Native Configuration Manager API for Windows Operating Systems

Programming Guide

April 2005

05-1904-003

Publication Date: April 2005

Document Number: 05-1904-003

Intel Converged Communications, Inc.
1515 Route 10
Parsippany, NJ 07054

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

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

For **Sales Offices** and other contact information, visit the Where to Buy Intel Telecom Products page at:
http://www.intel.com/buy/wtb/wtb1028.htm

Contents

Figures

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1904-003	April 2005	Features section: Removed information about support for third party devices.
05-1904-002	November 2003	Features section: Reclassified the NCM API features. Restrictions and Limitations section: Added note recommending that users not parse the device name. Added note that detection is always run automatically via the Plug and Play Observer service that is set to run automatically on system start. NCM API Architecturefigure: Replaced with redrawn version. Programming Models chapter: Added paragraph about exceptions to the synchronous mode of operation. Event Handling chapter: Added paragraph about the NCM AUID functions. Configuration Parameter Property Groups section: Added mention of two more groups (Misc and TDM Bus Configuration). Device Model Names and Unique Device Names section: Added NCM_ApplyTrunkConfig and NCM_ReconfigBoard to the list. Added information about the automatic creation of device names.
05-1904-001	November 2002	Initial version of document. Much of the information in this document was previously published in the *Customization Tools for Installation and Configuration for Windows*, document number 05-1103-007.

About This Publication

The following topics provide information about this *Native Configuration Manager API for Windows Operating Systems Programming Guide*:

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

Purpose

This publication provides guidelines for building customized Intel® Dialogic® system management applications using the Native Configuration Manager (NCM) API on Windows* operating systems. Such applications include, but are not limited to, TDM bus clock fallback management, device configuration, and single board stop/start programs.

This publication is a companion guide to the *Native Configuration Manager API for Windows Operating Systems Library Reference*, which provides details on the functions and parameters in the NCM library.

Intended Audience

This information is intended for:

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

How to Use This Publication

This document assumes that you are familiar with and have prior experience with Windows operating systems and the C programming language. Use this document in tandem with the *Native Configuration Manager API for Windows Operating Systems Library Reference*.

This publication is organized as follows:

- Chapter 1, "Product Description" introduces the key features of the NCM library and provides a brief description of each feature.
- Chapter 2, "Programming Models" provides a brief overview of supported programming models.
- Chapter 3, "Event Handling" provides information about handling events that are generated by certain NCM library functions.
- Chapter 4, "Error Handling" includes information about handling errors within your application.
- Chapter 5, "Application Development Guidelines" provides programming guidelines and techniques for developing an application using the NCM library.
- Chapter 6, "Clock Master Fallback List" includes guidelines for using the NCM library to set a customized CT Bus clock master fallback list.
- Chapter 7, "Building Applications" discusses compiling and linking requirements such as include and library files.

Related Information

Refer to the following publications and websites for more information:

- *Native Configuration Manager API for Windows Operating Systems Library Reference*
- The appropriate Configuration Guide(s):
  - *Intel DM3 Architecture PCI Products on Windows Configuration Guide*
  - *Intel Dialogic on Springware Architecture Products for PCI on Windows Configuration Guide*
  - *Intel NetStructure Products on DM3 Architecture for CompactPCI on Windows Configuration Guide*
- *Event Service API for Windows Operating Systems Library Reference*
- *Event Service API for Windows Operating Systems Programming Guide*
- The appropriate Administration Guide:
  - *Intel Dialogic System Software for PCI Products on Windows Administration Guide*
  - *Intel Dialogic System Release CompactPCI for Windows Administration Guide*
- The Release Guide for your specific Intel Dialogic software version
- The online Release Update for your specific Intel Dialogic software version
- *http://developer.intel.com/design/telecom/support/* for technical support
- *http://www.intel.com/network/csp/* for product information

Product Description 1

This chapter provides a general description of the Native Configuration Manager (NCM) library. Topics include:

1.1 Features

The NCM API consists of a library of functions for creating and manipulating the configuration data necessary to initialize Intel® telecom devices and control their operation on Windows* systems. The features available through the NCM API can be grouped as follows:

- **Configuration: get and set parameters**
  - **Board/Bus** - determine which devices can be installed and which configuration parameter values can be set for a given Intel® Dialogic® system software release; set configuration parameter values for the boards in your system.
  - **Clock Master Fallback list** - Get/set the TDM bus clock master fallback list for your system. 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.
- **Initialization/Unintialization**
  - **Start/stop system** - Start/stop the Intel Dialogic System and check the system status
  - **Stop/start board** - Stop and start individual boards in the system without affecting system operation. Refer to the System Administration Guide that accompanied your system software release for complete information about stopping and starting Intel telecom boards.
  - **System startup mode** - You can set the startup mode to automatic or manual.
- **Miscellaneous**
  - **Get AUID** - Get the device name for a given AUID and/or get the AUID for a given device name. An AUID is a unique string of numbers that the Intel Dialogic System Software assigns to any system component with which communications can be initiated. In the context of the NCM API, each unique device that is instantiated in the system is assigned an AUID.
  - **Country specific format** - Obtain information about country-specific parameters such as a list of supported countries, a country code or name for a given country, and values for country-specific configuration parameters. This only applies to Springware devices.
  - **Release and operating system information** - Determine which operating system and Intel Dialogic System Software versions are installed on the host computer.

1.2 Restrictions and Limitations

The following restrictions and limitations apply to the NCM API:

- You can add and modify configuration data only for those Intel telecom hardware products supported by the software release of which the NCM API is a component. Refer to the *Release Guide* that accompanies each system software release for a list of supported products.
- 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 strongly recommends that you embed the device model name within the unique device name when you instantiate the device with the **NCM_AddDevice( )** function.

  Note: You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.
- All auto-detectable devices in the system must be detected using either the **NCM_DetectBoards( )** function or the **NCM_DetectBoardsEx( ) f**unction before **NCM_StartDlgSrv( )** can be called to start the Intel Dialogic System Service.

  Note: Detection is always run automatically via the Plug and Play Observer service that is set to run automatically on system start.
- When using cPCI boards, you can add or remove boards 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.

1.3 NCM API Architecture

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

- **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 NCM API is a component. The NCM 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.
- **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 telecom devices and configuration parameter values you can instantiate are determined by the DCM catalog. The NCM 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 Intel Dialogic System Software through the NCM API when it initializes Intel telecom devices.

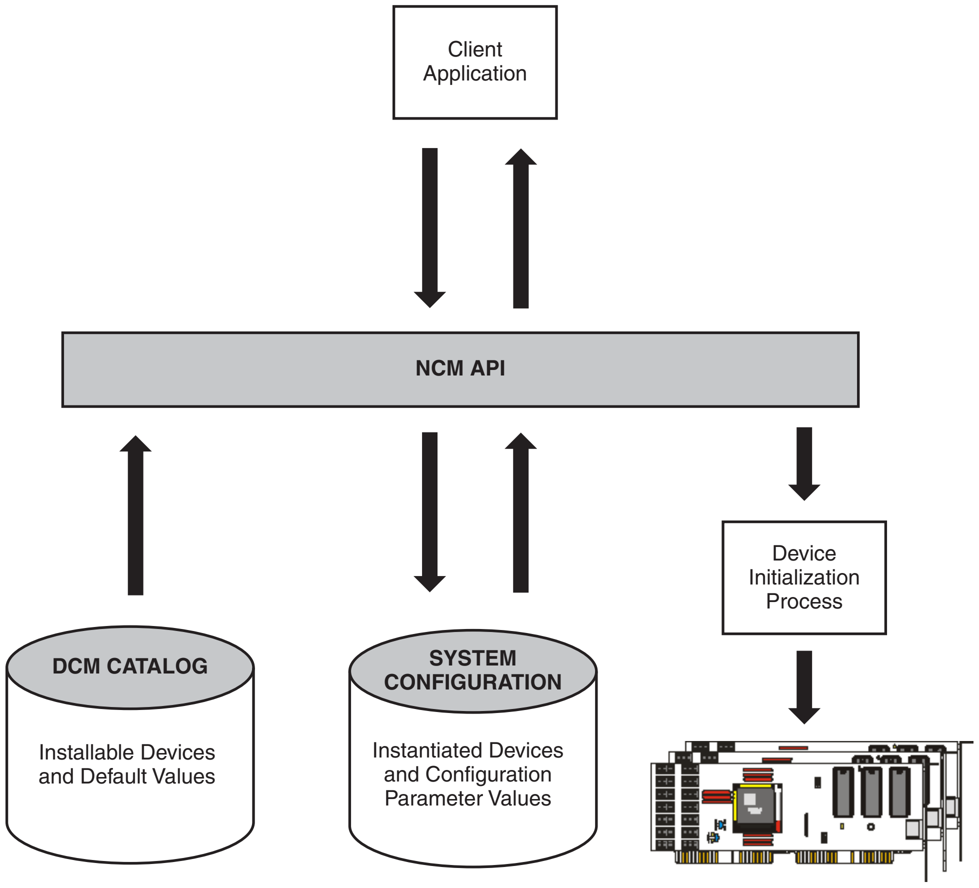
Note: In the current version of the NCM API, the system configuration is stored in the Windows registry. Do not attempt to modify the NCM registry data through any means other than the NCM API. By adhering to this guideline, you ensure forward-compatibility between your application and future versions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

- **NCM API:** The NCM API consists of a library of functions for instantiating Intel telecom devices in the system configuration and for modifying configuration parameter values. It also enables you to start and stop the Intel Dialogic System, stop and start individual boards, and to auto-detect devices.
- **Client Application:** A client application makes the functionality of the NCM 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 configuration utility GUI client, which is included in every Intel Dialogic System Software release that includes the NCM API. Access the DCM GUI client by clicking the **Configuration Manager-DCM** icon from the **Intel Dialogic System Release** submenu in the Windows Start menu.

Figure 1 shows the various components that interface with the NCM API architecture:

Figure 1. NCM API Architecture


Client Application
NCM API
DCM CATALOG
Installable Devices and Default Values
SYSTEM CONFIGURATION
Instantiated Devices and Configuration Parameter Values
Device Initialization Process

Programming Models 2

This chapter provides information about supported programming models.

All functions in the NCM API run exclusively in synchronous mode. The synchronous programming model uses functions that block application execution until the function completes. This model requires that each device be controlled from a separate process. This allows you to assign distinct applications to different devices dynamically in real time.

Synchronous programming models allow you to scale an application by simply instantiating more threads or processes. This programming model may be easy to encode and manage but it relies on the system to manage scalability.

The only exceptions to the synchronous mode of operation are the **NCM_StartDlgSrv** and **NCM_StopDlgSrv** APIs. These APIs send a start/stop command to the Intel® Dialogic® System, but do not wait until the system has started/stopped. The application should check the system state in a loop (via **NCM_GetDlgSrvStateEx**) to confirm that start/stop has finished execution.

Event Handling 3

This chapter provides information on handling events that are generated by certain NCM library functions (other than the exceptions noted in Chapter 2, “Programming Models”).

All functions in the NCM API operate in synchronous mode, so the start/completion of each function is not determined by events. However, certain functions generate events that are transmitted via the Intel® Event Notification framework’s ADMIN_CHANNEL.

You can design your application to receive events from the ADMIN_CHANNEL. This allows your system management application to be informed when an individual board has started/stopped and when the Intel® System has started/stopped.

The following NCM library functions generate events that are carried on the event notification framework’s ADMIN_CHANNEL:

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

The events reported by Event Service contain the AUID to identify a device. The NCM API normally uses the family and device to identify a device. The NCM AUID functions (**NCM_GetAUID** and **NCM_GetFamilyDeviceByAUID**) provide a mapping from family/device to AUID and vice-versa, thus allowing the NCM API to be used in conjunction with Event Service to control the application.

Refer to the *Event Service API for Windows Operating Systems Library Reference* and the *Event Service API for Windows Operating Systems Programming Guide* for information about registering your application to receive events on the ADMIN_CHANNEL.

Error Handling 4

This chapter discusses how to handle errors that can occur when running an application.

All NCM library functions have a return code (**NCMRetCode**) to indicate success or failure of the function. A return code of NCM_SUCCESS indicates that the function completed successfully. Any other return value indicates failure.

If an NCM library function fails, call the **NCM_GetErrorMsg( )** function to process the returned error code. The following sample code shows how to implement error handling into NCM API function calls:

```
#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);
...
```

For a list of error codes that can be returned by the NCM library functions, see the *Native Configuration Manager API for Windows Operating Systems Library Reference*. You can also look up the error codes in the *NCMTypes.h* file.

Application Development Guidelines 5

This chapter contains guidelines for developing applications with the NCM API. Topics include:

5.1 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® telecom board in your system configuration. When the Intel® Dialogic® System Service is initiated, it reads the configuration parameter values through the NCM 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 Release** submenu in the Windows* Start menu. Access the online help by pressing the **F1** key at any window.

A device is an Intel telecom board, such as a D/240JCT-T1. The NCM 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.

5.2 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:

- **AUID**
- **DlgcOUI**

- **InstanceNumber**
- **IntVector**
- **IRQLevel**
- **LogicalID**
- **PciBusNumber**
- **PciID**
- **PciSlotNumber**
- **PLXAddr**
- **PLXlength**
- **PrimaryBoardID**
- **SecondaryBoardID**
- **SerialNumber**
- **SRAMAddr**
- **SRAMlength**
- **SRAMSize**

There are other groups like “Misc” and “TDM Bus Configuration”. The parameters in any given property group vary depending on the Intel System Software release with which the NCM API is provided.

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

5.3 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:

Note: Refer to Section 5.3.1, “Determining Configuration Parameter Scope”, on page 21 for information about determining a parameter’s 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.
- **Family-Level:** a family-level configuration parameter applies to a family of devices (boards) in the system, such as the DM3 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, when you set the value of a global parameter, the value applies to all boards to which the parameter applies.

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 NCM API.

5.3.1 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.

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 Release** submenu 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.

5.3.2 Reading and Writing Configuration Parameter Values

Functions such as **NCM_GetValue( )**, **NCM_GetValueEx( )**, **NCM_SetValue( )** and **NCM_SetValueEx( )** 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 data structures as input, and it is through these structures that you specify the parameter's scope.

Set the data 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.

  Note: 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.

5.4 Populating Required Data Structures

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

The following steps illustrate how to populate these data structures:

1. Use **NCM_GetAllFamilies( )** to return a list of NCMFamily data 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 data 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 data structures for a specific device as input to **NCM_GetProperties( )**; this function returns a list of NCMProperty data structures for the device identified by the NCMFamily and NCMDevice data structures. Lists of properties can be retrieved only for one device at time.
4. Use the NCMProperty, NCMFamily, and NCMDevice data structures as input to **NCM_GetVariables( )**; this function returns a list of NCMVariable data structures for the installable device indicated by the specified input structures. The list of NCMVariable data structures contains all the installable configuration parameters within the property for the specified device.

5.5 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 name for a device, such as "D/240JCT". The device model name is necessary to retrieve a list of installable devices with the **NCM_GetAllDevices( )** function.

Note: You are strongly discouraged from parsing the unique device name from your application. Although the name is guaranteed to be unique, Intel reserves the right to change the format of the device name in future releases.

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.

NCM 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_ApplyTrunkConfig( )**
  - **NCM_DeleteEntry( )**
  - **NCM_EnableBoard( )**
  - **NCM_IsBoardEnabled( )**
  - **NCM_ReconfigBoard( )**
  - **NCM_SetValue( )**
  - **NCM_SetValueEx( )**
  - **NCM_StartBoard( )**
  - **NCM_StopBoard( )**

  When **NCM_DetectBoards** is run, unique device names are automatically created by the NCM framework. You can also create a 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.
- **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.

The NCM API function descriptions in the *Native Configuration Manager API for Windows Operating Systems Library 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.

5.6 Dynamic Memory Allocation

Many NCM 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 the *Native Configuration Manager API for Windows Operating Systems Library Reference*.

5.7 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 5.3.2, “Reading and Writing Configuration Parameter Values”, on page 21. 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. In the case of country-specific configuration parameters, 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.

Clock Master Fallback List 6

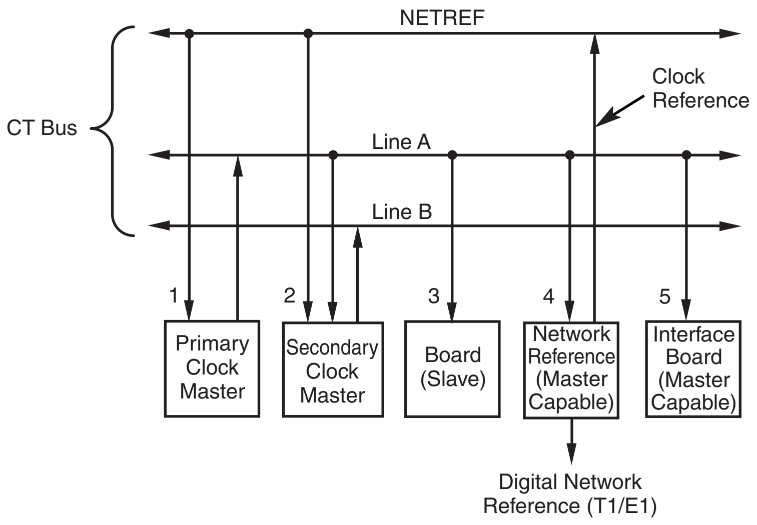
This chapter provides guidelines for using the NCM API to define a TDM bus clock master fallback list.

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.

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 shows the CT Bus clocking scheme:

Figure 2. CT Bus Clocking


NETREF
Clock Reference
CT Bus
Line A
Line B
1
2
3
4
5
Primary Clock Master
Secondary Clock Master
Board (Slave)
Network Reference (Master Capable)
Interface Board (Master Capable)
Digital Network Reference (T1/E1)

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 farther down the list.

Note: 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.

Building Applications 7

This chapter provides information about building applications using the NCM library. The following topics are discussed:

7.1 Compiling and Linking

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries
- Variables for Compiling and Linking

7.1.1 Include Files

Function prototypes and equates are defined in include files, also known as header files. Applications that use NCM library functions must contain statements for include files in this form, where *filename* represents the include file name:

```
#include <filename.h>
```

The following header files must be included in application code prior to calling NCM library functions:

NCMApi.h
Contains function prototypes for the NCM library. Used for all NCM library application development.

devmap.h
Contains equates used to assign AUIDs to boards. Used only if you are invoking the **NCM_GetAUID( )** or **NCM_GetFamilyDeviceByAUID( )** functions in your application.

Note: By default, the header files are located at <*install drive*>*:*\<*install directory*>*\dialogic\inc*.

7.1.2 Required Libraries

You must link the following library file when compiling your NCM API application:

NCMApi.lib
Main NCM library function

Note: By default, the library files are located at <*install drive*>*:*\<*install directory*>*\dialogic\lib.*

7.1.3 Variables for Compiling and Linking

In System Release 6.0, the following variables have been introduced to provide a standardized way of referencing the directories that contain header files and shared objects:

INTEL_DIALOGIC_INC
: Variable that points to the directory where header files are stored.

INTEL_DIALOGIC_LIB
: Variable that points to the directory where shared library files are stored.

These variables are automatically set at login and should be used in compiling and linking commands. The following is an example of a compiling and linking command that uses these variables:

```
cc -I${INTEL_DIALOGIC_INC} -o myapp myapp.c -L${INTEL_DIALOGIC_LIB} -lgc
```

Note: It is strongly recommended that developers begin using these variables when compiling and linking applications since they will be required in future releases. The name of the variables will remain constant, but the values may change in future releases.

Index