrieve a list of valid country-specific configuration parameters for a given country with the **NCM_GetCspFeaturesVariables( )** function.
- **The valid values for the Country configuration parameter must be determined using special functions.** A distinct set of functions enables you to work with country codes and country names. You can use these functions to help set the **Country** configuration parameter, which contains the ISO country code for the country you want to set. The functions are:
  - **NCM_GetCspCountries( )**, to get a list of supported countries from the DCM Catalog.
  - **NCM_GetCspCountryCode( )**, to get the country code for a country from the DCM Catalog.
  - **NCM_GetCspCountryName( )**, to get the country name for a given country code from the DCM Catalog.

2.8. Starting and Stopping the Intel® Dialogic System Service

The Intel® Dialogic System Service must be started before you can use Intel® Dialogic boards. When it is started, the Intel® Dialogic System Service initializes Intel® Dialogic devices using the configuration parameter values stored in the

current system configuration. It also initiates the device drivers through which your telephony applications employ the functionality of Intel® Dialogic boards.

The DCM API also enables you to set the Intel® Dialogic System Service startup mode. When set to **Automatic**, the Intel® Dialogic System Service starts every time your system boots. When it is set to **Manual**, the end-user must start the Intel® Dialogic System Service through your client application. Since starting a service requires administrative access privileges, it can be advantageous to set the Intel® Dialogic System Service startup mode to **Automatic**. However, when the Intel® Dialogic System Service startup mode is set to **Automatic**, the system does not auto-detect devices before starting the Intel® Dialogic System Service. Therefore the following restrictions apply when the system is rebooted in **Automatic** mode:

- You cannot add or remove SpringWare devices.
- You cannot add or remove DM3 PCI devices. However, you can replace a DM3 PCI device with an identical device. For example, you can physically remove a DM/V960-4T1-PCI from the system, but you must install another DM/V960-4T1-PCI in the same PCI slot as the original board.
- You can add or remove DM3 CompactPCI (cPCI) devices while the system is powered on. However, you must first use the **NCM_StopDlgSrv( )** function to stop the Intel® Dialogic System Service and then use the **NCM_DetectBoardsEx( )** function before rebooting the system. Alternatively, you can use the **NCM_StopBoard( )** and **NCM_StartBoard( )** functions to perform a Hot Swap of the DM3 cPCI device without stopping the Intel® Dialogic System Service or telephony application. Refer to Section 2.9, “Single Board Stop/Start (Hot Swap)”, on page 23 for more information about Hot Swap.

The following functions enable you to work with the Intel® Dialogic System Service:

- **To initiate starting and stopping the Intel® Dialogic System Service:** use the **NCM_StartDlgSrv( )** and **NCM_StopDlgSrv( )** functions.

NOTE: All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )**

function can be used to start the Intel® Dialogic System Service. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable Intel® Dialogic devices.

- **To determine whether the Intel® Dialogic System Service is currently running:** use the **NCM_GetDlgSrvStateEx( )** function.
- **To determine whether the Intel® Dialogic System Service starts whenever the system starts:** use the **NCM_GetDlgSrvStartupMode( )** function.
- **To allow or prevent devices from being initialized when the Intel® Dialogic System Service starts:** use the **NCM_EnableBoard( )** and **NCM_IsBoardEnabled( )** functions.

For information about using these functions, see Chapter 4, "Function Reference".

2.9. Single Board Stop/Start (Hot Swap)

DM3 H.110 capable boards can be individually stopped and started, allowing you to replace CompactPCI (cPCI) boards without shutting down the system or telephony application. This process is known as *Hot Swap*. The DM3 cPCI board that you wish to replace is first stopped and then physically removed from the system. Next, the replacement board is installed in the same cPCI slot as the original board and then started. The following restrictions apply to the *Hot Swap* process:

- You must use the **NCM_StopBoard( )** function to manually stop the cPCI board before it is physically removed from the system.
- The boards that are being swapped must be identical board models. For example, a DM/V960-4T1-cPCI can only be swapped with another DM/V960-4T1-cPCI.
- You must use the **NCM_StartBoard( )** function to manually start the replacement board.

NOTE: DM3 H.100 (PCI) boards can also be individually stopped and started, but not for the purpose of performing a *Hot Swap*. For example, you can stop and start a DM3 PCI board to re-download the firmware to the board.

For information about using these functions, see Chapter 4, "Function Reference".

2.10. Clock Master Fallback List

The Primary Clock Master provides bus timing (bit clock and frame synchronization) to all boards in the system. The Primary Clock Master derives its timing from either a network interface (optimum) or from its own internal oscillator. When clocking is derived from a network interface, the Primary Master Clock uses the CT Bus NETREF signal as the clock reference (see Figure 2, “CT Bus Clocking”, on page 24).

Under normal operation, the Primary Clock Master clock output is re-driven by the Secondary Clock Master, providing redundant backup clocking to all boards in the system should the Primary Clock Master fail.

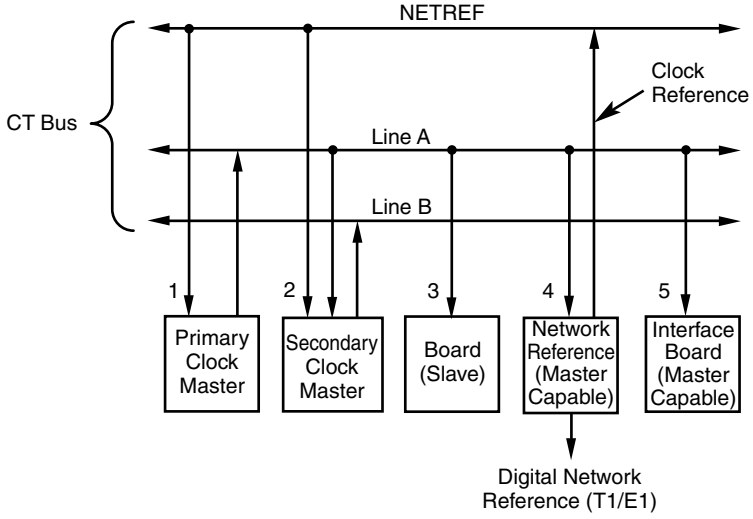

Figure 2. CT Bus Clocking

In addition, multiple clock master fallback devices can be defined using the clock master fallback list. The clock master fallback list defines a list of master capable devices in a preferred order. If the current Primary Clock Master should fail, this list is consulted by the system and a new Primary Clock Master is assigned.

The CT Bus includes a primary clock signal line (Line A) and a secondary clock signal line (Line B). Either Line A or Line B can be assigned as the Primary Line (driven by the Primary Clock Master). The remaining line is assigned as the Secondary Line (driven by the Secondary Clock Master). The Primary Line carries clock synchronization to all boards in the system.

Primary and Secondary Clock Masters are always selected automatically by the system. When a clock failure occurs and the clock master fallback list is defined, the list is consulted by the system (without the intervention of the administrative application) to determine the new clock master device. The list is consulted from the first entry down each time to assign the best clock master device available.

If the clock master fallback list is not defined, the system will select a new clock master device on its own, should a failure occur.

For example, if the clock fallback list is defined as follows:

1. Board 1
2. Board 2
3. Board 5
4. Board 4

and Board 2 is the current Primary Clock Master, if a failure should occur on the Primary lines, the system will first check if Board 1 is capable of being the Primary Clock Master before proceeding further down the list.

If a previously failed clock master recovers, the system will not automatically assign that clock master as the Primary Clock Master.

The clock fallback list is defined using the **NCM_SetClockMasterFallbackList( )** function. Clock fallback clock master sources are defined with the **NCM_SetTDMBusValue( )** function.

2.11. Compiling an Application Program That Uses the DCM API

The DCM API requires the following files:

- Header files:
  - *NCMApi.h*
  - *NCMTypes.h*
- Library:
  - *NCMApi.lib*

    The default location of this file is *\Program Files\Dialogic\Lib*.
- DLLs:
  - *NCMApi.dll*
  - *NCMData.dll*
  - *NCMCconf.dll*

These files are installed as part of the Intel® Dialogic System Software release with which the DCM API is a component.

3. DCM API Structures

3.1. Data Structures

The NCMString structure defines most structures in the DCM API.

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

The following structures are aliases for NCMString:

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

The NCMPropertyAttributes is defined as follows:

```
typedef enum
{
     NCM_PROPERTY_UNDEFINED = -1,
     NCM_PROPERTY_VISIBLE,
     NCM_PROPERTY_HIDDEN
} NCMPropertyAttributes;
```

The NCMVariableAttributes structure is defined as follows:

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

The NCMSysVersion structure is defined as follows:

```
typedef struct _NCMSysVersion
{
     char szOSName[MAX_PATH];          // Name of operating system (Windows …)
     char szOSVersion[MAX_PATH];       //Version of OS
     char szOSBuild[MAX_PATH];         //OS build number
     char szOSType[MAX_PATH];          //OS Type
     char szOSSvcPack[MAX_PATH];       // OS Service Pack number
     char szDSSVersion[MAX_PATH];      //Dialogic software version
     char szDSSRelease[MAX_PATH];      // Dialogic software release
     char szDSSBuild[MAX_PATH];        //Dialogic software build number
     char szDSSSvcPack[MAX_PATH];      //Dialogic software Service Pack number
} NCMSysVersion;
```

The NCM_TS_BLOCK_STRUCT data structure is defined as follows:

```
typedef _TS_BLOCK_STRUCT
{
   int   version;
   int   start_time_slot; //starting time slot number of the reserved block
   int   number_of_time_slots; //total number of time slots in the reserved block
} NCM_TS_BLOCK_STRUCT
```

The NCM_TDM_BUS_CAPS data structure is defined as follows:

```
typedef struct
{
   int    structVersion;
   bool   bH100MasterCapable;
   bool   bH100SlaveCapable;
   bool   bH110MasterCapable;
   bool   bH110SlaveCapable;
   bool   bScbusMasterCapable;
   bool   bScbusSlaveCapable;
   bool   bMvipMasterCapable;
   bool   bMvipSlaveCapable;
   bool   bScbus2MhzCapable;
   bool   bScbus4MhzCapable;
   bool   bScbus8MhzCapable;
} NCM_TDM_BUSCAPS;
```

For instructions on using these structures, see Section 2.3.1, “Populating Required Structures”, on page 11.

3.2. Structures for Extended Functions

The DCM API also includes structures for the extended functions, **NCM_DetectBoardsEx( )**, **NCM_GetDlgSrvStateEx( )**, **NCM_GetValueEx( )**, **NCM_GetValueRangeEx( )**, and **NCM_SetValueEx( )**.

- NCMDataType

```
typedef enum
{
     UNDEFINED=0,
     NUMERIC,
     ALPHANUMERIC,
     NCMFILE                  // to be used for filenames
} NCMDataType;
```

- NCMValueEx

```
typedef struct _NCMValueEx
{
     int                structSize;  // sizeof(NCMValueEx)
     NCMDataType        dataType;    // enumerated type signifies the type of
                                     // data
     void               *dataValue;  // a buffer which holds the data
     int                dataSize;    // size of the buffer
     struct _NCMValueEx  *next;
} NCMValueEx;
```

The following structures are used in the **NCM_DetectBoardsEx( )** function:

- NCM_DETECTION_INFO

```
typedef struct _NCM_DETECTION_INFO     //User -> API
{
     int                        structSize;          // sizeof(
                                                     // NCM_DETECTION_INFO )
     NCM_CALLBACK_FCN           *callbackFcn;        // address of call back
                                                     // function
     NCM_PCDFILE_SELECTION_FCN  *pcdFileCallbackFcn; //address of PCD file
                                                     //callback function
} NCM_DETECTION_INFO;
```

The NCM_DETECTION_INFO structure contains the structure size and the address of the following two callback functions:

- NCM_CALLBACK_FCN

```
typedef int (NCM_CALLBACK_FCN) (OUT UINT percentage completed, OUT const
                                                          char *message);
```

- NCM_PCDFILE_SELECTION_FCN

```
typedef int (NCM_PCDFILE_SELECTION_FCN) (IN NCMFileInfo *fileList,
                                         IN size_t numFiles,
                                         IN NCMDevInfo devInfo,
                                         OUT int*index);
```

The NCM_PCDFILE_SELECTION_FCN function contains the following structures:

- NCM_FileInfo

```
typedef struct _NCMFileInfo
     {
          DWORD         version;
          char          fileName[MAX_PATH];
          char          fileDesc[NCM_MAX_FILEDESC];
          char          modelName[MAX_PATH];
     } NCMFileInfo;
```

- NCM_DevInfo

```
typedef struct_NCMDevInfo
{
                   DWORD          version;
                   char          InstanceLabel[256];
                   unsigned long     InstanceNumber;
                   unsigned long     LogicalID;
                   unsigned long     IRQLevel;
                   unsigned long     intVector;
                   LARGE_INTEGER     PLXAddr;
                   unsigned long     PLXLength;
                   LARGE_INTEGER     SRAM Addr;
                   unsigned long     SRAMLength;
                   unsigned long     busNumber;
                   unsigned long     slotNumber;
                   unsigned long     dlgcOUI;
                   UCHAR          primBoardId[4];
                   UCHAR          secBoardId[4];
                   unsigned long     modelNumber;
                   unsigned long     sramSize;
                   unsigned long     locatorId;
                   UCHAR          serialNumber[NCM_MAX_SERNUM_LEN];
                   unsigned long     curState;
                   BOOL               startPending;
                   unsigned long     shelfID;
                   unsigned long     subnet ID;
                   unsigned long     physicalState;
                   }NCMDevInfo;
```

- NCM_DETECTION_RESULT

```
typedef struct _NCM_DETECTION_RESULT     // User <- API
{
int                          structSize;              // size of
                                                     //  NCM_DETECTION_RESULT
int                          totalDetectedBoards;     // total detected boards
NCM_DETECTION_DETAILS        returnInfo;               // detection returned info.
} NCM_DETECTION_RESULT;
```

- NCM_DETECTION_DETAILS

```
typedef struct _NCM_DETECTION_DETAILS     // User <- API
{
     int      structSize;               // sizeof( NCM_DETECTION_DETAILS
     int      numDetectors;               // number of detectors
     int      numBoardsDetected[256];     // number of boards detected by the
                                          // detector
     int      returnCode[256];          // detector return code
     char     returnMsg[256][256];     // detector returned message
     char     detector[256][256];          // board detector name
} NCM_DETECTION_DETAILS;
```

- NCM_DETECTOR_INFO

```
typedef struct _NCM_DETECTOR_INFO          // API -> Detector
{
     int                    structSize;     // sizeof(NCM_DETECTOR_INFO)
     NCM_CALLBACK_FCN     *callbackFcn;     // address of the call back function
} NCM_DETECTOR_INFO;
```

- NCM_DETECTOR_RESULT

```
typedef struct _NCM_DETECTOR_RESULT     // API <- Detector
{
     int          structSize;               // sizeof(NCM_DETECTOR_RESULT)
     int          numBoardsDetected;          // number of boards detected
     char          returnMsg[256];          // detection returned message
} NCM_DETECTOR_RESULT;
```

- NCM_DETECTOR_FCN

```
     typedef int (NCM_DETECTOR_FCN) (IN NCM_DETECTOR_INFO*, OUT
                                                          NCM_DETECTOR_RESULT*);
```

- DETECT_BOARDS_FCN

```
     typedef int ( DETECT_BOARDS_FCN) (IN NCM_DETECTOR_INFO*detectionInfo,
                                       OUT NCM_DETECTOR_RESULT*detectionResults);
```

4. Function Reference

This section provides an alphabetical reference of all DCM API functions. The table of contents for this manual indicates the page number for each function description.

The following limitations and restrictions apply to the functions described in this section:

- The code samples in the Function Reference use C syntax.
- The DCM API function code samples are not compilable.
- The **NCM_GetVariables( )** function can be used to retrieve a list of all global configuration parameters from the DCM Catalog by setting both the NCMFamily and NCMDevice structures to NULL.
- All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )** function can be used to start the Intel® Dialogic System Service.
- 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. Intel® Dialogic strongly recommends that you embed the device model name within the unique device name when you instantiate a device with the **NCM_AddDevice( )** function.
- If you use the **NCM_SetDlgSrvStartupMode( )** function to set the Intel® Dialogic System Service startup mode to **Automatic**, the system does not auto-detect devices before starting the Intel® Dialogic System Service. Therefore, the following restrictions apply when the system is rebooted in **Automatic** mode:
  - You cannot add or remove SpringWare devices.
  - You cannot add or remove DM3 PCI devices. However, you can replace a DM3 PCI device with an identical device. For example, you can physically remove a DM/V960-4T1-PCI from the system, but you must install another DM/V960-4T1-PCI in the same PCI slot as the original board.

- You can add or remove DM3 CompactPCI (cPCI) devices while the system is powered on. However, you must first use the **NCM_StopDlgSrv( )** function to stop the Intel® Dialogic System Service and then use the **NCM_DetectBoardsEx( )** function before rebooting the system. Alternatively, you can use the **NCM_StopBoard( )** and **NCM_StartBoard( )** functions to perform a Hot Swap of the DM3 cPCI device without stopping the Intel® Dialogic System Service or telephony application. Refer to Section 2.9, "Single Board Stop/Start (Hot Swap)", on page 23 for more information about Hot Swap.

NOTE: Do not use any of the functions described in this section without first reading Chapter 2, "Using the DCM API".

Name: NCMRetCode NCM_AddDevice(pncmFamily, pncm DeviceModel, pncmDeviceUnique)

Inputs:

NCMFamily* pncmFamily	• pointer to a structure containing a device family name
NCMDevice* pncmDeviceModel	• pointer to a structure containing a device model 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

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. It also sets the device's System parameters based on the system's available resources (for more information, see Section 2.6.2, "System Property Parameters", on page 18).

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family
pncmDeviceModel	pointer to the structure containing the device's model name; the value of the structure must be an installable device

Parameter	Description
pncmDeviceUnique	pointer to the structure containing the device's unique name, this name can be any string sufficient to distinguish multiple instantiations of the same device model

■ 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. Intel® Dialogic strongly recommends 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, see Section 2.3.1, "Populating Required Structures", on page 11.

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, see Section 2.1, "The DCM API Architecture", on page 5.

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_NO_RESOURCES	there are no more system resources (memory, IRQ, or ports) for the device to use
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	the family name is invalid
NCME_INVALID_DEVICE	the 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

■ See Also

- **NCM_GetInstalledDevices( )**

- **NCM_GetInstalledFamilies( )**
- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**

Name: NCMRetCode NCM_AddThirdPartyDevice(pDeviceName, TDMBusCapabilities, eMasterStatus)

Inputs:	NCMDevice* pDeviceName	• third party device name that is being added
	NCM_TDM_BUSCAPS TDMBusCapabilities	• TDM bus capabilities of the third party device
	NCMMasterStatus eMasterStatus	• specifies the clock role of the third party device
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_AddThirdPartyDevice( )** function adds a third party device to the Intel® Dialogic system. The function uses the NCM_TDM_BUSCAPS data structure to define the TDM bus capabilities of the third party device. The TDM bus capabilities indicate the following:

- bus type (H.100, H.110 or SCbus)
- if the device is *capable* of being defined as the primary clock master and/or the secondary clock master
- clock rate (for SCbus systems only)

The **eMasterStatus** parameter *actually sets* the TDM bus status of the third party device.

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of the third party device being added
TDMBusCapabilities	indicates the TDM bus capabilities of the third party device as defined by the NCM_TDM_BUSCAPS data structure
eMasterStatus	sets the TDM bus status of the third party device. Possible values are as follows: • NCM_PRIMARY- device will serve as the primary clock master in the system • NCM_SECONDARY- device will serve as the secondary clock master in the system • NCM_SLAVE- device will be a slave in the system

■ Cautions

- You cannot set the **pDeviceName** parameter to NULL.
- You must ensure that each third party device that is added to the system has a unique device name. Intel® Dialogic recommends embedding a "ThirdPartyDevice" prefix to any third party device name that is added to the system. For example, if your third party board is from XYZ company, Intel® Dialogic recommends using "ThirdPartyDevice-XYZ*nnn*" as the device name, where *nnn* is a device number or an identifiable string.

■ Example

```
#include "NCMApi.h"

NCM_TDM_BUSCAPS   busCAps;
NCMRetCode        ncmRc=NCM_SUCCESS;
NCMMasterStatus   clockStatus=NCM_PRIMARY;

NCMDevice deviceName;
Char DeviceString[] = "ThirdPartyDevice-XYZ#1";
deviceName.name = (char *) DeviceString;
deviceName.next = NULL;

//adding a third party device that is only SCBus/H100 capable
```

```
BusCaps.structVersion = NCM_BUSCAPS_VER_0100;
BusCaps.bH100MasterCapable = true;
BusCaps.bH100SlaveCapable = true;
BusCaps.bH110MasterCapable = false;
BusCaps.bH110SlaveCapable = false;
BusCaps.bScbusMasterCapable = true;
BusCaps.bScbusSlaveCapable = true;
BusCaps.bMvipMasterCapable = false;
BusCaps.bMvipSlaveCapable = false;
BusCaps.bScbus2MhzCapable = false;
BusCaps.bScbus4MhzCapable = true;
BusCaps.bScbus8MhzCapable = true

//call NCM API function:
ncmRc = NCM_AddThirdPartyDevice(deviceName, busCaps, clockStatus);

if (ncmRc ! = NCM_SUCCESS)
{
  /*process error*/
}
else
{
   //process successful function call
}
...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	invalid inputs
NCME_MISSING_BUS_CAPABILITIES	invalid TDM bus capabilities
NCME_SYSTEMERROR	lack of system resources
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_FAIL_TO_CONFIGURE_BUS	failure to configure TDM bus

■ See Also

- **NCM_RemoveThirdPartyDevice( )**

Name: NCMRetCode NCM_AllocateTimeslots(pDeviceName, iNumTimeSlots, eArbitrary, ePersistent, pnNumOfBlocks, pNCMTSBlock)

Inputs:	NCMDevice *pDeviceName	• pointer to a third party device name
	int iNumTimeSlots	• number of time slots to be allocated
	NCMTSRequestType eArbitrary	• determines whether the time slots that are allocated will be determined by the system or specifically set by the user
	NCMTSReserveType ePersistent	• determines the time slot reservation type (transient, persistent or all)
	int *pnNumOfBlocks	• pointer to the actual number of time slot blocks that are allocated
	NCM_TS_BLOCK_STRUCT *pNCMTSBlock	• pointer to the block of time slots to be returned
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_AllocateTimeslots( )** function is used to allocate TDM bus time slots for use by third party devices. Intel® Dialogic boards will not use time slots that are allocated for third party devices.

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of the third party device that will be associated with the allocated time slots. A device name for this parameter is optional. If you do not want to associate a specific third party device with the allocated time slots, set this parameter to NULL. If you are associating a specific third party device with the allocated time slots, then the device name must be the same name you used to add the third party device to the system configuration with the **NCM_AddThirdPartyDevice( )** function.
iNumTimeSlots	indicates the number of time slots to be allocated. This parameter is ignored if the **eArbitrary** parameter is set to NCM_TIMESLOT_SPECIFIC.
eArbitrary	indicates whether the allocated time slots will be arbitrary or specific. Valid settings are as follows: • NCM_TIMESLOT_ARBITRARY - time slot numbers within the reserved block of allocated time slots is determined by the system software. • NCM_TIMESLOT_SPECIFIC - time slot numbers within the reserved block of allocated time slots is set by the user. The **iNumTimeSlots** parameter is ignored when this value is used.
ePersistent	determines whether or not the allocated time slots will persist when the system is rebooted. Valid settings are as follows: • NCM_TIMESLOT_PERSISTENT - allocated time slot blocks will be retained each time the system is rebooted. • NCM_TIMESLOT_TRANSIENT - allocated time slots will be released after the system is rebooted.

Parameter	Description
pnNumOfBlocks	pointer to the actual number of time slot blocks that are allocated by the function call. A value of 1 indicates that a single, continuous block of time slots will be allocated. A value of greater than 1 indicates that multiple blocks of time slots will be allocated.
pNCMTSBlock	pointer to NCM_TS_BLOCK_STRUCT data structure containing a block of time slots that is returned by the function call

The following table summarizes the valid parameter combinations for the **NCM_AllocateTimeslots( )** function:

pDeviceName	eArbitrary	ePersistent	Result
NULL	NCM_TIMESLOT _ARBITRARY	NCM_TIMESLOT _PERSISTENT	time slot blocks are not associated with a specific device, determined by the system software and retained when the system is rebooted.
NULL	NCM_TIMESLOT _SPECIFIC	NCM_TIMESLOT _PERSISTENT	time slot blocks are not associated with a specific device, determined by the user and retained when the system is rebooted.

pDeviceName	eArbitrary	ePersistent	Result
NULL	NCM_TIMESLOT_ARBITRARY	NCM_TIMESLOT_TRANSIENT	time slot blocks are not associated with a specific device, determined by the system software and released when the system is rebooted.
NULL	NCM_TIMESLOT_SPECIFIC	NCM_TIMESLOT_TRANSIENT	time slot blocks are not associated with a specific device, determined by the user and released when the system is rebooted.
device name set	NCM_TIMESLOT_ARBITRARY	NCM_TIMESLOT_PERSISTENT	time slot blocks are associated with a specific device, determined by the system software and retained when the system is rebooted.
device name set	NCM_TIMESLOT_SPECIFIC	NCM_TIMESLOT_PERSISTENT	time slot blocks are associated with a specific device, determined by the user and retained when the system is rebooted.

pDeviceName	eArbitrary	ePersistent	Result
device name set	NCM_TIMESLOT_ARBITRARY	NCM_TIMESLOT_TRANSIENT	time slot blocks are associated with a specific device, determined by the system software and released when the system is rebooted.
device name set	NCM_TIMESLOT_SPECIFIC	NCM_TIMESLOT_TRANSIENT	time slot blocks are associated with a specific device, determined by the user and released when the system is rebooted.

■ Cautions

- You must set the **pnNumOfBlocks** parameter to at least one. Similarly, you must allocate at least one element for the **pNCMTSBlock** parameter.
- If you are reserving more than one block of time slots, the number of reserved time slot blocks (set by the **pnNumOfBlocks** parameter) must match the number of reserved NCM_TS_BLOCK_STRUCT data structures (set by **pNCMTSBlock** parameter).
- The function returns NCME_BUFFER_TOO_SMALL if the memory reserved for the **pNCMTSBlock** parameter is not enough and returns the actual number of blocks needed in **pNumOfBlocks**. In this case, the caller should reallocate sufficient memory for the **pNCMTSBlock** parameter. Then make a second call to the **NCM_AllocateTimeslots( )** function.

■ Example

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

```
int                    iBusNumber = 0;
NCM_TS_BLOCK_STRUCT    *lpTimeSlotBlock = NULL;
int                    iNumOfBlocks = 1;
int                    iNumofTimeslot = 512;
NCMRetCode             ncmRc = NCM_SUCCESS;
int                    i = 0;
int                    timeslot = 0;
NCMDevice              deviceName;
Char                   DeviceString[] = "ThirdPartyDevice-XYZ#1";
deviceName.name = (char *) DeviceString;
deviceName.next = NULL;

/*example for arbitrary time slot allocation*/

do
{
   if (lpTimeSlotBlock)
      free(lpTimeSlotBlock);

   lpTimeSlotBlock = (NCM_TS_BLOCK_STRUCT *)malloc
       (sizeof(TS_BLOC_STRUCT) * lNumOfBlocks);
   memset(lpTimeSlotBlock, 0, sizeof(TS_BLOCK_STRUCT);

for (int i = 0; i < lNumOfBlocks; i++)
{
   lpTimeSlotBlock[i].version = NCM_TIMESLOT_VER_0100;
}

   //call NCM API function
   ncmRc = NCM_AllocateTimeslots(&deviceName, iNumofTimeslot,
NCM_TIMESLOT_ARBITRARY, NCM_TIMESLOT_TRANSIENT, lpTimeSlotBlock, &iNumOfBlocks);
}
while (ncmRc == NCME_BUFFER_TOO_SMALL)

if (ncmRc == NCM_SUCCESS)
{
for (i=0; i < dwNumOfBlocks; i++)
{
   for (timeSlot = lpTimeSlotBlock[i].start_time_slot;
       timeSlot < lTimeSlotBlock[i].start_time_slot +
      lpTimeSlotBlock[i].number_of_time_slots; timeSlot++)
    { //timeSlot is an acutal value of a time slot
 //do something with timeSlot
    }
}}

/*example for allocation of a set of user-defined time slots (10-310, 500-712)*/
iNumOfBlocks = 2;
```

```
lpTimeSlotBlock = (NCM_TS_BLOCK_STRUCT *)malloc
       (sizeof(NCM_TS_BLOCK_STRUCT) *iNumOfBlocks);
lpTimeSlotBlock[0].version = NCM_TIMESLOT_VER_0100;
lpTimeSlotBlock[0].struct_size = sizeof(NCM_TS_BLOCK_STRUCT);
lpTimeSlotBlock[0].start_time_slot = 10;
lpTimeSlotBlock[0].number_of_time_slots = 300;
lpTimeSlotBlock[1].version = NCM_TIMESLOT_VER_0100;
lpTimeSlotBlock[1].struct_size = sizeof(NCM_TS_BLOCK_STRUCT);
lpTimeSlotBlock[1].start_time_slot = 500;
lpTimeSlotBlock[1].number_of_time_slots = 212;

//call NCM API function
ncmRc = NCM_AllocateTimeslots(NULL, iNumofTimeslot, NCM_TIMESLOT_SPECIFIC,
NCM_TIMESLOT_TRANSIENT, lpTimeSlotBlock, &iNumOfBlocks);

if (ncmRc !=NCM_SUCCESS)
   /*process error*/

if (ncmRc == NCM_SUCCESS)
{
   /*print out the time slots in each block*/
   printf("For the request of %d time slots, %d block(s) of time slots have been
   allocated: \n", iNumofTimeslot, iNumOfBlocks);
   for (i=0; i < iNumOfBlocks; i++)
   {
      printf("Block %d: \n", i);
      for (timeslot = lpTimeSlotBlock[i].start_time_slot;
            + lpTimeSlotBlock[i].number_of_time_slots; timeslot ++)
      { /*timeSlot is an actual value of a time slot
            do something with timeslot */
           printf("%d", timeslot);
      }
       printf("\n");
     }
    }

/*you might choose to keep the starting time slot number in this transaction to
use it in releasing time slots. */

...
/*free the memory for time slot block after use*/
if (lpTimeSlotBlock)
      free(lpTimeSlotBlock);
```

■ Error Codes

Equate	Returned When
NCME_BUFFER_TOO_SMALL	buffer is of an insufficient size

Equate	Returned When
NCME_INVALID_INPUTS	invalid inputs
NCME_UNAVAILABLE_TIMESLOT	requested time slot is not available
NCME_INVALID_THIRDPARTY_DEVICE	specified third party device does not exist

See Also

- **NCM_DeallocateTimeslots( )**

Name: NCMRetCode NCM_DeallocateTimeslots(pDeviceName, nStartTimeSlot, bPermanent)

Inputs:

NCMDevice *pDeviceName	• pointer to a third party device name
int nStartTimeSlot	• starting time slot number of the block to be released
bool bPermanent	• determines whether or not the released time slots are persistent

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_DeallocateTimeslots( )** function is used to release a single block of time slots that had been reserved for third party devices.

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of the third party device that was associated with the allocated block of time slots. The device name must be the same name you associated with the time slot block using the **NCM_AllocateTimeslots( )** function. A device name for this parameter is optional. If the block of time slots you are deallocating was not associated with a third party device name, set this parameter to NULL.

Parameter	Description
nStartTimeSlot	starting time slot for the block of time slots that is to be deallocated. You can set this parameter to a time slot number or to NO_UNIQUE_ID
bPermanent	indicates whether or not the block of time slots will be permanently released. If the boolean is set to TRUE, then the block of time slots is permanently deallocated. If the boolean is set to FALSE then the deallocation will not be preserved when the system is rebooted (i.e. the time slot block will be automatically re-allocated when the system is rebooted.)

The following table summarizes the different parameter combinations for the **NCM_DeallocateTimeslots( )** function:

pDeviceName	nStartTimeSlot	bPermanent	Result
NULL	set to a valid value	TRUE	time slots within the block associated with the start time slot will be permanently released.
NULL	set to a valid value	FALSE	all time slots will be released and available for use by other devices until the system is rebooted. After the system is rebooted, the time slots will be reclaimed.
NULL	NO_UNIQUE_ID	TRUE	all time slots will be permanently released.

pDeviceName	nStartTimeSlot	bPermanent	Result
NULL	NO_UNIQUE_ID	FALSE	all time slots will be released and available for use by other devices until the system is rebooted. After the system is rebooted, the time slots will be reclaimed.
device name set	set to a valid value	TRUE	time slot block associated with the device and the starting time slot number will be permanently released.
device name set	set to a valid value	FALSE	time slot block associated with the device and the starting time slot number will be released and available for use by other devices until the system is rebooted. After the system is rebooted, the time slots will be reclaimed.
device name set	NO_UNIQUE_ID	TRUE	all time slots associated with the device will be permanently released.
device name set	NO_UNIQUE_ID	FALSE	all time slots associated with the device will be released and available for use by other devices until the system is rebooted. After the system is rebooted, the time slots will be reclaimed.

■ Cautions

The function cannot partially release the time slot block(s) that are allocated by the **NCM_AllocateTimeslots( )** function. You must deallocate a complete block of time slots each time the **NCM_DeallocateTimeslots( )** function is called.

■ Example

```
#include "NCMApi.h"

NCMRetCode   ncmRc=NCM_SUCCESS;

int    start_timeslot = 10;
bool   bPermanent = true;
NCMDevice deviceName;
char DeviceString[] = "ThirdPartyDevice-XYZ#1";
deviceName.name = (char *) DeviceString;
deviceName.next = NULL;

ncmRc = NCM_DeallocateTimeslots(NULL, start_timeslot, true);

if (ncmRc != NCM_SUCCESS)
{
   //process error
}
ncmRc = NCM_DeallocateTimeslots(&deviceName, NO_UNIQUE_ID, true);

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

...
```

■ Error Codes

Equate	Returned When
NCME_RELEASE_TIMESLOT	time slots could not be released
NCME_INVALID_THIRDPARTY_DEVICE	specified third party device does not exist
NCME_CTBB_LIB	*CTBBFace.dll* file is either not in the system or is the incorrect version

Equate	Returned When
NCME_RELEASE_TIMESLOT	failed to release the specified time slots
NCME_SYSTEMERROR	specific system resources were not found

See Also

- **NCM_AllocateTimeslots( )**

Name: NCMRetCode NCM_Dealloc(pncmString)

Inputs: NCMString* pncmString • pointer to an NCMString

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_Dealloc( )** function deallocates memory allocated for DCM API structures. For more information about memory allocation, see Section 2.4, “Dynamic Memory Allocation”, on page 17.

The function parameter is defined as follows:

Parameter	Description
pncmString	pointer to the structure occupying the memory to be freed; if the structure is the first in a linked list, this function deallocates the memory occupied by all structures in the list

■ Cautions

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 );
...
```

Name: NCMRetCode NCM_DeallocValue(pncmValueEx)

Inputs: NCMValueEx *pncmValueEx • pointer to an NCMValueEx structure

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_DeallocValue( )** function deallocates memory allotted for DCM API structures. For more information about memory allocation, see Section 2.4, “Dynamic Memory Allocation”, on page 17.

The function parameter is defined as follows:

Parameter	Description
pncmValueEx	pointer to the structure occupying the memory to be freed; if the structure is the first in a linked list, this function deallocates the memory occupied by all structures in the list

■ Cautions

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
{     // Process error
     ...
}

...

//
// Execute
//

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

...
```

Name: NCMRetCode NCM_DeleteEntry(pncmFamily, pncmDeviceUnique)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device 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

Mode: synchronous

■ Description

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

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

- **To remove configuration information for a 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.

The function parameters are defined as follows:

Parameter	Description

pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family
pncmDeviceUnique	pointer to the 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

■ 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, see Section 2.1, "The DCM API Architecture", on page 5.

■ 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
     ...
}

...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_GENERAL	a problem occurred retrieving the data

Name: NCMRetCode NCM_DetectBoards(pCallBackFunc, pnNumBrdsFound)

Inputs:

GL_PROG_FUNC *pCallBackFunc	• pointer to a callback function
int *pnNumBrdsFound	• pointer to 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

Mode: synchronous

■ Description

The **NCM_DetectBoards( )** function initiates a process that detects auto-detectable boards installed in the system. For more information about auto-detection, see Section 2.6.1, “Auto Detection”, on page 18.

The function parameters are defined as follows:

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

■ Cautions

Intel® Dialogic recommends using the **NCM_DetectBoards ( )** function for non-DM3 boards only.

All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )** function can be used to start the Intel® Dialogic System Service. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable Intel® Dialogic devices.

The callback function referenced by the **pCallBackFunc** pointer must have the following format:

- **int func_name (UINT percentageCompleted, const char *message);**
  - **percentageCompleted** is an unsigned integer variable that the board detection 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.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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

Name: NCMRetCode NCM_DetectBoardsEx(pdetectInfo, pdetectResult)

Inputs:

NCM_DETECTION_INFO *pdetectInfo	• pointer to a detection info structure
NCM_DETECTION_RESULT *pdetectResult	• pointer to detection result structure

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_DetectBoardsEx( )** function initiates auto -detection. This function initiates a process that detects any auto-detectable Intel® Dialogic boards installed on the system. For more information about auto-detection, see Section 2.6.1, “Auto Detection”, on page 18.

The function parameters are defined as follows:

Parameter	Description
pdetectInfo	Pointer to a NCM_DETECTION_INFO data structure. The NCM_DETECTION_INFO structure contains the structure size and two callback function pointers. (See Cautions below).
pdetectResult	Pointer to a NCM_DETECTION_RESULT structure. The NCM_DETECTION_RESULT structure contains the detection results, including total number of boards detected. It also contains the details of each detection result (NCM_DETECTION_DETAIL), such as each detector’s return code, return message and number of boards each detector detected.

■ Cautions

All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )** function can be used to start the Intel® Dialogic System Service. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable Intel® Dialogic devices.

The **pdetectInfo** pointer points to a NCM_DETECTION_INFO structure which contains the structure size and the following two callback function pointers:

- **NCM_CALLBACK_FCN:** points to the function which must be called to retrieve the progress of the detection routine. If this pointer is NULL, then no progress indication can be returned.

  The callback function must have the following format:

  int func_name (UINT percentageCompleted, const char *message);

  - **percentageCompleted** is an unsigned integer variable that the board detection 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".

  When using the NCM_DETECTION_INFO structure, the structure size must be filled into the first parameter of the structure, structSize.

- **NCM_PCDFILE_SELECTION_FCN:** used to determine the PCD file to be associated with each DM3 board. This pointer should be set to NULL for non-DM3 boards. If this pointer is NULL for DM3 boards, then a NCME_PCD_SELECTION error is returned.

  The selection function must have the following format:

  int func_name (NCMFileInfo *fileList, size_t numFiles, NCMDevInfo devinfo, int * index);

  - **fileList** is a list of PCD files (refer to the structure of NCMFileInfo)
  - **numFile** is the total number of PCD files in the list
  - **devInfo** Device Info (refer to the structure of NCMDevInfo)
  - **index** is the index number of chosen PCD file

For further details on data structure format, see Section 3.2, “Structures for Extended Functions”, on page 29.

■ 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 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;
}
}
}
```

■ /Error Codes

Equate	Returned When
NCME_GENERAL	a problem occurred retrieving the data
NCME_BRD_DETECT	auto detect fails
NCME_DETECTOR_FCN_NOT_FOUND	there was an error getting the detector function
NCME_DETECTOR_LIB_NOT_FOUND	there was an error loading the detector library
NCME_SP	invalid state transition
NCME_CTBB_DEVICE_DETECTED	error configuring the TDM Bus
NCME_PCD_SELECTION	Error, no PCD callback function with DM3 boards
NCME_REG_CALLBK	callback function cannot be registered with initialization process

■ See Also

- **NCM_DetectBoards( )**

Name: NCMRetCode NCM_EnableBoard(pncmFamily, pncmDeviceUnique, bEnable)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family name
	NCMDevice *pncmDeviceUnique	• pointer to a 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	
Mode:	synchronous	

■ Description

The **NCM_EnableBoard( )** function enables or disables device initialization when the Intel® Dialogic System Service is run. For more information about device initialization, see Section 2.8, “Starting and Stopping the Intel® Dialogic System Service”, on page 21.

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 initilization for a device: pncmFamily** and **pncmDeviceUnique** should point to an instantiated device.

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

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below)
pncmDeviceUnique	pointer to the 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
bEnable	Boolean variable specifying that 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.

■ 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
ncmRc = NCM_EnableBoard( &family, NULL, TRUE );

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

// Disable all devices
ncmRc = NCM_EnableBoard( NULL, NULL, FALSE );

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

...
```

■ Error Codes

Equate	Returned When
NCME_GENERAL	a problem occurred retrieving the data

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_SP	invalid state transition
NCME_BAD_DATA_LOC	the data destination is invalid or indeterminate
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage

See Also

- **NCM_IsBoardEnabled( )**

Name: NCMRetCode NCM_GetAllDevices(pncmFamily, ppncmDeviceModel)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a family of devices
NCMDevice **ppncmDeviceModel	• address of pointer where device model names will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetAllDevices( )** function gets a list of installable device models for a family. For information about using this function to fill all the structures you need to instantiate and modify configuration parameter values, see Section 2.3.1, “Populating Required Structures”, on page 11.

NOTE: This function provides a list of installable device models from the DCM Catalog. This function does **not** return a list of installed devices. See **NCM_GetInstalledDevices( )** for that functionality.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family
ppncmDeviceModel	address of the pointer to the list to be filled with structures containing device model names

■ Cautions

The DCM 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.

■ 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 );

...
```

■ Error Codes

Equate	Returned When
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

■ See Also

- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetProperties( )**
- **NCM_GetValueRange( )**
- **NCM_GetVariables( )**

Name: NCMRetCode NCM_GetAllFamilies(ppncmFamily)

Inputs: NCMFamily **ppncmFamily
- address of a pointer in which the list of device families will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetAllFamilies( )** function gets a list of installable families. For information about using this function to fill all the structures you need to instantiate and modify configuration parameter values, see Section 2.3.1, “Populating Required Structures”, on page 11.

The function parameter is defined as follows:

Parameter	Description
ppncmFamily	address of the pointer to the list to be filled with family structures

■ Cautions

The information provided by this function is limited to the DCM Catalog. To retrieve information that is instantiated in your system configuration, use **NCM_GetInstalledFamilies( )**. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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

See Also

- **NCM_GetAllDevices( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**
- **NCM_GetProperties( )**
- **NCM_GetValueRange( )**
- **NCM_GetValueRangeEx( )**
- **NCM_GetVariables( )**

Name: NCMRetCode NCM_GetClockMasterFallbackList(pncmBus, pnumInList, ppfallbackList)

Inputs:	NCMDevice *pncmBus	• address of a pointer to a specific bus name
Outputs:	Int *pnumInList	• address of a pointer to total number of boards in list to be returned
	NCMDevice **ppfallbackList	• address of a pointer to list of devices in the fallback list to be returned

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetClockMasterFallbackList( )** function returns the clock fallback list. This function fills a pointer to a pointer with the beginning address of a list of devices in the clock fallback list. In addition, this function also returns the total number of devices in the list. For more information about the master clock fallback list, see Section 2.10, “Clock Master Fallback List”, on page 24.

The function parameters are defined as follows:

Parameter	Description
pncmBus	pointer to a structure containing the specific bus name, for example, “Bus-0”
pnumInList	pointer to the total number of devices in the clock fallback list, including the Primary and Secondary
ppfallbackList	address to pointer of the list of devices to be returned

■ Cautions

Currently, only a single bus is supported. Therefore, the bus name for the parameter **pncmBus** should be “Bus-0”.

The **pnumInList** includes both the Primary Clock Master device and the Secondary Clock Master device.

The DCM 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( )** function.

■ Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

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

NCMValue *     pfallbackList = NULL;
int total;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetClockMasterFallbackList( &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 );
```

■ Error Codes

Equate	Returned When
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

■ See Also

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

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

Mode: synchronous

■ Description

The **NCM_GetCspCountries( )** function gets a list of supported countries from the DCM Catalog. For more information about modifying country-specific parameter values, see Section 2.7, “Working with Country Specific Parameters”, on page 20.

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® Dialogic devices may be configured.

The function parameter is defined as follows:

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

■ Cautions

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetCspCountryCode( )**

- **NCM_GetCspCountryName( )**
- **NCM_DeallocValue( )**
- **NCM_Dealloc( )**

Name: NCMRetCode NCM_GetCspCountryCode(szCountryName, ppncmCode)

Inputs:

char *szCountryName	• country name
NCMValue **ppncmCode	• address of pointer where ISO country code will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

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 passed to it. For more information about modifying country-specific parameter values, see Section 2.7, “Working with Country Specific Parameters”, on page 20.

The function parameters are defined as follows:

Parameter	Description
szCountryName	ASCII-Z string containing country name
ppncmCode	address of the pointer that will point to the ISO country code

■ Cautions

The DCM 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( )** function.

■ 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 );

...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetCspCountries( )**

- **NCM_GetCspCountryName( )**
- **NCM_DeallocValue( )**
- **NCM_Dealloc( )**

Name: NCMRetCode NCM_GetCspCountryName(szCountryCode, ppncmName)

Inputs:

char *szCountryCode	• country code
NCMValue **ppncmName	• address of pointer where country name will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetCspCountryName( )** function gets the country name for a country code from the DCM Catalog. For more information about modifying country-specific parameter values, see Section 2.7, “Working with Country Specific Parameters”, on page 20.

The function parameters are defined as follows:

Parameter	Description
szCountryCode	ASCII-Z string containing ISO country code
ppncmName	address of the pointer that will point to the country name

■ Cautions

The DCM 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( )** function.

■ Example

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

```
...

NCMValue *     pName = NULL;

//
// Execute
//

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

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

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

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetCspCountries( )**
- **NCM_GetCspCountryCode( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

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 **szFeatures**
	NCMValue **ppncmValue	• address of the pointer to the value of the variable specified by **pncmVariable**
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
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 contained in the **Features** configuration parameter. The value is either extracted from the string of values passed in **szFeatures** or found in the current system configuration.

For more information about modifying country-specific parameter values, see Section 2.7, “Working with Country Specific Parameters”, on page 20.

The function parameters are defined as follows:

Parameter	Description

szCountryCode	ASCII-Z string containing ISO country code
szFeatures	ASCII-Z string containing either a pointer to a comma-separated list of country specific configuration parameters or NULL
pncmVariable	pointer to the country specific configuration parameter for which a value will be returned
ppncmValue	address of the pointer that will contain the value of the country specific configuration parameter value

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

- **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 **szFeatures** to NULL.

■ Cautions

The set of country-specific configuration parameters retrieved by **NCM_GetCspFeaturesValue( )** is determined by the value of the Country parameter.

The DCM 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 the **NCM_DeallocValue( )** function.

■ 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 );

...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

- **NCM_GetCspFeaturesValueRange( )**
- **NCM_GetCspFeaturesVariables( )**

Name: NCMRetCode NCM_GetCspFeaturesValueRange (szCountryCode, pncmVariable, ppncmRange)

Inputs:

char *szCountryCode	• country code
NCMVariable *pncmVariable	• pointer to a country specific configuration parameter
NCMValue **ppncmRange	• address of pointer where range will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

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

For more information about modifying country-specific parameter values, see Section 2.7, “Working with Country Specific Parameters”, on page 20.

The function parameters are defined as follows:

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

■ Cautions

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_Dealloc( )**
- **NCM_GetCspFeaturesValue( )**
- **NCM_GetCspFeaturesVariables( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetCspFeaturesVariables(szCountryCode, ppncmVariables)

Inputs:

char *szCountryCode	• country code
NCMVariable **ppncmVariables	• address of 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

Mode: synchronous

■ Description

The **NCM_GetCspFeaturesVariables( )** function gets values for country specific configuration parameters for a specified country from the DCM Catalog.

The function parameters are defined as follows:

Parameter	Description
szCountryCode	ASCII-Z string containing ISO country code; for information about how to find the country code, see Section 2.7, “Working with Country Specific Parameters”, on page 20.
ppncmVariables	address of the pointer that will point to the “Features” components

■ Cautions

The country specific configuration parameters retrieved for the Country property vary with the value of the Country parameter. For all other properties, the

configuration parameters are retrieved with the **NCM_GetVariables( )** function; the configuration parameters retrieved in this way are those that are applicable to the device name input to this function. For example, the **System** property for a D/41D device contains the **D41DAddress**, **D41DInterrupt**, and **ISABusWidth** parameters, whereas the same property for a D/240SC device contains **BLTAddress**, **BLTInterrupt**, **BLTId**, and **ISABusWidth**. In the case of country-specific configuration parameters, however, it is the value of the **Country** configuration parameter that determines which country-specific configuration parameters belong to the property.

The information provided by this function is limited to the DCM Catalog. To retrieve information that is instantiated in your system configuration, use **NCM_GetCspFeaturesValue( )**.

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetCspFeaturesValue( )**
- **NCM_GetCspFeaturesValueRange( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetDialogicDir (szKey, size, pDlgcDir)

Inputs:

char *szKey	• the Dialogic path key
int *size	• address of buffer size
char *pDlgcDir	• pointer to the Dialogic directory to be returned

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

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

The function parameters are defined as follows:

Parameter	Description
szKey	ASCII Z string containing specific dialogic path key value. The possible key values with their corresponding path are defined as follows: • DIALOGICDIR: "...\Dialogic\" • DLCFGPATH: "...\Dialogic\cfg" • DLFWLPATH: "...\Dialogic\data" • DLINFPATH: "...\Dialogic\inf" • DLLOGPATH: "...\WINNT\System32" • DNASDKDIR: "...\Dialogic\bin" • MRFDir: "\Dialogic\mrf"

Parameter	Description
size	the buffer size for **pDlgcDir** to be returned
pDlgDir	pointer to the Dialogic directory to be returned

■ Cautions

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

■ Example

```
#include "NCMApi.h"

...

NCMRetCode ncmRc = NCM_SUCCESS;

// get Dialogic data 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 );

...
```

■ Error Codes

Equate	Returned When
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

Name: NCMRetCode NCM_GetDlgSrvStartupMode(pncmStartupMode)

Inputs: NCMDlgSrvStartupMode *pncmStartupMode
- pointer where Startup Mode will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetDlgSrvStartupMode( )** function gets the startup mode of Intel® Dialogic System Service. For more information about the Intel® Dialogic System Service, see Section 2.8, “Starting and Stopping the Intel® Dialogic System Service”, on page 21.

The function parameter is defined as follows:

Parameter	Description
pncmStartupMode	points to the current startup mode setting for the Dialogic System Service. It may be one of the following modes: • **NCM_DLGSRV_AUTO**: service starts automatically at system startup time • **NCM_DLGSRV_MANUAL**: service must be manually started • **NCM_DLGSRV_DISABLED**: service is disabled • **NCM_DLGSRV_STARTUP_UNDEFINED**: startup mode of service is undefined

■ Cautions

None

■ Example

```
#include "NCMApi.h"

...

NCMDlgSrvStartupMode      startupMode = NCM_DLGSRV_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
     ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service
NCME_QUERY_SVC_STATUS	an error occurred querying the status of the Intel® Dialogic System Service
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetDlgSrvState( )**
- **NCM_SetDlgSrvStartupMode( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

Name: NCMRetCode NCM_GetDlgSrvState(pncmSrvState)

Inputs: NCMDlgSrvState *pncmSrvState
- pointer where Intel® Dialogic System Service state will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetDlgSrvState( )** function gets the Intel® Dialogic System Service state. For more information about the Intel® Dialogic System Service, see Section 2.8, “Starting and Stopping the Intel® Dialogic System Service”, on page 21.

This function returns the state of the Intel® Dialogic System Service by filling the passed pointer.

The function parameter is defined as follows:

Parameter	Description
pncmSrvState	points to the current state of the Intel® Dialogic System Service. Refer to the Windows* documentation for possible states of a service.

■ Cautions

None

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service
NCME_QUERY_SVC_STATUS	an error occurred querying the status of the Intel® Dialogic System Service
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetDlgSrvStartupMode( )**
- **NCM_SetDlgSrvStartupMode( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

Name: NCMRetCode NCM_GetDlgSrvStateEx(pncmSrvState)

Inputs: SERVICE_STATUS *srvcStatus • pointer to Win32 SERVICE_STATUS structure

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetDlgSrvStateEx( )** function gets the Intel® Dialogic System Service state. For more information about the Intel® Dialogic System Service, see Section 2.8, “Starting and Stopping the Intel® Dialogic System Service”, on page 21.

This function returns the state of the Intel® Dialogic System Service by filling the passed pointer.

This is the new version of the **NCM_GetDlgSrvState( )** function. It replaces the input parameter by the standard SERVICE_STATUS structure.

The function parameters are defined as follows:

Parameter	Description
pncmSrvState	points to the current state of the Intel® Dialogic System Service. Refer to the Windows documentation for possible states of a service.

■ Cautions

None

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service
NCME_QUERY_SVC_STATUS	an error occurred querying the status of the Intel® Dialogic System Service
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetDlgSrvState( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

Name: NCMRetCode NCM_GetErrorMsg(ncmRcIn, ppncmErrorMsg)

Inputs:

NCMRetCode ncmRcIn	• NCM return code
NCMErrorMsg **ppncmErrorMsg	• address of pointer where error message will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetErrorMsg( )** function gets the error message for an error code. Every function in the DCM API returns a code indicating the success or failure of the function. This function accepts one of those codes and returns the appropriate text string for it.

NOTE: Refer to the System Log of the Windows Event Viewer for a detailed explanation of DCM API error messages.

The function parameters are defined as follows:

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

■ Cautions

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage

See Also

- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetInstalledDevices (pncmFamily, ppncmDeviceUnique)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a device family name
NCMDevice **ppncmDeviceUnique	• address of pointer where unique device names will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetInstalledDevices( )** function gets all instantiated devices for a family.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name whose devices should be returned; the value contained in the structure must be an instantiated family (see Cautions below)
ppncmDeviceUnique	address of the pointer to the list to be filled with structures containing unique device names; the unique device names will be the same names you used to add the devices to the system configuration with the **NCM_AddDevice( )** function

■ Cautions

This function enables you to determine what devices are instantiated in the system configuration. To find out what devices are installable from the DCM Catalog, use the **NCM_GetAllDevices( )** function. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The DCM 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( )** function.

■ Example

```
#include "NCMApi.h"

...

//
// Prepare inputs
//

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

NCMDevice *     pDevices = 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 );
...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_INVALID_FAMILY	the family name is invalid

■ See Also

- **NCM_GetInstalledFamilies( )**
- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetInstalledFamilies (ppncmFamily)

Inputs: NCMFamily **ppncmFamily • address of pointer where the instantiated board families will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetInstalledFamilies( )** function gets all instantiated device families.

This function fills a pointer to a pointer with the beginning address of a list of instantiated device families.

The function parameter is defined as follows:

Parameter	Description
ppncmFamily	address of the pointer to the list to be filled with family structures

■ Cautions

This function enables you to determine what families are instantiated in the system configuration. To find out what families are available in the DCM Catalog, use the **NCM_GetAllFamilies( )** function. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The DCM 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( )** function.

■ Example

```
#include "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 );

...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetInstalledDevices( )**
- **NCM_GetValue( )**
- **NCM_GetValueEx( )**

- **NCM_SetValue( )**
- **NCM_SetValueEx( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetProperties(pncmFamily, pncmDevice, ppncmProperties)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a device family name
NCMDevice *pncmDevice	• pointer to a structure containing a device name
NCMProperty **ppncmProperties	• address of pointer where properties will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

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 need to instantiate and modify configuration parameter values, see Section 2.3.1, "Populating Required Structures", on page 11.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family (see Cautions below.)
pncmDevice	pointer to the structure containing a device name; the device name can be either 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
ppncmProperties	address of the pointer to the list to be filled with property structures

■ Cautions

The **pncmFamily** and **pncmDevice** 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, see Section 2.3.1, “Populating Required Structures”, on page 11.

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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

■ See Also

- **NCM_GetAllDevices( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetValueRange( )**
- **NCM_GetVariables( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetPropertyAttributes (pncmFamily, pncmDevice, pncmProperty, pncmPropAttribs)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family name
	NCMDevice *pncmDevice	• pointer to a structure containing a device name
	NCMProperty *pncmProperty	• pointer to structure containing a property section
	NCMPropertyAttribute *pncmPropAttribs	• pointer to a structure containing the property's attributes
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_GetPropertyAttributes( )** function gets a properties attributes.

This function queries the system configuration to return a property's attributes-(VISIBLE/HIDDEN).

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the property name
pncmPropAttribs	pointer to the property's attributes to be returned

■ Cautions

None

■ 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
    ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	invalid inputs
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage

Name: NCMRetCode NCM_GetVersionInfo (OUT NCMSysVersion *psysver)

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h
NCMTypes.h

Mode: synchronous, asynchronous

■ Description

The **NCM_GetVersionInfo( )** function gets OS and Intel® Dialogic System information. This function retrieves the Operating System and Intel® Dialogic System Service information for local and remote computers.

■ Cautions

None

■ Example

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

//
//Execute
//

NCMRetCode ncmRc = NCM_GetVersionInfo (*psysver);

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

Error Codes

Equate	Returned When
NCME_REMOTE_REG_ERROR	error opening registry key of remote computer

}Name: NCMRetCode NCM_GetTDMBusValue(pncmBus, pvariable, ppvalue)

Inputs:	NCMDevice *pncmBus	• pointer to the structure containing a specific bus name
	NCMVariable *pvariable	• pointer to a structure containing the variable name to be retrieved
Outputs	NCMValue **ppvalue	• address to pointer of value to be returned

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetTDMBusValue( )** function gets the parameter value of the TDM bus. This function also allows the user to retrieve the value of user defined and resolved variables in the TDM Bus family. For additional information about using this function, see Section 2.10, “Clock Master Fallback List”, on page 24.

The function parameters are defined as follows:

Parameter	Description
pncmBus	pointer to the structure containing a specific bus name, for example, “Bus-0”
pvariable	name of the variable
ppvalue	pointer to the name of the value to be returned

■ Cautions

The variable must be a valid parameter under the TDM Bus Configuration, otherwise the function returns NCME_INVALID_INPUTS.

Memory to hold the returned data is allocated in this function. To avoid memory leaks, the application must deallocate the memory by calling **NCM_Dealloc( )**.

Currently, only a single bus is supported. Therefore, the bus name for the parameter **pncmBus** should be “Bus-0”.

■ 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 );
```

■ Error Codes

Equate	Returned When
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

■ See Also

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

Name: NCMRetCode NCM_GetThirdPartyDeviceBusCaps(pDeviceName, pTDMBusCapabilities)

Inputs:

NCMDevice *pDeviceName	• third party device name
NCM_TDM_BUSCAPS *pTDMBusCapabilities	• pointer to the TDM bus capabilities of the third party device

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetThirdPartyDeviceBusCaps( )** function returns the TDM bus capabilities of a third party device. The TDM bus capabilities of a third party device is defined when the **NCM_AddThirdPartyDevice( )** function is called to add the device to the system.

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of a third party device. The device name must be the same name you used to add the third party device to the system configuration with the **NCM_AddThirdPartyDevice( )** function
pTDMBusCapabilities	pointer to the NCM_TDM_BUSCAPS data structure that holds the devices TDM bus capabilities

■ Cautions

You cannot set the **pDeviceName** parameter to NULL.

■ Example

```
#include "NCMApi.h"

NCM_TDM_BUSCAPS busCaps;
NCMRetCode      nmcRc=NCM_SUCCESS;

NCMDevice deviceName;
Char DeviceString[] = ”ThirdPartyDevice-XYZ#1”;
deviceName.name = (char *) DeviceString;
deviceName.next = NULL;

//call NCM API function:
ncmRc = NCM_GetThirdPartyDeviceBusCaps(deviceName, &busCaps);

if (ncmRc !=NCM_SUCCESS)
{
   /*process error*/
}
else
{
   /*process success*/
}

...
```

■ Error Codes

Equate	Returned When
NCME_INALID_INPUTS	invalid inputs

Name: NCMRetCode NCM_GetValue(pncmFamily, pncmDevice, pncmProperty, pncmVariable, ppncmValue)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family name
	NCMDevice *pncmDevice	• pointer to a structure containing a device name
	NCMProperty *pncmProperty	• pointer to a structure containing a property section
	NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
	NCMValue **ppncmValue	• address of pointer where value will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetValue( )** function gets an instantiated or default parameter value. This function enables you to determine either the instantiated value of a configuration parameter in the system configuration or the default value of a configuration parameter in the DCM Catalog.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name

Parameter	Description
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the property name
pncmVariable	pointer to the structure containing the configuration parameter name
ppncmValue	an address of the pointer to be filled with the configuration parameter value

Whether this function gets an instantiated value from the system configuration or a default value from the DCM Catalog depends on whether **pncmDevice** points to a unique device name or a device model name:

- **To get an instantiated configuration parameter value: pncmDevice** must point to 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.
- **To get the default configuration parameter value**: **pncmDevice** must point to an installable device model name; the device model name must be for a model that is defined in the DCM Catalog.

For more information about the distinction between device model names and unique device names, see Section 2.3.4, “Device Model Names and Unique Device Names”, on page 14.

■ Cautions

The DCM 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( )** function.

■ 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 );

...
```

■ Error Codes

Equate	Returned When

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

■ See Also

- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetValueEx( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetValueEx(pncmFamily, pncmDevice, pncmVariable, ppncmValueEx)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family name
	NCMDevice *pncmDevice	• pointer to a structure containing a device name
	NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
	NCMValueEx **ppncmValueEx	• address of pointer where value will be output
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_GetValueEx( )** function gets an instantiated or default parameter value. This function enables you to determine either the instantiated value of a configuration parameter in the system configuration or the default value of a configuration parameter in the DCM Catalog.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the configuration parameter name
ppncmValueEx	address of the pointer to the structure to be filled with the configuration parameter value

Whether this function gets an instantiated value from the system configuration or a default value from the DCM Catalog depends on whether **pncmDevice** points to a unique device name or a device model name:

- **To get an instantiated configuration parameter value: pncmDevice** must point to 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.
- **To get the default configuration parameter value**: **pncmDevice** must point to an installable device model name; the device model name must be for a model that is defined in the DCM Catalog.

For more information about the distinction between device model names and unique device names, see Section 2.3.4, “Device Model Names and Unique Device Names”, on page 14.

■ Cautions

The DCM 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( )** function.

■ 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 );

...
```

■ Error Codes

Equate	Returned When
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

■ See Also

- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetValue( )**
- **NCM_SetValue( )**
- **NCM_SetValueEx( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetValueRange(pncmFamily, pncmDevice, pncmProperty, pncmVariable, ppncmValues)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a family of boards
NCMDevice *pncmDevice	• pointer to a structure containing a device
NCMProperty *pncmProperty	• pointer to a structure containing a property
NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
NCMValue **ppncmValues	• address of pointer where range will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetValueRange( )** function gets the value range for a parameter.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family (see Cautions below)

Parameter	Description
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the property name; the value of the structure must be an installable property (see Cautions below)
pncmVariable	pointer to the structure containing the configuration parameter; the value of the structure must be an installable configuration parameter (see Cautions below)
ppncmValues	address of the pointer to be filled with the configuration parameter values

■ Cautions

This function provides the range of values that can be set for an installable configuration parameter. To determine the actual value of a configuration parameter instantiated in your system configuration, use **NCM_GetValue( )**. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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

Equate	Returned When
NCME_BAD_INF	there was an error parsing the DCM Catalog
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetAllDevices( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetProperties( )**
- **NCM_GetVariables( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetValueRangeEx(pncmFamily, pncmDevice, pncmVariable, ppncmRangeEx)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a family of boards
NCMDevice *pncmDevice	• pointer to a structure containing a device
NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
NCMValueEx **ppncmRangeEx	• address of pointer where range will be output

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetValueRangeEx( )** function gets the value range of a parameter.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family (see Cautions below)
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the configuration parameter; the value of the structure must be an installable configuration parameter (see Cautions below)
ppncmRangeEx	an address of the pointer to be filled with the configuration parameter value range

■ Cautions

This 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 system configuration, use **NCM_GetValue( )**. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The DCM 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( )** function.

■ 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 );

...
```

■ Error Codes

Equate	Returned When
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	the values of the parameters supplied are invalid

■ See Also

- **NCM_GetAllDevices( )**
- **NCM_GetAllFamilies( )**
- **NCM_GetProperties( )**
- **NCM_GetValueRange( )**

- **NCM_GetVariables( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetVariables(pncmFamily, pncmDevice, pncmProperty, ppncmVariables)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a family of boards
NCMDevice *pncmDevice	• pointer to a structure containing a device name
NCMProperty *pncmProperty	• pointer to a 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

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.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family (see Cautions below)

Parameter	Description
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the property name; the value of the structure must be an installable property (see Cautions below)
ppncmVariables	address of the pointer to the list to be filled with configuration parameter structures

■ Cautions

This function provides the 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( )**. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, “The DCM API Architecture”, on page 5.

The **NCM_GetVariables ( )** function can be used to retrieve a list of all global configuration parameters from the DCM Catalog by setting both the NCMFamily and NCMDevice structures to NULL.

The DCM 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( )** function.

■ 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 );
...
```

■ Error Codes

Equate	Returned When
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

See Also

- **NCM_GetAllDevices( ),**
- **NCM_GetAllFamilies( )**
- **NCM_GetProperties( )**
- **NCM_GetValueRange( )**
- **NCM_GetValueRangeEx( )**
- **NCM_Dealloc( )**
- **NCM_DeallocValue( )**

Name: NCMRetCode NCM_GetVariableAttributes (pncmFamily, pncmDevice, pncmVariable, pncm VariableAttribs)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a family of boards
	NCMDevice *pncmDevice	• pointer to a structure containing a device name
	NCMVariable *pncmVariable	• pointer to a structure containing a property section
	NCMVariableAttributes *pncmVariableAttribs	• pointer to a structure containing the variable's attributes

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_GetVariableAttributes( )** function returns the parameter's attributes.

This function fills a pointer to a pointer with the beginning address of a list of variables for a particular property section.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	points to the family of boards to which the device belongs
pncmDevice	pointer to the device for which the variables should be returned
pncmVariable	pointer to a variable

Parameter	Description
pncmVariableAttribs	pointer to the where the variable's attributes are returned

■ Cautions

The global or family-level calls are not fully supported. Default values are returned with return code of NCM_SUCCESS.

■ Example

```
#include "NCMApi.h"

...
//
// Prepare inputs
//

NCMFamily family;
family.name = "D/x1E";
family.next = NULL;

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

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

NCMVariableAttribute pVariableAttribs;

//
// Execute
//

NCMRetCode     ncmRc = NCM_GetVariableAttributes( &family, &device,
                                                  &variable, &pVariableAttribs );
if ( ncmRc == NCM_SUCCESS )
{
// Process Attributes
   ...
}
else
{     // Process error
     ...
}
```

. . .

■ Error Codes

Equate	Returned When
NCME_DATA_NOT_FOUND	Requested data not found in NCM data storage
NCME_INVALID_INPUTS	Invalid inputs

Name: NCMRetCode NCM_IsBoardEnabled(pncmFamily, pncmDeviceUnique, pbEnabled)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family name
	NCMDevice *pncmDeviceUnique	• pointer to a structure containing unique device name
	BOOL *pbEnabled	• pointer to a Boolean
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_IsBoardEnabled( )** function determines if a device is to be initialized.

This function queries the system configuration to determine if a device is enabled to be initialized. If so, the address referenced by the **pbEnabled** pointer is set to TRUE; otherwise it is set to FALSE. For more information about device initialization, see Section 2.8, “Starting and Stopping the Intel® Dialogic System Service”, on page 21.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below)

Parameter	Description
pncmDeviceUnique	pointer to the 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
pbEnabled	pointer to a Boolean variable indicating that device is to be initialized (TRUE) or is not to be initialized (FALSE)

■ Cautions

This function queries devices that are 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, see Section 2.1, "The DCM API Architecture", on page 5.

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
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
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage

■ See Also

- **NCM_EnableBoard( )**

Name: NCMRetCode NCM_IsEditable(pncmFamily, pncmDevice, pncmProperty, pncmVariable, pbEditable)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device family
	NCMDevice *pncmDevice	• pointer to a structure containing a device name
	NCMProperty *pncmProperty	• pointer to a structure containing a property section
	NCMVariable *pncmVariable	• pointer to a 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

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.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value of the structure must be an installable family (see Cautions below)
pncmDevice	pointer to the structure containing a device name; the device name can be either 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	pointer to the structure containing the property name; the value of the structure must be an installable property (see Cautions below)
pncmVariable	pointer to the structure containing the configuration parameter; the value of the structure must be an installable configuration parameter (see Cautions below)
pbEditable	a pointer 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 current DCM Catalog.

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
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

Name: NCMRetCode NCM_QueryTimeslots(pDeviceName, nStartTimeSlotNum, ePersistent, pnNumOfBlocks, pNCMTSBlock)

Inputs:	NCMDevice *pDeviceName	• pointer to a third party device name
	int nStartTimeSlotNum	• starting time slot number of the block to be queried
	NCMTSReserveType ePersistent	• type of time slots to be queried (persistent, transient or all)
	int *pnNumOfBlocks	• pointer to the number of time slot blocks to be returned
	NCM_TS_BLOCK_STRUCT *pNCMTSBlock	• pointer to the data structure containing the time slots being queried
Returns:	NCM_SUCCESS if success NCM error code if failure	
Includes:	NCMApi.h	
Mode:	synchronous	

■ Description

The **NCM_QueryTimeslots( )** function allows you to query either the entire Intel® Dialogic system for reserved time slots or a specific third party device for its associated time slots. The **ePersistent** parameter determines whether the query returns persistent time slots only, transient time slots only or all time slots (i.e both persistent and transient types).

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of the third party device that you are querying. The device name must be the same name you used to add the third party device to the system configuration with the **NCM_AddThirdPartyDevice( )** function A device name for this parameter is optional. Set this parameter to NULL to return all reserved third party device time slots within the Intel® Dialogic system.
nStartTimeSlotNum	indicates the starting time slot for the block of time slots that is to be queried. This parameter can either be set to a valid integer or NO_UNIQUE_ID (if you are not associating the query with a specific block of time slots).
ePersistent	determines whether the query will return persistent time slots only, transient time slots only or all time slots (i.e both transient and persistent). Valid values are as follows: • NCM_TIME_SLOT_PERSISTENT - only persistent time slots are returned • NCM_TIME_SLOT_TRANSIENT - only transient time slots are returned • NCM_TIME_SLOT_ALL - all time slots are returned (i.e. both transient and persistent)
pnNumOfBlocks	points to the number of time slot blocks to be returned
pNCMTSBlock	points to the NCM_TS_BLOCK_STRUCT data structure that is returned by the function

The following table summarizes the supported parameter combinations for the **NCM_QueryTimeslots( )** function:

pDeviceName	nStartTimeSlotNum	ePersistent	Result
NULL	set to a valid value	NCM_TIMESLOT_ALL	returns all reserved time slots associated with the block that starts with the **nStartTimeSlotNum** value
NULL	NO_UNIQUE_ID	NCM_TIMESLOT_PERSISTENT	returns all reserved time slots in the system that are persistent
NULL	NO_UNIQUE_ID	NCM_TIMESLOT_TRANSIENT	returns all reserved time slots in the system that are transient
NULL	NO_UNIQUE_ID	NCM_TIMESLOT_ALL	returns all time slots that have been reserved for third party devices (i.e. persistent and transient time slots for all third party devices are returned)

pDeviceName	nStartTimeSlotNum	ePersistent	Result
device name set	NO_UNIQUE_ID	NCM_TIMESLOT_PERSISTENT	returns all reserved time slots that are persistent and associated with the third party device indicated by the **pDeviceName** parameter
device name set	set to a valid value	NCM_TIMESLOT_PERSISTENT	returns reserved time slots within the block that starts with the **nStartTimeSlotNum** parameter, are associated with the third party device indicated by the **pDeviceName** parameter and are persistent

■ Cautions

None

■ Example

```
#include "NCMApi.h"

NCMDevice device.Name;
char DeviceString[] = "ThirdPartyDevice-XYZ#1";
deviceName.name = (char *) DeviceString;
deviceName.next = NULL;
```

```
NCMRetCode ncmRc = NCM_SUCCESS;
NCMTSReserveType reserveType = NCM_TIMESLOT_PERSISTENT;
DWORD numBlocks = 1;
NCM_TS_BLOCK_STRUCT *pTimeslotBlock = NULL;

int nStartTimeSlotNum = 10;

ncmRc = NCM_QueryTimeslots(&deviceName, nStartTimeSlotNum, reserveType,
&numBlocks, pTimeslotBlock);

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

■ Error Codes

Equate	Returned When
NCME_NO_TIMESLOT	specified time slots queried do not exist
NCME_INVALID_THIRDPARTY_DEVICE	specified third party device does not exist
NCME_DATA_NOT_FOUND	data not found
NCME_BUFFER_TOO_SMALL	allocated buffer is too small
NCME_SYSTEMERROR	specific system resources were not found

■ See Also

- **NCM_AllocateTimeslots( )**
- **NCM_DeallocateTimeslots( )**

Name: NCMRetCode NCM_RemoveThirdPartyDevice(pDeviceName)

Inputs: NCMDevice *pDeviceName
- pointer to the third party device name that will be removed

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_RemoveThirdPartyDevice( )** function deletes a third party device's TDM bus configuration information from the Intel® Dialogic system. This function also releases all time slots that are allocated to the deleted third party device.

NOTE: If the third party device you are removing is the system's primary clock master, you must define a new primary clock master before calling the **NCM_RemoveThirdPartyDevice( )** function.

The function parameters are defined as follows:

Parameter	Description
pDeviceName	pointer to the data structure containing the name of the third party device being deleted. The device name must be the same name you used to add the third party device to the system configuration with the **NCM_AddThirdPartyDevice( )** function

■ Cautions

You cannot set the **pDeviceName** parameter to NULL.

■ Example

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

```
   NCMDevice deviceName;
   Char DeviceString[] = "ThirdPartyDevice-XYZ#1";
   deviceName.name = (char *)DeviceString;
   deviceName.next = NULL;

   //call NCM API function
   ncmRc = NCM_RemoveThirdPartyDevice(deviceName);

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

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	invalid inputs
NCME_INVALID_THIRDPARTY_DEVICE	specified third party device does not exist
NCME_CTBB_LIB	*CTBBface.dll* file either cannot be found in the system or is the incorrect version
NCME_CTBB_DEVICESDETECTED	re-detection of devices failed

■ See Also

- **NCM_AddThirdPartyDevice( )**

Name: NCMRetCode NCM_SetClockMasterFallbackList(pncmBus, pfallbackList)

Inputs: NCMDevice *pncmBus • pointer to a structure containing a specific bus name

NCMDevice *pfallbackList • pointer to list of devices to be set

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_SetClockMasterFallbackList( )** function sets the clock fallback list.

This function will issue a CTBB_UserApply to validate the changes. If the CTBB returns an error, then the list will not be set and previous values remain unchanged. For more information about defining the master clock fallback list, see Section 2.10, “Clock Master Fallback List”, on page 24.

The function parameters are defined as follows:

Parameter	Description
pncmBus	pointer to the a structure containing a specific bus name, for example, “Bus-0”
pfallbackList	pointer to device list to be set in the clock fallback list

■ Cautions

NCM Device structure to be passed in is of a single link list form.

Currently, only one bus is supported. Therefore, the parameter **pncmBus** should equal “Bus-0”.

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

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, the system will choose both the Primary and Secondary Clock Master.

■ 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
     ...
}
```

■ Error Codes

Equate	Returned When
NCME_GENERAL	a problem occurred retrieving the data
NCME_DATA_NOT_FOUND	requested data not found in NCM data storage
NCME_CTBB_LIB	the system failed to load the CTBB library
NCME_CTBB_USERAPPLY	error updating the TDM Bus parameters
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_MEM_ALLOC	memory could not be allocated to perform the function

■ See Also

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

Name: NCMRetCode NCM_SetDlgSrvStartupMode(ncmStartupMode)

Inputs: NCMDlgSrvStartupMode ncmStartupMode • startup mode

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_SetDlgSrvStartupMode( )** function sets Intel® Dialogic System Service startup mode. For more information about the Intel® Dialogic System Service, see Section 2.8, "Starting and Stopping the Intel® Dialogic System Service", on page 21.

The function parameter is defined as follows:

Parameter	Description
ncmStartupMode	new startup mode setting for the Intel® Dialogic System Service; the values for this parameter are: • **NCM_DLGSRV_AUTO**: service starts automatically at system startup time • **NCM_DLGSRV_MANUAL**: service must be manually started • **NCM_DLGSRV_DISABLED**: disable the service

■ Cautions

If you use the **NCM_SetDlgSrvStartupMode ( )** function to set the Intel® Dialogic System Service startup mode to **Automatic**, the system does not auto-detect devices before starting the Intel® Dialogic System Service. Therefore, the following restrictions apply when the system is rebooted in **Automatic** mode:

- You cannot add or remove SpringWare devices.

- You cannot add or remove DM3 PCI devices. However, you can replace a DM3 PCI device with an identical device. For example, you can physically remove a DM/V960-4T1-PCI from the system, but you must install another DM/V960-4T1-PCI in the same PCI slot as the original device.
- You can add or remove DM3 CompactPCI (cPCI) devices while the system is powered on. However, you must first use the **NCM_StopDlgSrv( )** function to stop the Intel® Dialogic System Service and then use the **NCM_DetectBoardsEx( )** function before rebooting the system. Alternatively, you can use the **NCM_StopBoard( )** and **NCM_StartBoard( )** functions to perform a Hot Swap of the DM3 cPCI device without stopping the Intel® Dialogic System Service or telephony application. Refer to Section 2.9, "Single Board Stop/Start (Hot Swap)", on page 23 for more information about Hot Swap.

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service

NCME_CHANGE_SVC_STATUS	an error occurred changing the Intel® Dialogic System Service status
NCME_UNKNOWN_SERVICE_TYPE	the service type is unknown

■ See Also

- **NCM_GetDlgSrvStartupMode( )**
- **NCM_GetDlgSrvState( )**
- **NCM_StartDlgSrv( )**
- **NCM_StopDlgSrv( )**

Name: NCM_RetCode NCM_SetTDMBusValue(pncmBus, pvariable, pvalue)

Inputs:

NCMDevice *pncmBus	• pointer to the specific bus name
NCMVariable *pvariable	• pointer to a structure containing the variable name
NCMValue *pvalue	• pointer to value to set

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_SetTDMBusValue( )** function sets the values of the TDM bus. Variables under the TDM Bus family with "User Defined" in the parameter name can be changed by the user. Those with "Resolved" cannot.

For additional information about using this function, see Section 2.10, "Clock Master Fallback List", on page 24.

The function parameters are defined as follows:

Parameter	Description
pncmBus	pointer to a structure containing a specific bus name, for example, "Bus-0"
pvariable	pointer to a structure containing the name of the variable
pvalue	pointer to a 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 NCME_ACCESS_DENIED.

The variable must be a valid parameter under the TDM Bus configuration; otherwise, the function returns NCME_INVALID_INPUTS.

Currently, only a single bus is supported. Therefore, the bus name for the parameter **pncmBus** should be “Bus-0”.

■ 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
     ...
}

...
```

■ Error Codes

Equate	Returned When
NCME_MEM_ALLOC	memory could not be allocated to perform the function
NCME_GENERAL	a problem occurred retrieving the data
NCME_CTBB_LIB	a failure to load the CTBB library occurred
NCME_CTBB_ USERAPPLY	error updating the TDM Bus parameters
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_ACCESS_DENIED	variable has read-only access or is unmodifiable

■ See Also

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

Name: NCMRetCode NCM_SetValue(pncmFamily, pncmDeviceUnique, pncmProperty, pncmVariable, pncmValue)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a device family
NCMDevice *pncmDeviceUnique	• pointer to a structure containing unique device name
NCMProperty *pncmProperty	• pointer to a structure containing a property section
NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
NCMValue *pncmValue	• pointer to a structure containing the new value

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_SetValue( )** function sets a configuration parameter value.

For more information about using this function, see Section 2.3.3, “Reading and Writing Configuration Parameter Values”, on page 13.

The function parameters are defined as follows:

Parameter	Description

pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below.)
pncmDeviceUnique	pointer to the 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
pncmProperty	pointer to the structure containing the property name; the value contained in the structure must be an instantiated property (see Cautions below)
pncmVariable	pointer to the structure containing the configuration parameter name; the value contained in the structure must be an instantiated configuration parameter (see Cautions below)
pncmValue	pointer to the value to be set

■ Cautions

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. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, "The DCM API Architecture", on page 5.

The DCM 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( )** function.

■ 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
     ...
}

...
```

■ Error Codes

Equate	Returned When
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid
NCME_BAD_DATA_TYPE	the data type of the variable is incorrect or indeterminable
NCME_CTBB_DEVICE_ DETECTED	error configuring the TDM bus
NCME_SP	invalid state transition
NCME_BAD_DATA_LOC	the data destination is invalid or indeterminate

See Also

- **NCM_AddDevice( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_SetValueEx( )**

Name: NCMRetCode NCM_SetValueEx(pncmFamily, pncmDeviceUnique, pncmVariable, pncmValueEx)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a device family
NCMDevice *pncmDeviceUnique	• pointer to a structure containing unique device name
NCMVariable *pncmVariable	• pointer to a structure containing a configuration parameter
NCMValueEx *pncmValueEx	• pointer to a structure containing the new value

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

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

For more information about using this function, see Section 2.3.3, “Reading and Writing Configuration Parameter Values”, on page 13.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below)

Parameter	Description
pncmDeviceUnique	pointer to the 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
pncmVariable	pointer to the structure containing the configuration parameter name; the value contained in the structure must be an instantiated configuration parameter (see Cautions below)
pncmValueEx	a pointer to the value to be set

■ Cautions

This function enables you to set the value of a configuration parameter in the system configuration. It does not enable you to add configuration parameters to the DCM Catalog. For more information about the distinction between the system configuration and the DCM Catalog, see Section 2.1, "The DCM API Architecture", on page 5.

The DCM 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( )** function.

■ 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
     ...
}
...
```

■ Error Codes

Equate	Returned When
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_SP	invalid state transition
NCME_INVALID_INPUTS	the values of the parameters supplied are invalid

■ See Also

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

- **NCM_GetValue( )**
- **NCM_GetValueEx( )**
- **NCM_SetValue( )**

Name: NCMRetCode NCM_StartBoard(pncmFamily, pncmDeviceUnique)

Inputs:	NCMFamily *pncmFamily	• pointer to a structure containing a device 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	
Mode:	synchronous	

■ Description

The **NCM_StartBoard( )** function starts a single Intel® Dialogic DM3 board. This function only applies to DM3 PCI (H.100) and CompactPCI (H.110) boards. The function blocks until the board is completely started. For more information about starting a single Intel® Dialogic DM3 Board, see Section 2.9, "Single Board Stop/Start (Hot Swap)", on page 23.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below)
pncmDeviceUnique	pointer to the 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

■ Cautions

This function is applicable only to devices that are capable of being started and stopped on an individual basis (i.e., DM3 PCI and CompactPCI boards).

■ 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
        ...
}
```

■ Error Codes

Equate	Returned When
NCME_GENERAL	a problem occurred retrieving the data

■ See Also

- **NCM_StopBoard( )**

Name: NCMRetCode NCM_StartDlgSrv( )

Inputs: None

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_StartDlgSrv( )** function initiates the Intel® Dialogic System Service. For more information about the Intel® Dialogic System Service, see Section 2.8, "Starting and Stopping the Intel® Dialogic System Service", on page 21.

■ Cautions

All auto-detectable Intel® Dialogic devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( )** function before the **NCM_StartDlgSrv( )** function can be used to start the Intel® Dialogic System Service. Refer to Section 2.6.1, "Auto Detection", on page 18 for information about auto-detectable Intel® Dialogic devices.

A successful completion code for this function only indicates that the Intel® Dialogic System Service was initiated. To determine whether the Intel® Dialogic System Service completed its startup successfully, use the **NCM_GetDlgSrvState( )** function.

■ Example

```
#include "NCMApi.h"

...

//
// Execute
//
```

```
NCMRetCode     ncmRc = NCM_StartDlgSrv(  );

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

...
```

■ Error Codes

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service
NCME_STARTING_DLGC_SVC	an error occurred starting the Intel® Dialogic System Service

■ See Also

- **NCM_GetDlgSrvStartupMode( )**
- **NCM_GetDlgSrvState( )**
- **NCM_SetDlgSrvStartupMode( )**
- **NCM_StopDlgSrv( )**

Name: NCMRetCode NCM_StopBoard (pncmFamily, pncmDeviceUnique)

Inputs:

NCMFamily *pncmFamily	• pointer to a structure containing a device family name
NCMDevice *pncmDeviceUnique	• pointer to a structure containing a unique device

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: synchronous

■ Description

The **NCM_StopBoard( )** function stops a single Intel® Dialogic DM3 board. This function only applies to DM3 PCI (H.100) and CompactPCI (H.110) boards. The function blocks until the board is completely stopped. For more information about stopping a single Intel® Dialogic DM3 Board, see Section 2.9, "Single Board Stop/Start (Hot Swap)", on page 23.

The function parameters are defined as follows:

Parameter	Description
pncmFamily	pointer to the structure containing the family name; the value contained in the structure must be an instantiated family (see Cautions below)
pncmDeviceUnique	pointer to the 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

■ Cautions

This function is applicable only to devices that are capable of being started and stopped on an individual basis (i.e., DM3 PCI and CompactPCI boards).

■ 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
        ...
}
```

■ Error Codes

Equate	Returned When
NCME_GENERAL	a problem occurred retrieving the data

■ See Also

- **NCM_StartBoard( )**

Name: NCMRetCode NCM_StopDlgSrv( )

Inputs: None

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h

Mode: Synchronous

■ Description

The **NCM_StopDlgSrv( )** function stops the Intel® Dialogic System Service. For more information about the Intel® Dialogic System Service, see Section 2.8, "Starting and Stopping the Intel® Dialogic System Service", on page 21.

■ Cautions

A successful completion code indicates that this function attempted to stop the Intel® Dialogic System Service. To determine whether the Intel® Dialogic System Service completed its stop, use the **NCM_GetDlgSrvState( )** function.

■ Example

```
#include "NCMApi.h"

...

//
// Execute
//

NCMRetCode     ncmRc = NCM_StopDlgSrv(  );

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

■ Error Codes

Equate	Returned When
NCME_OPENING_SCM	an error occurred opening service control manager
NCME_OPENING_DLGC_SVC	an error occurred opening the Intel® Dialogic System Service
NCME_STOPPING_DLGC_SVC	an error occurred stopping the Intel® Dialogic System Service

■ See Also

- **NCM_GetDlgSrvStartupMode( )**
- **NCM_GetDlgSrvState( )**
- **NCM_SetDlgSrvStartupMode( )**
- **NCM_StartDlgSrv( )**

5. Dialogic OEM Installation Tool (DOIT)

5.1. Overview

The Intel® Dialogic OEM Installation Tool (DOIT) enables you to create customized installation programs for system software. To do so, you include the contents of the Intel® Dialogic CD on your own distribution media and then pass command-line parameters to the Intel® Dialogic *Setup.exe* program through a batch file.

You will need to observe the following guidelines when using DOIT:

- Your OEM CD must contain the entire contents of the Intel® Dialogic CD, both the contents of the root directory and the *\Componnt* subdirectory.
- The DOIT command line input is limited to 64 characters.

5.2. Command-Line Usage

The DOIT options are implemented by means of command-line parameters input to *Setup.exe*, which is located in the root directory of the Intel® Dialogic CD.

The usage of *Setup.exe* is as follows:

```
SETUP DIR(destination directory)DIR COMP(Component
Tokens)COMP Switches
```

- The **DIR** keyword (optional) lets you specify a destination directory for the installation other than the default of *C:\Program Files\Dialogic*. The specified directory will be created if it doesn't exist. If it does exist, any existing files with the same names as the files being installed will be overwritten.
- The **COMP** keyword lets you specify what to install. Component tokens are described in Table 2, "DOIT Component Tokens", on page 196, and in the *DOITReadMe.txt* file, which is located in the *\Componnt* subdirectory of the CD.

NOTE: The PROGDRV component token, which installs drivers, firmware, and configuration files, must **always** be specified.

- The switches specify how the install procedure is carried out. Switches are described in Table 3, "DOIT Switches", on page 199.
- There must be no spaces between each **DIR** keyword and the parenthesis or between each **COMP** keyword and the parenthesis.
- Component tokens are separated from each other by spaces.
- Switches are separated from the **COMP** keyword by a space, and from each other by a space.
- Without the -s switch, the script will not prompt the user for any information but will display the installation progress graphically. Using the -s switch will cause the script to run without any screen output.
- Command line switches after the -s switch are ignored; if it is used, the -s switch must appear last.

The DOIT command line is limited to 64 characters, to overcome this limitation, the Instruction File parameter may be used with *Setup.exe* to reference command line information contained in another file. The Instruction File uses the following format:

```
SETUP I_F (file) I_F
```

where (*file*) identifies an absolute path and filename containing valid command line instructions for the Intel® Dialogic OEM installation script.

NOTE: Valid command line instructions are listed in Table 2, "DOIT Component Tokens", on page 196 and Table 3, "DOIT Switches", on page 199, except for the -s switch. The optional -s switch can only be used with the SETUP I_F (*file*) I_F command line and not in the referenced file.

Table 2. DOIT Component Tokens

Component Token	Component Installed
ANTARES	Drivers for Antares products
BOARDWATCH BWNODEV41	BoardWatch with Compaq Insight Manager V4.1

Table 2. DOIT Component Tokens

BOARDWATCH BWNODEV42	BoardWatch with Compaq Insight Manager V4.2
BRI	Basic Rate Interface (BRI) support
DEVLIB	Development library source (*.C), header (*.H), and library (*.LIB) files
DM3 CTSWITCH	Firmware and configuration files for Dialogic Integrated Products
DM3FAX	Firmware and configuration files for DM3 Fax
DM3 FARALLON	Firmware and configuration files for DM3 IPLink analog
GDK	Dialogic CP Fax products
GLOBALCALL GCSS7	GlobalCall SS7 call control libraries including ISDN, ICAPI, and ANAPI; GlobalCall Protocol PDK runtime component (PDKRT)
DM3 HDSI	Firmware and configuration files for MSI/12000
DM3 IPDIGITAL	Firmware and configuration files for DM3 IPLink

Table 2. DOIT Component Tokens

ISDN 1TR6 ISDN 4ESS ISDN 5ESS ISDN CTR4 ISDN DASS2 ISDN DMS ISDN ETN ISDN ETU ISDN NE1 ISDN NI2 ISDN NT1 ISDN NTT ISDN QNT ISDN QTE ISDN QTN ISDN QTU ISDN TPH ISDN TPHNT ISDN VN ISDN VNNT	ISDN protocols
ONLDOC DOCLOCAL	Online documentation copied to hard drive
ONLDOC DOCREMOTE	Online documentation accessed from CD
PERFMON	Performance counters for the Windows Performance Monitor
PROGDRV	Runtime components necessary for running Dialogic boards (drivers, firmware, and configuration files), Dialogic Configuration Manager (DCM) utility, DCM API, Country Specific Parameter files and Universal Dialogic Diagnostics (UDD) The PROGDRV component token must **always** be specified.
DM3 QVS	Firmware and configuration files for DM3 QuadSpan

Table 2. DOIT Component Tokens

SCX	SCxbus Adapter support
SAMPLES	Sample programs for testing the Dialogic platform
SP	Continuous Speech Processing
TAPI	TAPI service provider

Table 3. DOIT Switches

Switch	Description
C1(directory)C1	Copy a directory tree from the indicated location into the Dialogic installation directory
REBOOT	Reboot after installation if any locked files (e.g., DLLs) were found during copying
REBOOT_ALWAYS	Reboot after installation in all cases
SIGNAL_SUCCESS	Create a file *dlgcinst.fin* in the windows directory at the end of a successful installation
-s	Install "silently" (i.e., without any screen output) **Note:** If it is used, the -s switch must appear last. If it is used with the Instruction File parameter, it must appear with the SETUP I_F (*file*) I_F command.

5.3. Examples

This section provides examples of the DOIT component tokens and switches.

```
SETUP COMP(PROGDRV SAMPLES ONLDOC DOCLOCAL)COMP REBOOT -s
```

This command performs an installation where:

- The runtime components (PROGDRV) are installed.
- All sample programs (SAMPLES) are installed.
- Online documentation is installed locally (ONLDOC DOCLOCAL).
- The setup program will reboot (REBOOT) if any locked files are encountered during file copying.
- The setup program executes silently (-s).

Figure 3. DOIT Example 1

The following steps use an Instruction File to produce the same results as Figure 3, "DOIT Example 1", on page 200:

1. Create a text file: 'a:\somepath\somefile.txt'.
2. Enter the following into the file:

   [Dialogic Installation Parameters]
   Command Line=`COMP(PROGDRV SAMPLES ONLDOC DOCLOCAL) COMP REBOOT`
3. Call the Setup script as follows:

SETUP I_F (a:\somepath\somefile.txt) I_F -s

```
SETUP DIR(c:\somepath)DIR COMP(PROGDRV SAMPLES ONLDOC
DOCLOCAL)COMP REBOOT_ALWAYS -s
```

This command performs an installation where:

- The files are installed in the directory *c:\somepath* (DIR(c:\somepath)DIR) instead of the default installation directory.
- The runtime components (PROGDRV) are installed.
- All sample programs (SAMPLES) are installed.
- Online documentation is installed locally (ONLDOC DOCLOCAL).
- The setup program reboots at the end of the installation in all cases (REBOOT_ALWAYS).
- The setup program executes silently (-s).

Figure 4. DOIT Example 2

```
SETUP COMP(PROGDRV SAMPLES ONLDOC DOCLOCAL)COMP
C1(a:\somedir)C1  REBOOT_ALWAYS -s
```

This command performs an installation where:

- The runtime components (PROGDRV) are installed.
- All sample programs (SAMPLES) are installed.
- Online documentation is installed locally (ONLDOC DOCLOCAL).
- Immediately after file copying and before component registration with the Windows registry, the script will copy the directory tree beginning at *a:\somedir* to the Dialogic installation directory (C1(a:\somedir)C1).
- The setup program reboots at the end of the installation in all cases (REBOOT_ALWAYS).
- The setup program executes silently (-s).

Figure 5. DOIT Example 3