# OA&M API for Linux Operating Systems

## Library Reference

*August 2005*

05-1841-004



Publication Date: August 2005

Document Number: 05-1841-004

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

# Revision History

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

| Document No. | Publication Date | Description of Revisions |
|---|---|---|
| 05-1841-004 | August 2005 | AGENTTDMCONFIGURATION data structure reference: Noted that the SCbus value is not supported.<br>AGENTTDMCONFIGURATION data structure reference: Added Unknown to the ClockingModel data structure field.<br>ADMIN_CHANNEL Events section: Added ADMIN_CHANNEL Event Payloads table |
| 05-1841-003 | March 2005 | Events chapter: DLGC_EVT_BLADE_ABOUT_TO_REMOVE event added. This event is generated by cPCI boards only. The current release supports PCI and cPCI boards.<br>DLGC_EVT_BLADE_DETECTED event added. This event is generated by cPCI boards only.<br>DLGC_EVT_BLADE_BLADE_REMOVED event added. This event is generated by cPCI boards only. This event is generated by cPCI boards only.<br>DevAdminEventMsg data structure reference: Updated data structure to reflect current release<br>DlgEventMsgType data structure reference: Updated data structure to reflect current release |

| Document No. | Publication Date | Description of Revisions |
|---|---|---|
| 05-1841-002 | November 2004 | Global change: changed all function modes (except for **StartListening( )** and **HandleEvent( )**) from asynchronous to synchronous.<br>ICTNode::GetSystemReleaseVersionInfo( ) function reference: Noted that any application using this function as part of the current release must be recompiled. This required recompilation is due to a change in the CTPLATFORMVERSIONINFO data structure.<br>Events: DLGC_EVT_NETREF2_LINEBAD event has been removed.<br>DLGC_EVT_BLADE_ABOUT_TO_REMOVE event removed. This event is generated by cPCI boards only. The current release does not support cPCI boards.<br>DLGC_EVT_BLADE_DETECTED event removed. This event is generated by cPCI boards only. The current release does not support cPCI boards.<br>DLGC_EVT_BLADE_BLADE_REMOVED event removed. This event is generated by cPCI boards only. The current release does not support cPCI boards.<br>DLGC_EVT_SYNC_MASTER_CLOCK event removed. This event is not supported.<br>DLGC_EVT_EXTERNAL_CARRIER_DETECT event removed. This event is not supported.<br>DLGC_EVT_BLADE_ABOUT_TO_START event added.<br>DLGC_EVT_BLADE_ABOUT_TO_STOP event added.<br>Noted that Springware architecture boards generate NETWORK_ALARM_CHANNEL events only. DM3 architecture boards generate all events.<br>AGENTTDMCONFIGURATION data structure reference: Noted that the MVIP bus value is not supported.<br>CTPLATFORMVERSIONINFO data structure reference: Added new data fields (lServiceUpdateNumber, lServiceUpdateBuildNumber, lFeatureReleaseNumber, lFeatureReleaseBuildNumber) to the data structure. Any application using this data structure as part of the current release must be recompiled to include the new data fields. |
| 05-1841-001 | September 2002 | Initial version of document. |

# About This Publication

The following topics provide information about this publication:



## Purpose

This publication provides a reference to all classes, functions, data structures, events and error codes in the OA&M (Operations, Administration & Maintenance) library.

This publication is a companion document to the *OA&M API for Linux Programming Guide* which provides guidelines for developing applications, including highly available applications, using the OA&M API.

## 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

Refer to this guide after you have installed the hardware and system software for Linux* which includes the OA&M API.

This guide assumes that you are familiar with the Linux operating system and the C++ programming language. If you are developing a CT Bus management application, you should also have a thorough understanding of CT Bus clocking concepts (Primary, Secondary and Network Reference clock masters).

The information in this publication is organized as follows:

- Chapter 1, "Function Summary by Category" introduces you to the various categories (organized by C++ class membership) of functions in the OA&M library.
- Chapter 2, "Function Information" provides a reference to the OA&M API functions. Functions are listed according to class membership.
- Chapter 3, "Events" provides a reference to the events used by the OA&M event notification framework.
- Chapter 4, "Data Types" includes reference information about the data types that are defined in the OA&M API.
- Chapter 5, "Data Structures" defines the data structures used by the OA&M API.
- Chapter 6, "Error Codes" includes reference information about the exception objects that can be thrown by the OA&M API functions.

## Related Information

See the following for more information:

- *OA&M API for Linux Operating Systems Programming Guide*
- *High Availability for Linux Operating System Demo Guide*
- *Intel® Dialogic® System Software on Linux Operating Systems Administration Guide*
- *Intel® DM3 Architecture PCI Products on Linux Configuration Guide*
- *Intel® Springware Architecture Products on Linux Configuration Guide*
- *Intel® NetStructure® IPT Series on Linux Operating Systems Configuration Guide*
- *System Release Guide*
- *System Release Update*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.ectf.org/* (for CT Bus clocking specifications)

# Function Summary by Category 1

The OA&M API functions are grouped into the following categories:



***Notes:*** ***1.*** Each category corresponds to a C++ class. `#include` statements for the CCTDomain, ICTNode, ICTBoard and ICTClockAgent class header files are found in *dasi.h*, the primary OA&M API header file. The DlgAdminConsumer class is defined in the *dlgadminconsumer.h* file. The CEventHandlerAdaptor class is defined in the *dlgadminmsg.h* file.

***2.*** OA&M API functions that appear in header files but aren't covered in this document are not supported.

## 1.1 CCTDomain Class Functions

CCTDomain member functions are used to instantiate a CCTDomain object as an entry point to the OA&M system and get a pointer to an instance of the ICTNode class (ICTNode object).

The following functions are included in the CCTDomain class:

**CCTDomain::CCTDomain( )**
Instantiates a CCTDomain object.

**CCTDomain::GetNode( )**
Returns a pointer to an ICTNode object.

***Note:*** The CCTDomain class uses the DASI namespace.

## 1.2 ICTNode Class Functions

ICTNode member functions return information about the Intel NetStructure®, DM3 architecture or Springware architecture boards and software that is installed in a node. In the context of the OA&M API, a node is defined as a single host computer that contains Intel® Dialogic® System Software and Intel NetStructure, DM3 architecture or Springware architecture boards.

The ICTNode class contains the following functions:

**ICTNode::GetAllInstalledBoards( )**
Returns a pointer to a Standard Template Library (STL) list of pointers to instances of the ICTBoard class (ICTBoard objects). Each ICTBoard object in the returned list represents a board that is installed in the node.

**ICTNode::GetBoardByID( )**
Gets a pointer to the ICTBoard object that corresponds to a given Addressable Unit Identifier (AUID).

**ICTNode::GetNumberOfInstalledBoards( )**
Retrieves the number of boards that are installed in the node.

**ICTNode::GetSystemReleaseVersionInfo( )**
Returns information about the Intel Dialogic System Software installed on the node.

***Note:*** The ICTNode class uses the DASI namespace.

## 1.3 ICTBoard Class Functions

ICTBoard member functions return information about individual Intel NetStructure®, DM3 architecture or Springware architecture boards that are installed in a node.

The following functions are contained in the ICTBoard class:

**ICTBoard::GetBoardID( )**
Returns the board's AUID.

**ICTBoard::GetClockingAgents( )**
Returns a pointer to an STL list of pointers to instances of the ICTClockAgent class (ICTClockAgent objects). Each ICTClockAgent object in the returned list represents a clocking agent that resides on the board.

**ICTBoard::GetNumClockingAgents( )**
Returns the number of clocking agents on a board.

**ICTBoard::GetNumTrunks( )**
Gets the number of digital network interfaces on a board.

**ICTBoard::GetPCIBus( )**
Gets the PCI bus number a board is installed on.

**ICTBoard::GetPCISlot( )**
Returns the PCI slot number a board is installed in.

**ICTBoard::GetPhysicalSlot( )**
Returns the physical slot number of a cPCI board.

**ICTBoard::GetSerialNumber( )**
Gets the serial number of a board.

**ICTBoard::GetShelfId( )**
Returns the shelf ID of a cPCI board.

***Note:*** The ICTBoard class uses the DASI namespace.

## 1.4 ICTClockAgent Class Functions

ICTClockAgent member functions are used to get the TDM bus capabilities and get or set the TDM bus configuration of a board's clocking agent.

The following functions are contained in the ICTClockAgent class:

**ICTClockAgent::GetClockAgentID( )**
Returns the AUID of a board's clocking agent.

**ICTClockAgent::GetTDMCapabilities( )**
Retrieves the TDM bus capabilities of a board's clocking agent.

**ICTClockAgent::GetTDMConfiguration( )**
Gets the current TDM bus configuration of a board's clocking agent.

**ICTClockAgent::SetTDMConfiguration ( )**
Sets the TDM bus configuration of a board's clocking agent.

***Note:*** The ICTClockAgent class uses the DASI namespace.

## 1.5 DlgAdminConsumer Class Functions

In order to build a completely automated OA&M application (for example, a CT Bus clock daemon), you must instantiate at least one DlgAdminConsumer object to receive asynchronous events from the event notification component of the OA&M library. The DlgAdminConsumer uses the DlgEvent Service namespace. The class member functions are used to instantiate event consumer objects and return information about the objects. Refer to the *OA&M API for Linux Programming Guide* for complete information about consumer objects and event notification.

The DlgAdminConsumer class contains the following functions:

**DlgAdminConsumer::DisableFilters( )**
Disables a consumer object's array of filters.

**DlgAdminConsumer::DlgAdminConsumer( )**
Instantiates a DlgAdminConsumer object.

**DlgAdminConsumer::EnableFilters( )**
Enables a consumer object's array of filters.

**DlgAdminConsumer::getChannelName( )**
Gets the channel name that a consumer object is monitoring for incoming events.

**DlgAdminConsumer::getConsumerName( )**
Returns the name of the consumer object.

**DlgAdminConsumer::StartListening( )**
Allows the consumer object to begin monitoring its associated channel for incoming events.

## 1.6 CEventHandlerAdaptor Class Functions

Applications must implement a class derived from the CEventhandlerAdaptor class to instantiate event handler objects. The CEventHandler class member function is invoked by the DlgAdminConsumer object when an event is received from the OA&M event notification framework. The CEventHandler class uses the DlgEventService namespace.

The CEventHandler adaptor class contains the following virtual function:

**CEventHandlerAdaptor::HandleEvent( )**
User-defined event handler that is invoked when a DlgAdminConsumer object receives an event.

# Function Information 2

This chapter provides a reference to the functions in the OA&M API library. Functions are listed according to class membership.

## 2.1 Function Syntax Conventions

The OA&M API functions use the following syntax:

```
return_type function_name(parameter1,....parameterN)
```

where:

return_type
: indicates the data type of the return value

function_name
: represents the function name

parameter1
: refers to the first parameter

parameterN
: represents the last parameter

# CCTDomain::CCTDomain( )

**Name:** CCTDomain(szNameServerIP, lNameServerPortNumber)

**Inputs:** const std::string szNameServerIP — IP address of the name server

const long lNameServerPortNumber — port number of the name server

**Returns:** nothing for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **CCTDomain( )** function is the CCTDomain class constructor, it is used to instantiate a CCTDomain object as an entry point into the OA&M system.

***Note:*** Use the C++ `new` operator to automatically instantiate a CCTDomain object without using the **CCTDomain( )** function parameters.

| Parameter | Description |
|---|---|
| **szNameServerIP** | indicates the IP address of the name server |
| **lNameServerPortNumber** | specifies the port number of the name server |

## Cautions

- Parameters for the **CCTDomain( )** function must be set to their default values, all other parameter values cause the OA&M API to throw an exception. The default parameter values are as follows:
  - **szNameServerIP**=127.0.0.1
  - **lNameServerPortNumber**=0

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **CCTDomain( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;
```

```
//Synchronize C and C++ IO
ios::sync_with_stdio();

cout <<"OA&M API Board Level Information"<< endl;

//Get Domain Object
pDomain = new CCTDomain();

//get node from domain using the default parameter
pNode = pDomain->GetNode("127.0.0.1");

   try
   {
   //get number of boards installed in node
   cout << "Number of Installed boards: "
        << pNode->GetNumberOfInstalledBoards()
        << endl;

   //get full board list
   pBoardList = pNode->GetAllInstalledBoards();

   //List physical identification attributes of all installed boards
      cout<<endl<<endl;
      cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
      ICTBoard  *brdPointers[20];
      int idx = 0;
      for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
         {
         pBoard = *i;
         brdPointers[idx] = pBoard;
         cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
             <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
             <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
             <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
             <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
             <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
             <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
         //don't need board memory anymore
          delete pBoard;
         }
   }
   catch (CCTDasiException e)
   {
   cerr<<"Exception: "<<endl;
   cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
   cerr<< "Error String: "<<e.m_errDesc<<endl;
   exit(1);
   }

//don't need these anymore
delete pBoardList;
delete pNode;
delete pDomain;
return;
}
```

## See Also

- **CCTDomain::GetNode( )**

# CCTDomain::GetNode( )

**Name:** ICTNode* GetNode(rszNodeIP)

**Inputs:** const std::string& rszNodeIP • reference to a constant string

**Returns:** pointer to an ICTNode object for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetNode( )** function returns a pointer to an instance of the ICTNode class (ICTNode object). The ICTNode class contains functions that allow you to get information about an Intel® Dialogic node.

| Parameter | Description |
|---|---|
| **rszNodeIP** | specifies a reference to a constant string that contains the node's IP address |

## Cautions

- The **rszNodeIP** parameter must be set to "127.0.0.1"; other values will cause the OA&M API to throw an exception.
- The memory for each ICTNode object that is returned by the **GetNode( )** function is allocated to the OA&M application. The memory must be manually deleted with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetNode( )** function.

## Example

The following example code includes a try/catch block that uses an incorrect default parameter for the **GetNode( )** function ("128.0.0.1" instead of "127.0.0.1"). A CCTDasiException object is thrown and caught by the system.

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;
```

```
int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
             //don't need board memory anymore
             delete pBoard;
            }
      }
      catch (CCTDasiException e)
      {
```

```
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }


    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;


return;
}
```

## See Also

- **CCTDomain::CCTDomain( )**

# ICTNode::GetAllInstalledBoards( )

**Name:** ICTBOARDLIST* GetAllInstalledBoards(void)

**Returns:** pointer to a list of pointers to ICTBoard objects for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetAllInstalledBoards( )** function returns a pointer to a Standard Template Library (STL) list of pointers to instances of the ICTBoard class (ICTBoard objects). The ICTBoard class contains functions that allow you to get information about individual Intel Dialogic boards.

Each ICTBoard object in the returned STL list represents a board that is installed in the node. A board is considered installed when the system software detects it in the chassis.

## Cautions

The memory for each returned ICTBoard object is allocated to the OA&M application. The memory must be manually deleted with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetAllInstalledBoards( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //get node from domain using the default parameter
   pNode = pDomain->GetNode("127.0.0.1");
```

```
      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
             //don't need board memory anymore
             delete pBoard;
            }
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
   for (i=pBoardList->begin(); i !=pBoardList->end(); i++,idx++) {
    pBoard = *i;
    delete pBoard;
    }

    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;

return;
}
```

## See Also

- **ICTNode::GetBoardByID( )**
- **ICTNode::GetNumberOfInstalledBoards( )**
- **ICTNode::GetSystemReleaseVersionInfo( )**

# ICTNode::GetBoardByID( )

**Name:** ICTBoard* GetBoardByID(rBoardID)

**Inputs:** const AUID& rBoardID • reference to the AUID of a board

**Returns:** pointer to an ICTBoard instance for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetBoardByID( )** function returns a pointer to the ICTBoard instance represented by the given Addressable Unit Identifier (AUID).

| Parameter | Description |
|---|---|
| **rBoardID** | specifies a reference to the constant long integer AUID of a board |

## Cautions

The memory pointed to by the returned ICTBoard object is allocated to the OA&M application. The application must manually delete the memory with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetBoardByID( )** function.

## Example

The following sample code uses the **GetBoardByID( )** function to get a pointer to the ICTBoard object represented by an AUID of 50285:

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   //Get Domain Object
   pDomain = new CCTDomain();
```

```
   //get node from domain using the default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //get board with AUID of 50285
   try
      {
      pBoard = pNode->GetBoardByID(50285);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }


    /* Freeing the board */
    delete pBoard;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;


return;
}
```

## See Also

- **ICTNode::GetAllInstalledBoards( )**
- **ICTNode::GetNumberOfInstalledBoards( )**

# ICTNode::GetNumberOfInstalledBoards( )

**Name:** long GetNumberOfInstalledBoards(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetNumberOfInstalledBoards( )** function returns the number of boards installed in a node. A board is considered installed when the system software detects it in the chassis.

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetNumberOfInstalledBoards( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   //Get Domain Object
   pDomain = new CCTDomain();


   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards();
           << endl;
      }
      catch (CCTDasiException e)
      {
```

```
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;
return;
}
```

## See Also

- **ICTNode::GetAllInstalledBoards( )**
- **ICTNode::GetBoardByID( )**

# ICTNode::GetSystemReleaseVersionInfo( )

**Name:** const CTPLATFORMVERSIONINFO GetSystemReleaseVersionInfo(void)

**Returns:** CTPLATFORMVERSIONINFO data structure for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetSystemReleaseVersionInfo( )** function returns a pointer to the CTPLATFORMVERSIONINFO data structure. This data structure contains information about the Intel® Dialogic System Software that is installed on the node.

## Cautions

Any application using the **GetSystemReleaseVersionInfo( )** function will need to be recompiled for the current Intel® Dialogic system software release. As part of the current release, additional data fields have been added to the CTPLATFORMVERSIONINFO data structure. Your application must be recompiled to incorporate the updated data structure.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetSystemReleaseVersionInfo( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main (void)
{
    CCTDomain *pDomain;
    ICTNode *pNode;
    CTPLATFORMVERSIONINFO  versionInfo;


    //instantiate domain
    pDomain=new CCTDomain();

    //get Node pointer
    pNode=pDomain->GetNode("127.0.0.1");

    //get software version info
    versionInfo = pNode->GetSystemReleaseVersionInfo();
    cout <<"+++Current System Software Version Information++++"<<endl;

    PrintVersionInfo(&versionInfo);
```

```
    //delete allocated memory
    delete pDomain;
    delete pNode;
    delete pVersionInfo;
    return;
}

void PrintVersionInfo(CTPLATFORMVERSIONINFO* VersionInfo)

{

    //Print info
     cout << "==================================" << endl;
     cout << "System Software Version Information" <<endl;
     cout << "==================================" <<endl;
     cout << "Major Release Number : "<<VersionInfo->lSystemReleaseMajorReleaseNumber<<endl;
     cout << "Minor Release Number : "<<VersionInfo->lSystemReleaseNinorReleaseNumber<<endl;
     cout << "SubMinor Release Number : "<<VersionInfo-
>lSystemReleaseSubMinorReleaseNumber<<endl;
     cout << "System Release Build Number: "<<VersionInfo->lSystemReleaseBuildNumber<<endl;
     cout << "Service Pack Number : "<<VersionInfo->lSystemServicePackNumber<<endl;
     cout << "Service Pack Build Number : "<<VersionInfo->lSystemServicePackBuildNumber<<endl;
     cout << "Feature Pack Number : "<<VersionInfo->lSystemFeaturePackNumber<<endl;
     cout << "Feature Pack Build Number : "<<VersionInfo->lSystemFeaturePackBuildNumber<<endl;



    cout << "Service Update Number : "<<VersionInfo-> lServiceUpdateNumber <<endl;
    cout << "Service Update Build Number : "<<VersionInfo-> lServiceUpdateBuildNumber <<endl;
    cout << "Feature Release Number : "<<VersionInfo-> lFeatureReleaseNumber <<endl;
    cout << "Feature Release Build Number : "<<VersionInfo-> lFeatureReleaseBuildNumber <<endl;

}
```

## See Also

- **ICTNode::GetAllInstalledBoards( )**
- **ICTNode::GetBoardByID( )**
- **ICTNode::GetNumberOfInstalledBoards( )**

# ICTBoard::GetBoardID( )

**Name:** const AUID GetBoardID(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetBoardID( )** function returns the unique Addressable Unit Identifier (AUID) of a board. The board does not have to be started to call the **GetBoardID( )** function.

An AUID is a unique string of digits that identifies a component with which communications may be initiated. The system software assigns an AUID to the following components:

- Intel NetStructure®, DM3 architecture or Springware architecture boards
- Intel NetStructure board clocking agents

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetBoardID( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();
```

```
   //get node from domain using the default parameter
   pNode = pDomain->GetNode("127.0.0.1");

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
                <<"Board AUID:                 "<<pBoard->GetBoardID()<<endl;
            //don't need board memory anymore
             delete pBoard;
            }
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;

return;
}
```

## See Also

- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetPCIBus( )**
- **ICTBoard::GetPCISlot( )**

- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetClockingAgents( )

**Name:** ICTCLOCKAGENTLIST* GetClockingAgents(void)

**Returns:** pointer to a list of pointers to ICTClockAgent objects for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetClockingAgents( )** function returns a pointer to a Standard Template Library (STL) list of pointers to instances of the ICTClockAgent class (ICTClockAgent objects). The ICTClockAgent class contains functions that allow you to get information about a clocking agent's TDM bus configuration.

Each ICTClockAgent object in the returned STL list represents one clocking agent on the board. Each board contains at least one clocking agent. Clocking agents are electronically connected to the TDM bus and control access to the TDM bus clocking facilities. Therefore, a board's TDM bus configuration is set at the clocking agent level.

## Cautions

- The **GetClockingAgents( )** function cannot be called for a board that is in a 'stopped' state. In order to call this function, the board must be started individually (`startbrd` utility) or as part of the system startup routine (`dlstart` utility).
- The memory for each returned ICTClockAgent object is allocated to the OA&M application. The memory must be manually deleted with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **CCTDomain( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

//FUNCTION PROTOTYPES
void CapabilitiesTest(ICTClockAgent   *pAgent);


//FUNCTIONS
string& boolval (bool val)
{
   static string tr="true";
   static string fa="false";
```

```
   if (val)
      return tr;
   else
      return fa;
}

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;
   bool           rc=false;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
            }
         int chosen_board=-1;
         while (chosen_board<0) || (chosen_board>=idx))
            {
            cout <<endl << "====> Choose Board for test: ";
            char buffer[3];
            fgets(buffer,3,stdin);
            chosen_board = atoi(buffer);
            }
            agent_menu(brdPointers[chosen_board];
      }

      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
return;
}
```

```
//agent menu function
void agent_menu(ICTBoard *pBoard)
{
ICTCLOCKAGENTLIST *pAgentList;
ICTLCLOCKAGENTLIST::iterator j;

   if (pBoard->GetNumClockingAgents()>0
      {
      pAgentList = pBoard->GetClockingAgents();
      for (j=pAgentList->begin();j!pAgentList->end();j++)
         {
         CapabilitiesTest(*j);
         }
      }
   else
      {
      cout <<"No Clocking Agent on board"<<endl;
      }

   /* Freeing the clocking agents in the agent list */
   for (j=pAgentList->begin(); j !=pAgentList->end(); j++)
      {
      ICTClockAgent   *pAgent;
      // Delete the pAgent
      pAgent = *j;
      delete pAgent;
      }
       /* Freeing the agent list */
       delete pAgentList;
return;
{

//print TDM capabilities of clocking agent
void CapabilitiesTest(ICTClockAgent *pAgent)
{
   AGENTTDMCAPABILITES *pCaps;

   pCaps = pAgent->GetTDMCapabilities();

   //Print capabilities
   cout <<"======================== " <<endl;
   cout <<"CLOCK AGENT CAPABILITIES "<<endl;
   cout <<"======================== "<<endl;
   cout << "Structure version           : "<<
      (int)(pCaps->structVersion<< endl;
         cout << "h100master Capable         : "
         << boolval(pCaps->canH100Master)<<endl;
         cout << "h100Slave Capable          : "
         << boolval(pCaps->canH100Slave)<<endl;
         cout << "h110Master Capable         : "
         << boolval(pCaps->canH110Master)<<endl;
         cout << "h110Slave Capable          : "
         << boolval(pCaps->canH110Slave)<<endl;
         cout << "SCBUS Master Capable       : "
         << boolval(pCaps->canScbusMaster)<<endl;
         cout << "SCBUS Slave Capable        : "
         << boolval(pCaps->canScbusSlave)<<endl;
         cout << "MVIP Master Capable        : "
         << boolval(pCaps->CanMvipMaster)<<endl;
         cout << "MVIP Slave Capable         : "
         << boolval(pCaps->canMvipSlave)<<endl;
         cout << "SCBUS at 2Mhz              : "
         << boolval(pCaps->canScbus2Mhz)<<endl;
         cout << "SCBUS at 4Mhz              : "
         << boolval(pCaps->canScbus4Mhz)<<endl;
         cout << "SCBUS at 8Mhz              : "
         << boolval(pCaps->canScbus8Mhz)<<endl;
```

```
        cout << "Netref Provide At 8Khz     : "
        << boolval(pCaps->canProvideNetrefAt8Khz)<<endl;
        cout << "Netref Provide At 1.536Mhz : "
        << boolval(pCaps->canProvideNetrefAt1536Mhz)<<endl;
        cout << "Netref Provide At 1.544Mhz : "
        << boolval(pCaps->canProvideNetrefAt1544Mhz)<<endl;
        cout << "Netref Provide At 2.048Mhz : "
        << boolval(pCaps->canProvideNetrefAt2048Mhz)<<endl;
        cout << "Netref Derive At 8Khz      : "
        << boolval(pCaps->canDeriveNetrefAt8Khz)<<endl;
        cout << "Netref Derive At 1.536Mhz  : "
        << boolval(pCaps->canDeriveNetrefAt1536Mhz)<<endl;
        cout << "Netref Derive At 1.544Mhz  : "
        << boolval(pCaps->canDeriveNetrefAt1544Mhz)<<endl;
        cout << "Netref Derive At 2.048Mhz  : "
        << boolval(pCaps->canDeriveNetrefAt2048Mhz)<<endl;
        cout << "Netref2 Provide At 8Khz    : "
        << boolval(pCaps->canProvideNetref2At8Khz)<<endl;
        cout << "Netref2 Provide At 1.536Mhz: "
        << boolval(pCaps->canProvideNetref2At1536Mhz)<<endl;
        cout << "Netref2 Provide At 1.544Mhz: "
        << boolval(pCaps->canProvideNetref2At1544Mhz)<<endl;
        cout << "Netref2 Provide At 2.048Mhz: "
        << boolval(pCaps->canProvideNetref2At2048Mhz)<<endl;
        cout << "Netref2 Derive At 8Khz     : "
        << boolval(pCaps->canDeriveNetref2At8Khz)<<endl;
        cout << "Netref2 Derive At 1.536Mhz : "
        << boolval(pCaps->canDeriveNetref2At1536Mhz)<<endl;
        cout << "Netref2 Derive At 1.544Mhz : "
        << boolval(pCaps->canDeriveNetref2At1544Mhz)<<endl;
        cout << "Netref2 Derive At 2.048Mhz : "
        << boolval(pCaps->canDeriveNetref2At2048Mhz)<<endl;
        cout << "canSilenceScbusCompatLines : "
        << boolval(pCaps->canSilenceScbusCompatLines)<<endl;
        cout << "canSilenceMvipCompatLines  : "
        << boolval(pCaps->canSilenceMvipCompatLines)<<endl;
        cout << "canDeriveNetrefFromOsc     : "
        << boolval(pCaps->canDeriveNetrefFromOsc)<<endl;
        cout << "Can Terminate bus          : "
        << boolval(pCaps->canTerminate)<<endl;
        cout << "Can DynamicConfigure       : "
        << boolval(pCaps->canDynamicConfigure)<<endl;
        cout << "Can Use Osc as Master Clock: "
        << boolval(pCaps->canUseLocalDNIAsMasterSource)<<endl;
        cout << "Max number of Master Sources: "
        << boolval(pCaps->maxClockMasterSourceListSize)<<endl;

    // Delete capabilities structure

    delete pCaps;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetNumClockingAgents( )

**Name:** const short GetNumClockingAgents(void)

**Returns:** short integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetNumClockingAgents( )** function gets the number of addressable clocking agents on a board that has been started by the system. Each board contains at least one clocking agent. Clocking agents are electronically connected to the TDM bus and control access to the TDM bus clocking facilities. Therefore, a board's TDM bus configuration is set at the clocking agent level.

## Cautions

The **GetNumClockingAgents( )** function cannot be called for a board that is in a 'stopped' state. In order to call this function, the board must be started individually (`startbrd` utility) or as part of the system startup routine (`dlstart` utility).

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetNumClockingAgents( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");
```

```
        try
        {
        //get number of boards installed in node
        cout << "Number of Installed boards: "
             << pNode->GetNumberOfInstalledBoards()
             << endl;

        //get full board list
        pBoardList = pNode->GetAllInstalledBoards();

        //List physical identification attributes of all installed boards
           cout<<endl<<endl;
           cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
           ICTBoard  *brdPointers[20];
           int idx = 0;
           for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
              {
              pBoard = *i;
              brdPointers[idx] = pBoard;
              cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                  <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                  <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                  <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                  <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                  <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                  <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
              //don't need board memory anymore
               delete pBoard;
              }
        }
        catch (CCTDasiException e)
        {
        cerr<<"Exception: "<<endl;
        cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
        cerr<< "Error String: "<<e.m_errDesc<<endl;
        exit(1);
        }

       /* Freeing the board list */
       delete pBoardList;

       /* Freeing the node object */
       delete pNode;

       /* Freeing the domain object */
      delete pDomain;

return;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetPCIBus( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetNumTrunks( )

**Name:** const short GetNumTrunks(void)

**Returns:** short integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetNumTrunks( )** function returns the number of T1/E1 trunk interfaces on a board that has been started by the system.

## Cautions

The **GetNumTrunks( )** function cannot be called for a board that is in a 'stopped' state. In order to call this function, the board must be started individually (`startbrd` utility) or as part of the system startup routine (`dlstart` utility).

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetNumTrunks( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");
```

```
        try
        {
        //get number of boards installed in node
        cout << "Number of Installed boards: "
             << pNode->GetNumberOfInstalledBoards()
             << endl;

        //get full board list
        pBoardList = pNode->GetAllInstalledBoards();

        //List physical identification attributes of all installed boards
           cout<<endl<<endl;
           cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
           ICTBoard  *brdPointers[20];
           int idx = 0;
           for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
              {
              pBoard = *i;
              brdPointers[idx] = pBoard;
              cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                  <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                  <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                  <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                  <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                  <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                  <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
              //don't need board memory anymore
               delete pBoard;
              }
        }
        catch (CCTDasiException e)
        {
        cerr<<"Exception: "<<endl;
        cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
        cerr<< "Error String: "<<e.m_errDesc<<endl;
        exit(1);
        }
      /* Freeing the board list */
      delete pBoardList;

      /* Freeing the node object */
      delete pNode;

      /* Freeing the domain object */
     delete pDomain;

return;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetPCIBus( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetPCIBus( )

**Name:** const long GetPCIBus(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetPCIBus( )** function returns the PCI bus number a board is installed on. The board does not have to be started to call the **GetPCIBus( )** function.

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetPCIBus( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
```

```
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus();
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
            //don't need board memory anymore
             delete pBoard;
            }
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
    //don't need these anymore
   delete pDomain;
   delete pNode;
   delete pBoardList;
return;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetPCISlot( )

**Name:** const long GetPCISlot(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetPCISlot( )** function returns the PCI slot that a board is installed in. The board does not have to be started to call the **GetPCISlot( )** function.

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetPCISlot( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();


   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
```

```
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
            //don't need board memory anymore
            delete pBoard;
            }
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
   //don't need these anymore
   delete pDomain;
   delete pNode;
   delete pBoardList;
return;
}
```

## See Also

- ICTBoard::GetBoardID( )
- ICTBoard::GetClockingAgents( )
- ICTBoard::GetNumTrunks( )
- ICTBoard::GetNumClockingAgents( )
- ICTBoard::GetPCIBus( )
- ICTBoard::GetPhysicalSlot( )
- ICTBoard::GetSerialNumber( )
- ICTBoard::GetShelfId( )

# ICTBoard::GetPhysicalSlot( )

**Name:** const long GetPhysicalSlot(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetPhysicalSlot( )** function returns the physical slot number a cPCI board is installed in. The board does not have to be started to call the **GetPhysicalSlot( )** function.

## Cautions

The **GetPhysicalSlot( )** function only applies to cPCI boards.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetPhysicalSlot( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
```

```
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
            //don't need board memory anymore
             delete pBoard;
            }
         }
      catch (CCTDasiException e)

      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
    //don't need these anymore
   delete pDomain;
   delete pNode;
   delete pBoardList;
return;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetPCIBus( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetSerialNumber( )**
- **ICTBoard::GetShelfId( )**

# ICTBoard::GetSerialNumber( )

**Name:** const std::string& GetSerialNumber(void)

**Returns:** reference to a constant string for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetSerialNumber( )** function returns the serial number of a board. The board does not have to be started to call the **GetSerialNumber( )** function.

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetSerialNumber( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
```

```
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
                <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
                <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
                <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
                <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
                <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
            //don't need board memory anymore
             delete pBoard;
            }
      }

      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }
    //don't need these anymore
   delete pDomain;
   delete pNode;
   delete pBoardList;
return;
}
```

## See Also

- ICTBoard::GetBoardID( )
- ICTBoard::GetClockingAgents( )
- ICTBoard::GetNumTrunks( )
- ICTBoard::GetNumClockingAgents( )
- ICTBoard::GetPhysicalSlot( )
- ICTBoard::GetPCIBus( )
- ICTBoard::GetPCISlot( )
- ICTBoard::GetShelfId( )

# ICTBoard::GetShelfId( )

**Name:** const long GetShelfId(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetShelfId**( ) function returns the shelf ID reported by a cPCI board. Individual cPCI chassis can be programmatically assigned unique shelf ID numbers, the shelf ID number for a chassis can then be reported by any board that is plugged into the chassis backplane.

## Cautions

- The **GetShelfId**( ) function only applies to cPCI boards.
- The **GetShelfId**( ) function cannot be called for a board that is in a 'stopped' state. In order to call this function, the board must be started individually (`startbrd` utility) or as part of the system startup routine (`dlstart` utility).

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetShelfId**( ) function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();
```

```
//Initiate and catch DLG_INVALID_SERVER exception
try
   {
   cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
   //Attempt to get node from domain using incorrect default parameter. Exception thrown.
   pNode = pDomain->GetNode("128.0.0.1");
   }
catch (CCTDasiException e)
   {
   cerr<<"Exception: "<<endl;
   cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
   cerr<<"Error String: "<<e.m_errDesc<<endl;
   }

//get node from domain using the correct default parameter
pNode = pDomain->GetNode("127.0.0.1");

//try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
try
   {
   cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
   //throw exception due to invalid board AUID of 999999
   pNode->GetBoardByID(999999);
   }
   catch (CCTDasiException e)
   {
   cerr<<"Exception: "<<endl;
   cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
   cerr<<"Error String: "<<e.m_errDesc<<endl;
   }

   try
   {
   //get number of boards installed in node
   cout << "Number of Installed boards: "
        << pNode->GetNumberOfInstalledBoards()
        << endl;

   //get full board list
   pBoardList = pNode->GetAllInstalledBoards();

   //List physical identification attributes of all installed boards
      cout<<endl<<endl;
      cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
      ICTBoard  *brdPointers[20];
      int idx = 0;
      for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
         {
         pBoard = *i;
         brdPointers[idx] = pBoard;
         cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
             <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
             <<"Physical Slot:             "<<pBoard->GetPhysicalSlot()<<endl;
             <<"Number of Trunks:          "<<pBoard->GetNumTrunks()<<endl;
             <<"Shelf ID:                  "<<pBoard->GetShelfId()<<endl;
             <<"Serial Number:             "<<pBoard->GetSerialNumber()<<endl;
             <<"Number of Clocking Agents: "<<pBoard->GetNumClockingAgents()<<endl;
         //don't need board memory anymore
          delete pBoard;
         }
   }

   catch (CCTDasiException e)
   {
   cerr<<"Exception: "<<endl;
   cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
   cerr<< "Error String: "<<e.m_errDesc<<endl;
```

```
      exit(1);
      }
   //don't need these anymore
   delete pDomain;
   delete pNode;
   delete pBoardList;
return;
}
```

## See Also

- **ICTBoard::GetBoardID( )**
- **ICTBoard::GetClockingAgents( )**
- **ICTBoard::GetNumTrunks( )**
- **ICTBoard::GetNumClockingAgents( )**
- **ICTBoard::GetPhysicalSlot( )**
- **ICTBoard::GetPCIBus( )**
- **ICTBoard::GetPCISlot( )**
- **ICTBoard::GetSerialNumber( )**

# ICTClockAgent::GetClockAgentID( )

**Name:** const AUID GetClockAgentID(void)

**Returns:** long integer for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetClockingAgentID( )** function returns the unique Addressable Unit Identifier (AUID) of a board's clocking agent.

An AUID is a unique string of digits that identifies a component with which communications may be initiated. The system software assigns an AUID to the following system components:

- Intel NetStructure®, DM3 architecture or Springware architecture boards
- Intel NetStructure board clocking agents

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **CCTDomain( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

    //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");
```

```
      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
            }
         int chosen_board=-1;
         while (chosen_board<0) || (chosen_board>=idx))
            {
            cout <<endl << "====> Choose Board for test: ";
            char buffer[3];
            fgets(buffer,3,stdin);
            chosen_board = atoi(buffer);
            }
            agent_menu(brdPointers[chosen_board];
      }

      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }



    /* Freeing the board */
    delete pBoard;


    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;
return;
}

//clocking agent menu function
void agent_menu(ICTBoard *pBoard)
{
ICTCLOCKAGENTLIST *pAgentList;
ICTCLOCKAGENTLIST::iterator j;
```

```
   if (pBoard->GetNumClockingAgents()>0
      {
      pAgentList = pBoard->GetClockingAgents();
      for (j=pAgentList->begin();j!pAgentList->end();j++)
         {
         cout<<j<<"Clocking Agent AUID:  "<<pAgentList->GetClockAgentID()
         }
      }
   else
      {
      cout <<"No Clocking Agent on board"<<endl;
      }
   /* Freeing the clocking agents in the agent list */
      for (j=pAgentList->begin(); j !=pAgentList->end(); j++) {
           ICTClockAgent   *pAgent;
           // Delete the pAgent
              pAgent = *j;
              delete pAgent;
            }
   /* Freeing the agent list */
     delete pAgentList;
return;
{
```

## See Also

- **ICTClockAgent::GetTDMCapabilities( )**
- **ICTClockAgent::GetTDMConfiguration( )**
- **ICTClockAgent::SetTDMConfiguration ( )**

# ICTClockAgent::GetTDMCapabilities( )

**Name:** const AGENTTDMCAPABILITIES* GetTDMCapabilities(void)

**Returns:** pointer to the AGENTTDMCAPABILITIES data structure for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetTDMCapabilities( )** function returns a pointer to the AGENTTDMCAPABILITIES data structure. This data structure contains the TDM bus capabilities of a board's clocking agent.

***Note:*** For the current release, the AGENTTDMCAPABILITIES data structure's structVersion field will be set to 0.

## Cautions

Memory for the AGENTTDMCAPABILITIES pointer that is returned by **GetTDMCapabilities( )** function should be deleted with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetTDMCapabilities( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

//FUNCTION PROTOTYPES
void CapabilitiesTest(ICTClockAgent   *pAgent);


//FUNCTIONS
string& boolval (bool val)
{
   static string tr="true";
   static string fa="false";

   if (val)
      return tr;
   else
      return fa;
}
```

```
int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;
   bool           rc=false;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
```

```
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
             }
          int chosen_board=-1;
          while (chosen_board<0) || (chosen_board>=idx))
             {
             cout <<endl << "====> Choose Board for test: ";
             char buffer[3];
             fgets(buffer,3,stdin);
             chosen_board = atoi(buffer);
             }
             agent_menu(brdPointers[chosen_board];
       }

       catch (CCTDasiException e)
       {
       cerr<<"Exception: "<<endl;
       cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
       cerr<< "Error String: "<<e.m_errDesc<<endl;
       exit(1);
       }
return;
}

//agent menu function
void agent_menu(ICTBoard *pBoard)
{
ICTCLOCKAGENTLIST *pAgentList;
ICTLCLOCKAGENTLIST::iterator j;

   if (pBoard->GetNumClockingAgents()>0
      {
      pAgentList = pBoard->GetClockingAgents();
      for (j=pAgentList->begin();j!pAgentList->end();j++)
         {
         CapabilitiesTest(*j);
         }
      }
   else
      {
      cout <<"No Clocking Agent on board"<<endl;
      }


    /* Freeing the clocking agents in the agent list */
      for (j=pAgentList->begin(); j !=pAgentList->end(); j++) {
           ICTClockAgent   *pAgent;
           // Delete the pAgent
              pAgent = *j;
              delete pAgent;
           }
   /* Freeing the agent list */
     delete pAgentList;


   /* Freeing the board */
    delete pBoard;

    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;
```

```
    /* Freeing the domain object */
   delete pDomain;

return;
{

//print TDM capabilities of clocking agent
void CapabilitiesTest(ICTClockAgent *pAgent)
{
   AGENTTDMCAPABILITES *pCaps;

   pCaps = pAgent->GetTDMCapabilities();

   //Print capabilities
   cout <<"======================== " <<endl;
   cout <<"CLOCK AGENT CAPABILITIES "<<endl;
   cout <<"======================== "<<endl;
   cout << "Structure version           : "<<
      (int)(pCaps->structVersion<< endl;
         cout << "h100master Capable         : "
         << boolval(pCaps->canH100Master)<<endl;
         cout << "h100Slave Capable          : "
         << boolval(pCaps->canH100Slave)<<endl;
         cout << "h110Master Capable         : "
         << boolval(pCaps->canH110Master)<<endl;
         cout << "h110Slave Capable          : "
         << boolval(pCaps->canH110Slave)<<endl;
         cout << "SCBUS Master Capable       : "
         << boolval(pCaps->canScbusMaster)<<endl;
         cout << "SCBUS Slave Capable        : "
         << boolval(pCaps->canScbusSlave)<<endl;
         cout << "MVIP Master Capable        : "
         << boolval(pCaps->CanMvipMaster)<<endl;
         cout << "MVIP Slave Capable         : "
         << boolval(pCaps->canMvipSlave)<<endl;
         cout << "SCBUS at 2Mhz              : "
         << boolval(pCaps->canScbus2Mhz)<<endl;
         cout << "SCBUS at 4Mhz              : "
         << boolval(pCaps->canScbus4Mhz)<<endl;
         cout << "SCBUS at 8Mhz              : "
         << boolval(pCaps->canScbus8Mhz)<<endl;
         cout << "Netref Provide At 8Khz     : "
         << boolval(pCaps->canProvideNetrefAt8Khz)<<endl;
         cout << "Netref Provide At 1.536Mhz : "
         << boolval(pCaps->canProvideNetrefAt1536Mhz)<<endl;
         cout << "Netref Provide At 1.544Mhz : "
         << boolval(pCaps->canProvideNetrefAt1544Mhz)<<endl;
         cout << "Netref Provide At 2.048Mhz : "
         << boolval(pCaps->canProvideNetrefAt2048Mhz)<<endl;
         cout << "Netref Derive At 8Khz      : "
         << boolval(pCaps->canDeriveNetrefAt8Khz)<<endl;
         cout << "Netref Derive At 1.536Mhz  : "
         << boolval(pCaps->canDeriveNetrefAt1536Mhz)<<endl;
         cout << "Netref Derive At 1.544Mhz  : "
         << boolval(pCaps->canDeriveNetrefAt1544Mhz)<<endl;
         cout << "Netref Derive At 2.048Mhz  : "
         << boolval(pCaps->canDeriveNetrefAt2048Mhz)<<endl;
         cout << "Netref2 Provide At 8Khz    : "
         << boolval(pCaps->canProvideNetref2At8Khz)<<endl;
         cout << "Netref2 Provide At 1.536Mhz: "
         << boolval(pCaps->canProvideNetref2At1536Mhz)<<endl;
         cout << "Netref2 Provide At 1.544Mhz: "
         << boolval(pCaps->canProvideNetref2At1544Mhz)<<endl;
         cout << "Netref2 Provide At 2.048Mhz: "
         << boolval(pCaps->canProvideNetref2At2048Mhz)<<endl;
         cout << "Netref2 Derive At 8Khz     : "
         << boolval(pCaps->canDeriveNetref2At8Khz)<<endl;
```

```
        cout << "Netref2 Derive At 1.536Mhz : "
        << boolval(pCaps->canDeriveNetref2At1536Mhz)<<endl;
        cout << "Netref2 Derive At 1.544Mhz : "
        << boolval(pCaps->canDeriveNetref2At1544Mhz)<<endl;
        cout << "Netref2 Derive At 2.048Mhz : "
        << boolval(pCaps->canDeriveNetref2At2048Mhz)<<endl;
        cout << "canSilenceScbusCompatLines : "
        << boolval(pCaps->canSilenceScbusCompatLines)<<endl;
        cout << "canSilenceMvipCompatLines  : "
        << boolval(pCaps->canSilenceMvipCompatLines)<<endl;
        cout << "canDeriveNetrefFromOsc     : "
        << boolval(pCaps->canDeriveNetrefFromOsc)<<endl;
        cout << "Can Terminate bus          : "
        << boolval(pCaps->canTerminate)<<endl;
        cout << "Can DynamicConfigure       : "
        << boolval(pCaps->canDynamicConfigure)<<endl;
        cout << "Can Use Osc as Master Clock: "
        << boolval(pCaps->canUseLocalDNIAsMasterSource)<<endl;
        cout << "Max number of Master Sources: "
        << (int)(pCaps->maxClockMasterSourceListSize)<<endl;

    // Delete capabilities structure

    delete pCaps;
}
```

## See Also

- **ICTClockAgent::GetClockAgentID( )**
- **ICTClockAgent::GetTDMConfiguration( )**
- **ICTClockAgent::SetTDMConfiguration ( )**

# ICTClockAgent::GetTDMConfiguration( )

**Name:** const AGENTTDMCONFIGURATION* GetTDMConfiguration(void)

**Returns:** pointer to the AGENTTDMCONFIGURATION data structure for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** synchronous

---

## Description

The **GetTDMConfiguration( )** function returns a pointer to the AGENTTDMCONFIGURATION data structure. This data structure contains the current TDM bus settings for a board's clocking agent.

*Note:* For the current release, the AGENTTDMCONFIGURATION data structure's structVersion field will always return a value of 0.

## Cautions

Memory for the AGENTTDMCONFIGURATION pointer that is returned by **GetTDMConfiguration( )** function should be deleted with the C++ `delete` operator.

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **GetTDMConfiguration( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

// PROTOTYPES
void ConfigurationTest(ICTClockAgent  *pAgent);

void PrintClockingModel(short model);
void PrintClockSource(long src);
void PrintBusType(short busType);
void PrintPrimaryLine(short line);
void PrintRole(short role);
void PrintClockRate(CLOCKRATE rate);
void PrintConfiguration(AGENTTDMCONFIGURATION  *cfg)

//FUNCTIONS
string& boolval (bool val)
{
   static string tr="true";
   static string fa="false";
```

```
   if (val)
      return tr;
   else
      return fa;
}

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
```

```
            for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
               {
               pBoard = *i;
               brdPointers[idx] = pBoard;
               cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                   <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
               }
            int chosen_board=-1;
            while (chosen_board<0) || (chosen_board>=idx))
               {
               cout <<endl << "====> Choose Board for test: ";
               char buffer[3];
               fgets(buffer,3,stdin);
               chosen_board = atoi(buffer);
               }
               agent_menu(brdPointers[chosen_board];
         }
         catch (CCTDasiException e)
         {
         cerr<<"Exception: "<<endl;
         cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
         cerr<< "Error String: "<<e.m_errDesc<<endl;
         exit(1);
         }

      for (i=pBoardList->begin(); i !=pBoardList->end(); i++,idx++) {
       pBoard = *i;
       delete pBoard;
       }

       /* Freeing the board list */
       delete pBoardList;

       /* Freeing the node object */
       delete pNode;

       /* Freeing the domain object */
      delete pDomain;

return;
}

//clocking agent menu function
void agent_menu(ICTBoard *pBoard)
{
ICTCLOCKAGENTLIST *pAgentList;
ICTLCLOCKAGENTLIST::iterator j;

   if (pBoard->GetNumClockingAgents()>0
      {
      pAgentList = pBoard->GetClockingAgents();
      for (j=pAgentList->begin();j!pAgentList->end();j++)
         {
         ConfigurationTest(*j);
         }
      }
   else
      {
      cout <<"No Clocking Agent on board"<<endl;
      }

     /* Freeing the clocking agents in the agent list */
      for (j=pAgentList->begin(); j !=pAgentList->end(); j++) {
           ICTClockAgent   *pAgent;
           // Delete the pAgent
           pAgent = *j;
           delete pAgent;
```

```
          }
     /* Freeing the agent list */
      delete pAgentList;
return;
{

//print TDM configuration of clocking agent
void ConfigurationTest(ICTClockAgent *pAgent)
{
   AGENTTDMCONFIGURATION *pCfg;

   pCfg = pAgent->GetTDMConfiguration();

   cout << "*+*_+*+*+*+*+*+*+*+>>>Current Configuration of Clock Agent"<<endl;
   PrintConfiguration(pCfg);
   return;
}

void PrintConfiguration(AGENTTDMCONFIGURATION * cfg)
{
   // Print Clocking Agent Configuration
   cout << "============= " << endl;
   cout << "CONFIGURATION " << endl;
   cout << "============= " << endl;
   cout << "Structure version       : " << cfg->structVersion << endl;

   // TDM bus Status
   cout << "TDM BUS Status          : ";
   if (cfg->onTelephonyBus)
      cout << "ON"<<endl;
   else
      cout << "OFF"<<endl;

   // TDM bus type
   cout << "TDM BUS Type            : ";
   PrintBusType(cfg->busType);

   // Clock rates
   cout << "SCBUS Clockrate         : ";
   PrintClockRate(cfg->scBusClockRate);
   cout << "Group 1 Clockrate       : ";
   PrintClockRate(cfg->group1ClockRate);
   cout << "Group 2 Clockrate       : ";
   PrintClockRate(cfg->group2ClockRate);
   cout << "Group 3 Clockrate       : ";
   PrintClockRate(cfg->group3ClockRate);
   cout << "Group 4 Clockrate       : ";
   PrintClockRate(cfg->group4ClockRate);

   // Compatibility lines
   cout << "MVIP Compat Line active : " <<
      boolval(cfg->MVIPCompatLinesActive) << endl;
   cout << "SCBUS Compat Line active: "<<
      boolval(cfg->SCBusCompatLinesActive)<< endl;

   // Clock role
   cout << "Master Clock Role       : ";
   PrintRole(cfg->clockMasterRole);

   for (int i=0;i<16;i++)
      {
      cout << "Master Source ["<<i<<"] : ";
      PrintClockSource(cfg->clockMasterSourceList[i]);
      }

   cout << "Current Clocking Model  : ";
   PrintClockingModel(cfg->clockingModel);
```

```
   cout << "Master Primary Line     : ";
   PrintPrimaryLine(cfg->primaryLines);

   cout << "Providing Netref1       : "<< boolval(cfg->isProvidingNetref)<<endl;
   if (cfg->isProvidingNetref)
      {
      cout << "Netref 1 Clock Source   : ";
      PrintClockSource(cfg->netrefSource);
      cout << "Netref 1 Clockrate      : ";
      PrintClockRate(cfg->netrefClockRate);
      }

   cout << "Providing Netref2       : "<< boolval(cfg->isProvidingNetref2)<<endl;
   if (cfg->isProvidingNetref2)
      {
      cout << "Netref 2 Clock Source   : ";
      PrintClockSource(cfg->netref2Source);
      cout << "Netref 2 Clockrate      : ";
      PrintClockRate(cfg->netref2ClockRate);
      }

   cout << "Providing termination    : "<< boolval(cfg->termination)<<endl;

}

void PrintClockingModel(short model)
{
   switch(model)
   {
   caseNORMAL:
      cout << "NORMAL"<<endl;
      break;
   case HOLDOVER:
      cout << "HOLDOVER"<<endl;
      break;
   case FREE_RUN:
      cout << "FREE RUN"<<endl;
      break;
   case AUTO_HOLDOVER:
      cout << "Auto Holdover"<<endl;
      break;
   case AUTO_FREE_RUN:
      cout << "Auto Free Run"<<endl;
      break;
   default:
      cout << "UNKNOWN"<<endl;
      break;
   }
}

void PrintClockSource(long src)
{
   switch(src)
   {
   case CT_NETREF2:
      cout <<"Netref 2"<<endl;
      break;
   case CT_DNI_1:
      cout <<"External 1"<<endl;
      break;
   case CT_DNI_2:
      cout <<"External 2"<<endl;
      break;
   case CT_DNI_3:
      cout <<"External 3"<<endl;
      break;
   case CT_DNI_4:
```

```
      cout <<"External 4"<<endl;
      break;
   case CT_DNI_5:
      cout <<"External 5"<<endl;
      break;
   case CT_DNI_6:
      cout <<"External 6"<<endl;
      break;
   case CT_DNI_7:
      cout <<"External 7"<<endl;
      break;
   case CT_DNI_8:
      cout <<"External 8"<<endl;
      break;
   case CT_NETREF:
      cout <<"NetRef"<<endl;
      break;
   case CT_INT_CLK:
      cout <<"Internal Oscillator"<<endl;
      break;
   case CT_DISABLE:
      cout <<"Disabled"<<endl;
      break;
   default:
      cout << "UNKNOWN"<<endl;
      break;
   }
}

void PrintBusType(short busType)
{
   switch(busType)
   {
   case CT_BUSTYPE_MVIP:
      cout << "MVIP"<<endl;
      break;
   case CT_BUSTYPE_SCBUS:
      cout << "SCBUS"<<endl;
      break;
   case CT_BUSTYPE_H100:
      cout << "H100"<<endl;
      break;
   case CT_BUSTYPE_H110:
      cout << "H110"<<endl;
      break;
   default:
      cout << "UNKNOWN"<<endl;
      break;
   }
}

void PrintPrimaryLine(short line)
{
   switch(line)
   {
   case CT_CLKA:
      cout << "Clock line A"<<endl;
      break;
   case CT_CLKB:
      cout << "Clock Line B"<<endl;
      break;
   case CT_DIS:
      cout << "Disabled"<<endl;
      break;
```

```
   default:
      cout << "UNKNOWN:"<<line<<endl;
      break;
   }
}

void PrintRole(short role)
{
   switch(role)
   {
   case CT_PRIMARY:
      cout << "PRIMARY"<<endl;
      break;
   case CT_SECONDARY:
      cout << "SECONDARY"<<endl;
      break;
   case CT_SLAVE:
      cout << "SLAVE"<<endl;
      break;
   default:
      cout << "UNKNOWN"<<endl;
      break;
   }
}

void PrintClockRate(CLOCKRATE rate)
{
   switch(rate)
   {
   case CT_CLOCKRATE_NONE:
      cout << "NONE";
      break;
   case CT_CLOCKRATE_2MHZ:
      cout << "2MHZ";
      break;
   case CT_CLOCKRATE_4MHZ:
      cout << "4MHZ";
      break;
   case CT_CLOCKRATE_8MHZ:
      cout << "8MHZ";
      break;
   case CT_CLOCKRATE_8KHZ:
      cout << "8KHZ";
      break;
   case CT_CLOCKRATE_1_536MHZ:
      cout << "1.536MHZ";
      break;
   case CT_CLOCKRATE_1_544MHZ:
      cout << "1.544MHZ";
      break;
   case CT_CLOCKRATE_2_048MHZ:
      cout << "2.048MHZ";
      break;
   default:
      cout << "UNKNOWN";
   }
   cout << endl;
}
```

## See Also

- **ICTClockAgent::GetClockAgentID( )**
- **ICTClockAgent::GetTDMCapabilities( )**
- **ICTClockAgent::SetTDMConfiguration ( )**

# ICTClockAgent::SetTDMConfiguration ( )

**Name:** void SetTDMConfiguration(AGENTTDMCONFIGURATION& rTDMConfig)

**Inputs:** const AGENTTDMCONFIGURATION& rTDMConfig
- reference to the TDM configuration to be set

**Returns:** Nothing for success
CCTDasiException for failure

**Includes:** dasi.h

**Mode:** asynchronous

---

## Description

The **SetTDMConfiguration( )** function allows you to send TDM bus configuration settings to a board's clocking agent using the AGENTTDMCONFIGURATION data structure.

***Note:*** For the current release, the AGENTTDMCONFIGURATION data structure's structVersion field should be set to 0.

| Parameter | Description |
|---|---|
| **rTDMConfig** | indicates a reference to the AGENTTDMCONFIGURATION data structure |

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for information about the exceptions that can be thrown by the **SetTDMConfiguration( )** function.

## Example

```
#include <dasi.h>
#include <stdio.h>

using namespace DASI;

// PROTOTYPES
void ChangeConfigurationTest(ICTClockAgent  *pAgent);

//FUNCTIONS
string& boolval (bool val)
{
   static string tr="true";
   static string fa="false";
```

```
   if (val)
      return tr;
   else
      return fa;
}

int main ()
{
   CCTDomain      *pDomain;
   ICTNode        *pNode;
   ICTBoard       *pBoard;
   ICTBOARDLIST   *pBoardList;

   //Synchronize C and C++ IO
   ios::sync_with_stdio();

   cout <<"OA&M API Board Level Information"<< endl;

   //Get Domain Object
   pDomain = new CCTDomain();

   //Initiate and catch DLG_INVALID_SERVER exception
   try
      {
      cout<< "Initiating DLG_INVALID_SERVER exception:"<<endl;
      //Attempt to get node from domain using incorrect default parameter. Exception thrown.
      pNode = pDomain->GetNode("128.0.0.1");
      }
   catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

   //get node from domain using the correct default parameter
   pNode = pDomain->GetNode("127.0.0.1");

   //try and catch DLG_INVALID_BOARDAUID exception using AUID of 999999
   try
      {
      cout<<"Initiating DLG_INVALID_BOARDAUID exception:"<<endl;
      //throw exception due to invalid board AUID of 999999
      pNode->GetBoardByID(999999);
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<<"Error String: "<<e.m_errDesc<<endl;
      }

      try
      {
      //get number of boards installed in node
      cout << "Number of Installed boards: "
           << pNode->GetNumberOfInstalledBoards()
           << endl;

      //get full board list
      pBoardList = pNode->GetAllInstalledBoards();

      //List physical identification attributes of all installed boards
         cout<<endl<<endl;
         cout<<"++++++++++++++++++++++++++++++++++++++++"<<endl;
         ICTBoard  *brdPointers[20];
         int idx = 0;
```

```
         for (i=pBoardList->begin(); i!=pBoardList->end(); i++,idx++)
            {
            pBoard = *i;
            brdPointers[idx] = pBoard;
            cout<<idx<<" : Board at PCI Bus:  "<<pBoard->GetPCIBus()
                <<"PCI Slot:                  "<<pBoard->GetPCISlot()<<endl;
            }
         int chosen_board=-1;
         while (chosen_board<0) || (chosen_board>=idx))
            {
            cout <<endl << "====> Choose Board for test: ";
            char buffer[3];
            fgets(buffer,3,stdin);
            chosen_board = atoi(buffer);
            }
            agent_menu(brdPointers[chosen_board];
      }
      catch (CCTDasiException e)
      {
      cerr<<"Exception: "<<endl;
      cerr<<"Error Value: "<<hex<<e.m_dlgSysError<<endl;
      cerr<< "Error String: "<<e.m_errDesc<<endl;
      exit(1);
      }

   for (i=pBoardList->begin(); i !=pBoardList->end(); i++,idx++) {
    pBoard = *i;
    delete pBoard;
    }

    /* Freeing the board list */
    delete pBoardList;

    /* Freeing the node object */
    delete pNode;

    /* Freeing the domain object */
   delete pDomain;

return;
}

//clocking agent menu function
void agent_menu(ICTBoard *pBoard)
{
ICTCLOCKAGENTLIST *pAgentList;
ICTLCLOCKAGENTLIST::iterator j;

   if (pBoard->GetNumClockingAgents()>0
      {
      pAgentList = pBoard->GetClockingAgents();
      j=pAgentList
      ChangeConfigurationTest(*j);}
      }
   else
      {
      cout <<"No Clocking Agent on board"<<endl;
      }
   /* Freeing the clocking agents in the agent list */
     for (j=pAgentList->begin(); j !=pAgentList->end(); j++) {
          ICTClockAgent   *pAgent;
          // Delete the pAgent
          pAgent = *j;
         delete pAgent;
        }
```

```
        /* Freeing the agent list */
        delete pAgentList;

return;
{

void ChangeConfigurationTest(ICTClockAgent *pAgent)

)

AGENTTDMCONFIGURATION *pCfg;
AGENTTDMCONFIGURATION cfg;

//set up board as CT Bus master that generates netref
cout <<"++++++++Making NetRef, Primary, and CT Bus Line B"<<endl;

cfg.clockMasterRole = CT_PRIMARY;
cfg.clockMasterSourceList[0]=CT_NETREF;
cfg.clockingModel = AUTO_HOLDOVER;
cfg.primaryLines = CT_CLKB;
cfg.isProvidingNetref=true;
cfg.netrefSource = CT_DNI_2;
cfg.netrefClockRate = CT_CLOCKRATE_8KHZ;
cfg.scBusClockRate= CT_CLOCKRATE_8MHZ;
cfg.group1ClockRate= CT_CLOCKRATE_8MHZ;
cfg.group2ClockRate= CT_CLOCKRATE_8MHZ;
cfg.group3ClockRate= CT_CLOCKRATE_8MHZ;
cfg.group4ClockRate= CT_CLOCKRATE_8MHZ;
cfg.busType = CT_BUSTYPE_H100;

pAgent->SetTDMConfiguration(cfg);

return;
}
```

## See Also

- **ICTClockAgent::GetClockAgentID( )**
- **ICTClockAgent::GetTDMCapabilities( )**
- **ICTClockAgent::GetTDMConfiguration( )**

# DlgAdminConsumer::DisableFilters( )

**Name:** void DisableFilters(pfilters, iCount)

**Inputs:** const unsigned long *pfilters
- pointer to the array of filters that will be disabled

int iCount
- size of the filter array

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** synchronous

---

## Description

The **DisableFilters( )** function disables a DlgAdminConsumer object's array of filters.The DlgAdminConsumer object's array of filters must be determined before the object is instantiated (**DlgAdminConsumer::DlgAdminConsumer( )** function).

***Note:*** You can disable individual filters by setting the individual filter elements enable field to DlgEvent_DISABLE. For example, if you wanted to disable the DLGC_EVT_CT_B_LINESBAD event filter, set its element in the array as follows:

```
array_name[].callback=pHandler;
array_name[].clientData= (void*)0;
array_name[].filter=DLGC_EVT_CT_B_LINESBAD;
array_name[].enable=DlgEvent_DISABLE;
```

| Parameter | Description |
|---|---|
| **pfilters** | points to the array of filters that will be disabled |
| **iCount** | indicates the number of elements in the filter array that will be disabled |

## Cautions

The **DisableFilters( )** function only applies to the filters that were passed to the DlgAdminConsumer object during instantiation.

## Errors

None

## Example

This example code refers to an implementation of the **CEventHandlerAdaptor::HandleEvent( )** function.

```
#include <stdio.h>
#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that extends the CEventHandlerAdaptor class class and provides an
implementation of CEventHandlerAdaptor::HandleEvent function*/

#include "clockhandler.h"



//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main (void)

{

     ClockHandler clockHandler;
     DlgEventService::DlgAdminConsumer *pClockConsumer;
     //sync C/C++ Input/Output
     ios::sync_with_stdio();
     cout <<"Monitoring Clock Channel for events:" <<endl;
     pClockConsumer = MonitorClock(&clockHandler);

     while (1)
     {
     sleep(200);
     }
     return 0;

}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{

     DlgEventService::DlgAdminConsumer *pConsumer;

     //create array of filters
     unsigned long myFilterId[2];

     //create array of filters
     DlgEventService::AdminConsumer::FilterCallbackAssoc  myFilterCallBackId[2];

     myFilterCallBackId[0].callback=pHandler;
     myFilterCallBackId[0].clientData= (void*)0;
     myFilterCallBackId[0].filter=DLGC_EVT_CT_A_LINESBAD;
     myFilterCallBackId[0].enable=DlgEvent_ENABLE; /*this field can be set to DlgEvent_DISABLE
         to disable an individual event filter*/

     myFilterCallBackId[1].callback=pHandler;
     myFilterCallBackId[1].clientData= (void*)0;
     myFilterCallBackId[1].filter=DLGC_EVT_CT_B_LINESBAD;
     myFilterCallBackId[1].enable=DlgEvent_ENABLE;
     myFilterCallBackId[2].callback=pHandler;

     //instantiate consumer object using filter array
     pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
     ((wchar_t*)"AdminMonitor"), myFilters,7);

     //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
     pConsumer->StartListening();
```

```
        //Disable the filters
        myFilterId[0] = DLGC_EVT_CT_A_LINESBAD;
        myFilterId[1] = DLGC_EVT_CT_B_LINESBAD;
        DisableFilters (myFilterId, 2);

        //Enable the filters
        myFilterId[0] = DLGC_EVT_CT_A_LINESBAD;
        myFilterId[1] = DLGC_EVT_CT_B_LINESBAD;
        EnableFilters (myFilterId, 2);

        return pConsumer;

}
```

## See Also

- **DlgAdminConsumer::DlgAdminConsumer( )**
- **DlgAdminConsumer::EnableFilters( )**
- **DlgAdminConsumer::getChannelName( )**
- **DlgAdminConsumer::getConsumerName( )**
- **DlgAdminConsumer::StartListening( )**

# DlgAdminConsumer::DlgAdminConsumer( )

**Name:** DlgAdminConsumer (szChannelName, szConsumerName, pFilters, iFilterCnt)

**Inputs:**

| | |
|---|---|
| const char* szChannelName | • channel name that will be monitored |
| const wchar_t* szConsumerName | • name of consumer object |
| AdminConsumer::FilterCallbackAssoc *pFilters | • pointer to an array of filter to event handler object associations |
| int iFilterCnt | • number of elements in the filter to event handler association array |

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** synchronous

---

## Description

The **DlgAdminConsumer( )** function is the DlgAdminConsumer class constructor.It allows you to instantiate a consumer object. Each DlgAdminConsumer object must be associated with one event notification channel. The filter array determines which event handler object is invoked when a particular event occurs on the channel. You must include an element in the array for each event that is to be received by the DlgAdminConsumer object.

| Parameter | Description |
|---|---|
| **szChannelName** | determines which event notification channel the consumer object registers with and monitors for events. |
| **szConsumerName** | indicates the unique name of the consumer object |
| **pFilters** | points to an array of filter to event handler object associations. |
| **iFilterCnt** | indicates the number of elements in the filter to event handler object association array |

## Cautions

None

## Errors

None

## Example

The following example program instantiates a DlgAdminConsumer object that monitors the CLOCK_EVENT_CHANNEL for incoming events. The DlgAdminConsumer object is named

"ClockMonitor" and contains a pointer to an array of seven event filters (one for each event that is carried on the CLOCK_EVENT_CHANNEL):

```
#include <stdio.h>

#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that defines class to override CEventHandlerAdaptor::HandleEvent
method*/
#include "clockhandler.h"

//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main ()
{
   ClockHandler clockHandler;
   DlgEventService::DlgAdminConsumer *pClockConsumer;

   //sync C/C++ Input/Output
   ios::sync_with_stdio();

   cout <<"Monitoring Clock Channel for events:" <<endl;
   pClockConsumer = MonitorClock(&clockHandler);

   while (1)
   {
      sleep(200);
   }
   return 0;
}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{
   DlgEventService::DlgAdminConsumer *pConsumer;

   //create array of filters
   DlgEventService::AdminConsumer::FilterCallbackAssoc myFilters[7];

   myFilters[0].callback=pHandler; //event handler object that is invoked
   myFilters[0].clientData= (void*)0;
             /*void pointer that is passed to the application during callback*/
   myFilters[0].filter=DLGC_EVT_CT_A_LINESBAD; //event name (i.e. msgId)
   myFilters[0].enable=DlgEvent_ENABLE; //status of filter

   myFilters[1].callback=pHandler;
   myFilters[1].clientData= (void*)0;
   myFilters[1].filter=DLGC_EVT_CT_B_LINESBAD;
   myFilters[1].enable=DlgEvent_ENABLE;

   myFilters[2].callback=pHandler;
   myFilters[2].clientData= (void*)0;
   myFilters[2].filter=DLGC_EVT_SCBUS_COMPAT_LINESBAD;
   myFilters[2].enable=DlgEvent_ENABLE;

   myFilters[3].callback=pHandler;
   myFilters[3].clientData= (void*)0;
   myFilters[3].filter=DLGC_EVT_MVIP_COMPAT_LINESBAD;
   myFilters[3].enable=DlgEvent_ENABLE;
```

```
    myFilters[4].callback=pHandler;
    myFilters[4].clientData= (void*)0;
    myFilters[4].filter=DLGC_EVT_NETREF1_LINEBAD;
    myFilters[4].enable=DlgEvent_ENABLE;

    myFilters[5].callback=pHandler;
    myFilters[5].clientData= (void*)0;
    myFilters[5].filter=DLGC_EVT_NETREF2_LINEBAD;
    myFilters[5].enable=DlgEvent_ENABLE;

    myFilters[6].callback=pHandler;
    myFilters[6].clientData= (void*)0;
    myFilters[6].filter=DLGC_EVT_LOSS_MASTER_SOURCE_INVALID;
    myFilters[6].enable=DlgEvent_ENABLE;

    //instantiate consumer object using filter array
    pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
       ((wchar_t*)"ClockMonitor"), myFilters,7);

    //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
    pConsumer->StartListening();

    return pConsumer;
}
```

## See Also

- **DlgAdminConsumer::DisableFilters( )**
- **DlgAdminConsumer::EnableFilters( )**
- **DlgAdminConsumer::getChannelName( )**
- **DlgAdminConsumer::getConsumerName( )**
- **DlgAdminConsumer::StartListening( )**

# DlgAdminConsumer::EnableFilters( )

**Name:** void EnableFilters(pFilters, iCount)

**Inputs:** const unsigned long *pFilters — pointer to an array of filters to be enabled

int iCount — size of the filter array to be enabled

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** synchronous

---

## Description

The **EnableFilters( )** function enables a DlgAdminConsumer object's array of filters. The DlgAdminConsumer object's array of filters is determined when the object is instantiated (**DlgAdminConsumer::DlgAdminConsumer( )** function).

| Parameter | Description |
|---|---|
| **pFilters** | points to the array of filters that will be enabled |
| **iCount** | indicates the size of the filter array that will be enabled |

## Cautions

The **EnableFilters( )** function only applies to filters that were passed to the consumer object during instantiation.

## Errors

None

## Example

```
#include <stdio.h>
#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that extends the CEventHandlerAdaptor class class and provides an
implementation of CEventHandlerAdaptor::HandleEvent function*/

#include "clockhandler.h"



//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main (void)
```

```
{

     ClockHandler clockHandler;
     DlgEventService::DlgAdminConsumer *pClockConsumer;
     //sync C/C++ Input/Output
     ios::sync_with_stdio();
     cout <<"Monitoring Clock Channel for events:" <<endl;
     pClockConsumer = MonitorClock(&clockHandler);

     while (1)
     {
     sleep(200);
     }
     return 0;

}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{

     DlgEventService::DlgAdminConsumer *pConsumer;

     //create array of filters
     unsigned long myFilterId[2];

     //create array of filters
     DlgEventService::AdminConsumer::FilterCallbackAssoc  myFilterCallBackId[2];

     myFilterCallBackId[0].callback=pHandler;
     myFilterCallBackId[0].clientData= (void*)0;
     myFilterCallBackId[0].filter=DLGC_EVT_CT_A_LINESBAD;
     myFilterCallBackId[0].enable=DlgEvent_ENABLE; /*this field can be set to DlgEvent_DISABLE
         to disable an individual event filter*/

     myFilterCallBackId[1].callback=pHandler;
     myFilterCallBackId[1].clientData= (void*)0;
     myFilterCallBackId[1].filter=DLGC_EVT_CT_B_LINESBAD;
     myFilterCallBackId[1].enable=DlgEvent_ENABLE;
     myFilterCallBackId[2].callback=pHandler;

     //instantiate consumer object using filter array
     pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
       ((wchar_t*)"AdminMonitor"),myFilters,7);

     //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
     pConsumer->StartListening();

     //Disable the filters
     myFilterId[0] = DLGC_EVT_CT_A_LINESBAD;
     myFilterId[1] = DLGC_EVT_CT_B_LINESBAD;
     DisableFilters (myFilterId, 2);

     //Enable the filters
     myFilterId[0] = DLGC_EVT_CT_A_LINESBAD;
     myFilterId[1] = DLGC_EVT_CT_B_LINESBAD;
     EnableFilters (myFilterId, 2);

     return pConsumer;

}
```

## See Also

- **DlgAdminConsumer::DisableFilters( )**
- **DlgAdminConsumer::DlgAdminConsumer( )**

- **DlgAdminConsumer::getChannelName( )**
- **DlgAdminConsumer::getConsumerName( )**
- **DlgAdminConsumer::StartListening( )**

# DlgAdminConsumer::getChannelName( )

**Name:** const char* getChannelName(void)

**Returns:** pointer to a constant character string for success
NULL for failure

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** synchronous

---

## Description

The **getChannelName( )** function returns the channel name that a DlgAdminConsumer object monitors for incoming events. The DlgAdminConsumer object's associated channel name is determined by the **szChannelName** parameter when the object is instantiated (**DlgAdminConsumer::DlgAdminConsumer( )** function).

Refer to Chapter 3, "Events" for a complete list of event notification framework channels.

## Cautions

None

## Errors

None

## Example

```
#include <stdio.h>

#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that defines class to override
 *CEventHandlerAdaptor::HandleEvent method
 */
#include "clockhandler.h"

//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main (void)
{
   ClockHandler clockHandler;
   DlgEventService::DlgAdminConsumer *pClockConsumer;

   //sync C/C++ Input/Output
   ios::sync_with_stdio();
```

```
   cout <<"Monitoring Clock Channel for events:" <<endl;
   pClockConsumer = MonitorClock(&clockHandler);

   while (1)
   {
      sleep(200);
   }
   return 0;
}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{
   DlgEventService::DlgAdminConsumer *pConsumer;

   //create array of filters
   DlgEventService::AdminConsumer::FilterCallbackAssoc myFilters[7];

   myFilters[0].callback=pHandler;
   myFilters[0].clientData= (void*)0;
   myFilters[0].filter=DLGC_EVT_CT_A_LINESBAD;
   myFilters[0].enable=DlgEvent_ENABLE;

   myFilters[1].callback=pHandler;
   myFilters[1].clientData= (void*)0;
   myFilters[1].filter=DLGC_EVT_CT_B_LINESBAD;
   myFilters[1].enable=DlgEvent_ENABLE;

   myFilters[2].callback=pHandler;
   myFilters[2].clientData= (void*)0;
   myFilters[2].filter=DLGC_EVT_SCBUS_COMPAT_LINESBAD;
   myFilters[2].enable=DlgEvent_ENABLE;

   myFilters[3].callback=pHandler;
   myFilters[3].clientData= (void*)0;
   myFilters[3].filter=DLGC_EVT_MVIP_COMPAT_LINESBAD;
   myFilters[3].enable=DlgEvent_ENABLE;

   myFilters[4].callback=pHandler;
   myFilters[4].clientData= (void*)0;
   myFilters[4].filter=DLGC_EVT_NETREF1_LINEBAD;
   myFilters[4].enable=DlgEvent_ENABLE;

   myFilters[5].callback=pHandler;
   myFilters[5].clientData= (void*)0;
   myFilters[5].filter=DLGC_EVT_NETREF2_LINEBAD;
   myFilters[5].enable=DlgEvent_ENABLE;

   myFilters[6].callback=pHandler;
   myFilters[6].clientData= (void*)0;
   myFilters[6].filter=DLGC_EVT_LOSS_MASTER_SOURCE_INVALID;
   myFilters[6].enable=DlgEvent_ENABLE;

   //instantiate consumer object using filter array
   pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
       ((wchar_t*)"AdminMonitor"),myFilters,7);

   //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
   pConsumer->StartListening();

   //get associated channel name
   pConsumer->getChannelName()

   return pConsumer;
}
```

## See Also

- DlgAdminConsumer::DisableFilters( )
- DlgAdminConsumer::DlgAdminConsumer( )
- DlgAdminConsumer::EnableFilters( )
- DlgAdminConsumer::getConsumerName( )
- DlgAdminConsumer::StartListening( )

# DlgAdminConsumer::getConsumerName( )

**Name:** const wchar_t* getConsumerName(void)

**Returns:** pointer to a constant character string for success
NULL for failure

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** synchronous

---

## Description

The **getConsumerlName( )** function returns the name of the DlgAdminConsumer object. The DlgAdminConsumer object name is determined by the **szConsumerName** parameter when theDlgAdminConsumer object is instantiated (**DlgAdminConsumer::DlgAdminConsumer( )** function).

## Cautions

None

## Errors

None

## Example

```
#include <stdio.h>

#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that defines class to override
 *CEventHandlerAdaptor::HandleEvent method
 */
#include "clockhandler.h"

//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main (void)
{
   ClockHandler clockHandler;
   DlgEventService::DlgAdminConsumer *pClockConsumer;

   //sync C/C++ Input/Output
   ios::sync_with_stdio();
```

```
   cout <<"Monitoring Clock Channel for events:" <<endl;
   pClockConsumer = MonitorClock(&clockHandler);

   while (1)
   {
      sleep(200);
   }
   return 0;
}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{
   DlgEventService::DlgAdminConsumer *pConsumer;

   //create array of filters
   DlgEventService::AdminConsumer::FilterCallbackAssoc myFilters[7];

   myFilters[0].callback=pHandler;
   myFilters[0].clientData= (void*)0;
   myFilters[0].filter=DLGC_EVT_CT_A_LINESBAD;
   myFilters[0].enable=DlgEvent_ENABLE;

   myFilters[1].callback=pHandler;
   myFilters[1].clientData= (void*)0;
   myFilters[1].filter=DLGC_EVT_CT_B_LINESBAD;
   myFilters[1].enable=DlgEvent_ENABLE;

   myFilters[2].callback=pHandler;
   myFilters[2].clientData= (void*)0;
   myFilters[2].filter=DLGC_EVT_SCBUS_COMPAT_LINESBAD;
   myFilters[2].enable=DlgEvent_ENABLE;

   myFilters[3].callback=pHandler;
   myFilters[3].clientData= (void*)0;
   myFilters[3].filter=DLGC_EVT_MVIP_COMPAT_LINESBAD;
   myFilters[3].enable=DlgEvent_ENABLE;

   myFilters[4].callback=pHandler;
   myFilters[4].clientData= (void*)0;
   myFilters[4].filter=DLGC_EVT_NETREF1_LINEBAD;
   myFilters[4].enable=DlgEvent_ENABLE;

   myFilters[5].callback=pHandler;
   myFilters[5].clientData= (void*)0;
   myFilters[5].filter=DLGC_EVT_NETREF2_LINEBAD;
   myFilters[5].enable=DlgEvent_ENABLE;

   myFilters[6].callback=pHandler;
   myFilters[6].clientData= (void*)0;
   myFilters[6].filter=DLGC_EVT_LOSS_MASTER_SOURCE_INVALID;
   myFilters[6].enable=DlgEvent_ENABLE;

   //instantiate consumer object using filter array
   pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
       ((wchar_t*)"AdminMonitor"),myFilters,7);

   //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
   pConsumer->StartListening();

   //get consumer name
   pConsumer->getConsumerName()

   return pConsumer;
}
```

## See Also

- **DlgAdminConsumer::DisableFilters( )**
- **DlgAdminConsumer::DlgAdminConsumer( )**
- **DlgAdminConsumer::EnableFilters( )**
- **DlgAdminConsumer::getChannelName( )**
- **DlgAdminConsumer::StartListening( )**

# DlgAdminConsumer::StartListening( )

**Name:** bool StartListening(void)

**Returns:** true for success
false for failure

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** asynchronous

---

## Description

The **StartListening**( ) function allows the DlgAdminConsumer object to begin monitoring its associated event notification channel for incoming events. The DlgAdminConsumer object's associated event notification channel is determined when the object is instantiated (**DlgAdminConsumer::DlgAdminConsumer( )** function).

## Cautions

None

## Errors

None

## Example

```
#include <stdio.h>

#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

/*user defined header file that defines class to override
 *CEventHandlerAdaptor::HandleEvent method
 */
#include "clockhandler.h"

//function prototypes
DlgEventService::DlgAdminConsumer * MonitorClock (ClockHandler *pHandler);

int main (void)
{
   ClockHandler clockHandler;
   DlgEventService::DlgAdminConsumer *pClockConsumer;

   //sync C/C++ Input/Output
   ios::sync_with_stdio();
```

```
   cout <<"Monitoring Clock Channel for events:" <<endl;
   pClockConsumer = MonitorClock(&clockHandler);

   while (1)
   {
      sleep(200);
   }
   return 0;
}

DlgEventService::DlgAdminConsumer * MonitorClock(ClockHandler *pHandler)
{
   DlgEventService::DlgAdminConsumer *pConsumer;

   //create array of filters
   DlgEventService::AdminConsumer::FilterCallbackAssoc myFilters[7];

   myFilters[0].callback=pHandler;
   myFilters[0].clientData= (void*)0;
   myFilters[0].filter=DLGC_EVT_CT_A_LINESBAD;
   myFilters[0].enable=DlgEvent_ENABLE;

   myFilters[1].callback=pHandler;
   myFilters[1].clientData= (void*)0;
   myFilters[1].filter=DLGC_EVT_CT_B_LINESBAD;
   myFilters[1].enable=DlgEvent_ENABLE;

   myFilters[2].callback=pHandler;
   myFilters[2].clientData= (void*)0;
   myFilters[2].filter=DLGC_EVT_SCBUS_COMPAT_LINESBAD;
   myFilters[2].enable=DlgEvent_ENABLE;

   myFilters[3].callback=pHandler;
   myFilters[3].clientData= (void*)0;
   myFilters[3].filter=DLGC_EVT_MVIP_COMPAT_LINESBAD;
   myFilters[3].enable=DlgEvent_ENABLE;

   myFilters[4].callback=pHandler;
   myFilters[4].clientData= (void*)0;
   myFilters[4].filter=DLGC_EVT_NETREF1_LINEBAD;
   myFilters[4].enable=DlgEvent_ENABLE;

   myFilters[5].callback=pHandler;
   myFilters[5].clientData= (void*)0;
   myFilters[5].filter=DLGC_EVT_NETREF2_LINEBAD;
   myFilters[5].enable=DlgEvent_ENABLE;

   myFilters[6].callback=pHandler;
   myFilters[6].clientData= (void*)0;
   myFilters[6].filter=DLGC_EVT_LOSS_MASTER_SOURCE_INVALID;
   myFilters[6].enable=DlgEvent_ENABLE;

   //instantiate consumer object using filter array
   pConsumer=new DlgEventService::DlgAdminConsumer(CLOCK_EVENT_CHANNEL,
       ((wchar_t*)"AdminMonitor"), myFilters,7);

   //begin monitoring CLOCK_EVENT_CHANNEL for incoming events
   pConsumer->StartListening();

   return pConsumer;
}
```

## See Also

- **DlgAdminConsumer::DisableFilters( )**

- **DlgAdminConsumer::DlgAdminConsumer( )**
- **DlgAdminConsumer::EnableFilters( )**
- **DlgAdminConsumer::getChannelName( )**
- **DlgAdminConsumer::getConsumerName( )**

# CEventHandlerAdaptor::HandleEvent( )

**Name:** int HandleEvent(evMsg, clientData)

**Inputs:**

| | |
|---|---|
| const DlgEventMsgTypePtr evMsg | • pointer to the event message sent by supplier object |
| ClientDataType clientData | • pointer value returned back to the application in the callback object |

**Returns:** 0 for success
non-zero for failure

**Includes:** dlgadminconsumer.h
dlgadminmsg.h
dlgcevents.h
dlgeventproxydef.h

**Mode:** asynchronous

---

## Description

The application must provide an implementation of the **HandleEvent( )** virtual function in order to receive events from the event notification framework. The **HandleEvent( )** function is invoked when a DlgAdminConsumer object receives an event.

| Parameter | Description |
|---|---|
| **evMsg** | points to the actual event (DlgEventMsgType data structure) that is sent by the supplier object |
| **clientData** | points to a value that is returned back to the application when the event handler is invoked |

## Cautions

None

## Errors

Refer to Chapter 6, "Error Codes" for a list of error codes that can be returned by the **HandleEvent( )** function.

## Example

The following example provides an implementation of the **HandleEvent( )** function for events received on the CLOCK_EVENT_CHANNEL:

```
#include <stdio.h>

#ifndef DLG_WIN32_OS
#include <unistd.h>
#include <wchar.h>
#endif
```

```
#include "dlgeventproxydef.h"
#include "dlgcevents.h"
#include "dlgadminconsumer.h"
#include "dlgadminmsg.h"

// Messages
#include "dlgcevents.h"

int ClockHandler::HandleEvent(const DlgEventMsgTypePtr pMsg, ClientDataType clientData)
{

   cout << "Clocking Channel Event ==> ";
   switch(pMsg->msgId) //event handler switches based on the contents of the msgId
   {
   case DLGC_EVT_CT_A_LINESBAD:
      cout << "DLGC_EVT_CT_A_LINESBAD"<<endl;
      break;
   case DLGC_EVT_CT_B_LINESBAD:
      cout << "DLGC_EVT_CT_B_LINESBAD"<<endl;
      break;
   case DLGC_EVT_SCBUS_COMPAT_LINESBAD:
      cout << "DLGC_EVT_SCBUS_COMPAT_LINESBAD"<<endl;
      break;
   case DLGC_EVT_MVIP_COMPAT_LINESBAD:
      cout << "DLGC_EVT_MVIP_COMPAT_LINESBAD"<<endl;
      break;
   case DLGC_EVT_NETREF1_LINEBAD:
      cout << "DLGC_EVT_NETREF1_LINEBAD"<<endl;
      break;
   case DLGC_EVT_NETREF2_LINEBAD:
      cout << "DLGC_EVT_NETREF2_LINEBAD"<<endl;
      break;
   case DLGC_EVT_LOSS_MASTER_SOURCE_INVALID:
      cout << "DLGC_EVT_LOSS_MASTER_SOURCE_INVALID"<<endl;
      break;
   }

   cout << "  Received Supplier IP   = " << pMsg->node << endl;
   cout << "  Received Event Id      = " << pMsg->msgId << endl ;
   cout << "  Received Msg Size      = " << pMsg->payLoadLen << endl;

   ClockingFaultMsg* pPayload = (ClockingFaultMsg *)pMsg->pPayLoad;
   cout << "  AUID             = " << pPayload->auid << endl;

   // Handled
   return 0;
}
```

## See Also

None

# Events 3

This chapter provides information about events that are transmitted via the various channels of the OA&M API event notification framework. Topics include:



***Notes:*** ***1.*** Refer to Chapter 5, "Data Structures" for information about the data structures and payload formats used by events that are transmitted via the OA&M event notification framework.

***2.*** Refer to the *OA&M API for Linux Programming Guide* for information about enabling your application to receive and handle OA&M events.

***3.*** DM3 architecture boards generate events on all channels. Springware architecture boards generate NETWORK_ALARM_CHANNEL events only.

## 3.1 ADMIN_CHANNEL Events

The following events are transmitted on the ADMIN_CHANNEL:

DLGC_EVT_BLADE_ABOUT_TO_REMOVE
: Generated when the extraction handles on a cPCI board are unlocked. This event is generated by cPCI boards only.

DLGC_EVT_BLADE_ABOUT_TO_START
: Generated when the process for starting the board begins.

DLGC_EVT_BLADE_ABOUT_TO_STOP
: Generated when the process for stopping the board begins.

DLGC_EVT_BLADE_DETECTED
: Indicates that a board has been inserted and detected by the Intel® Dialogic® System Software. This event is generated by cPCI boards only

DLGC_EVT_BLADE_REMOVED
: Generated when a board has been removed from the system. This event is generated by cPCI boards only

DLGC_EVT_BLADE_STARTED
: Indicates that a board has been individually started.

DLGC_EVT_BLADE_STOPPED
: Occurs when a board has been individually stopped.

Table 3-1. ADMIN_CHANNEL Event Payloads

| Event | Payload Type | Payload Size |
|---|---|---|
| DLGC_EVT_BLADE_ABOUT_TO_REMOVE | AUID | sizeof(AUID) = 4 bytes |
| DLGC_EVT_BLADE_ABOUT_TO_START | DevAdminEventMsg | sizeof(DevAdminEventMsg) |
| DLGC_EVT_BLADE_ABOUT_TO_STOP | DevAdminEventMsg | sizeof(DevAdminEventMsg) |
| DLGC_EVT_BLADE_DETECTED | AUID | sizeof(AUID) = 4 bytes |
| DLGC_EVT_BLADE_REMOVED | AUID | sizeof(AUID) = 4 bytes |
| DLGC_EVT_BLADE_STARTED | DevAdminEventMsg | sizeof(DevAdminEventMsg) |
| DLGC_EVT_BLADE_STOPPED | DevAdminEventMsg | sizeof(DevAdminEventMsg) |

## 3.2 CLOCK_EVENT_CHANNEL Events

The following events are transmitted on the CLOCK_EVENT_CHANNEL:

DLGC_EVT_CT_A_LINESBAD
Occurs if the signal on the CT Bus A Line fails.

DLGC_EVT_CT_B_LINESBAD
Occurs if the signal on the CT Bus B Line fails.

DLGC_EVT_LOSS_MASTER_SOURCE_INVALID
Signals that the source used by the primary master board to drive the primary line has failed. The primary master board can use its own internal oscillator or a CT Bus Network Reference line as its clock source.

DLGC_EVT_MVIP_COMPAT_LINESBAD
Generated if the MVIP compatibility line fails.

***Note:*** The MVIP bus is currently not supported.

DLGC_EVT_NETREF1_LINEBAD
Indicates that the signal on the CT Bus NetRef 1 line has failed.

DLGC_EVT_SCBUS_COMPAT_LINESBAD
Occurs if the SCbus compatibility line fails.

## 3.3 FAULT_CHANNEL Events

The following events are transmitted on the FAULT_CHANNEL:

DLGC_EVT_CP_FAILURE
Generated when a Control Processor failure has occurred on a board.

DLGC_EVT_SP_FAILURE
Generated when a Signal Processor failure has occurred on a board.

## 3.4 NETWORK_ALARM_CHANNEL Events

The following events are transmitted on the NETWORK_ALARM_CHANNEL:

DLGC_EVT_EXTERNAL_ALARM_RED
: Occurs when the device at the receiving end of a T-1 or E-1 line has detected a loss of signal or frame alignment in the incoming data.

DLGC_EVT_EXTERNAL_ALARM_RED_CLEAR
: Signifies that the loss of signal/frame alignment which caused a red alarm has recovered and the red alarm condition has been cleared.

DLGC_EVT_EXTERNAL_ALARM_YELLOW
: Generated by the device at the receiving end of a T-1 or E-1 line. The alarm is sent to the device at the transmitting (remote) end to signify that a red alarm has been declared at the receiving (local) end of the network.

DLGC_EVT_EXTERNAL_ALARM_YELLOW_CLEAR
: Indicates that a yellow alarm is no longer being sent to the transmitting (remote) end of the network.

DLGC_EVT_EXTERNAL_LOSS_OF_SIGNAL
: Indicates that a signal is not being detected on the incoming T-1 or E-1 line.

DLGC_EVT_EXTERNAL_LOSS_OF_SIGNAL_CLEAR
: Occurs when a signal has been detected on the incoming T-1 or E-1 line and the loss of signal alarm has been cleared.

# Data Types 4

This chapter contains reference information about the various data types used by the OA&M API.

The following list contains data types that are defined in the OA&M API:

AUID
: Long integer data type that indicates the Addressable Unit Identifier of a system component. The Intel® Dialogic® System Software assigns a unique AUID to each system component with which communications can be initiated. In the context of the OA&M API, each board and each clocking agent are assigned AUIDs.

ClientDataType
: A `void*` data type that is determined when a DlgAdminConsumer object's filter array is set. This value is sent to the event handler when an event is received and returned back to the client in the callback object.

DlgFilterType
: Unsigned long integer data type that identifies the event name for events that are carried on the event notification framework.

IpAddressStringType
: A string (`char*` data type) that contains the node IP Address of the supplier object.

PayloadDataType
: Unsigned character data type for the serialized message that is encoded by the supplier object and must be decoded by the consumer object via typecasting.

SupplierNameType
: A string (`wchar_t*` data type) that contains the name of the supplier object that generated the event.

  ***Note:*** The SupplierNameType data type is for informational purposes only and is subject to change in future Intel Dialogic System Software releases.

# Data Structures 5

This chapter includes reference information about the data structures that are used by the OA&M API. Information is provided for the following data structures:

# AGENTTDMCAPABILITIES

```
typedef struct
{
      long structVersion;    /*set to 0 for current release*/
      bool canH100Master;
      bool canH100Slave;
      bool canH110Master;
      bool canH110Slave;
      bool canScbusMaster;
      bool canScbusSlave;
      bool canMVIPMaster;
      bool canMVIPSlave;
      bool canScbus2Mhz;
      bool canScbus4Mhz;
      bool canScbus8Mhz;
      bool canProvideNetrefAt8Khz;
      bool canProvideNetrefAt1536Mhz;
      bool canProvideNetrefAt1544Mhz;
      bool canProvideNetrefAt2048Mhz;
      bool canDeriveNetrefAt8Khz;
      bool canDeriveNetrefAt1536Mhz;
      bool canDeriveNetrefAt1544Mhz;
      bool canDeriveNetrefAt2048Mhz;
      bool canProvideNetref2At8Khz;
      bool canProvideNetref2At1536Mhz;
      bool canProvideNetref2At1544Mhz;
      bool canProvideNetref2At2048Mhz;
      bool canDeriveNetref2At8Khz;
      bool canDeriveNetref2At1536Mhz;
      bool canDeriveNetref2At1544Mhz;
      bool canDeriveNetref2At2048Mhz;
      bool canSilenceScbusCompatLines;
      bool canSilenceMvipCompatLines;
      bool canDeriveNetrefFromOsc;           /*can derive reference from internal oscillator*/
      bool canTerminate;                     /*can terminate the bus*/
      bool canDynamicConfigure;              /*can be configured using the OA&M API*/
      bool canUseLocalDNIAsMasterSource;     /*can use the local telephony network frontends
                                               as a clock master source*/
      bool isAlawCapable;
      bool isMulawCapable;
      bool isConversionCapable;
      short maxClockMasterSourceListSize;
} AGENTTDMCAPABILITIES,  *LPAGENTTDMCAPABILITIES;
```

## Description

The AGENTTDMCAPABILITIES data structure defines the TDM bus capabilities of a clocking agent. The TDM bus capabilities of a clocking agent can be retrieved with the **ICTClockAgent::GetTDMCapabilities( )** function.

## Field Descriptions

All fields in the structure, except for maxClockMasterSourceListSize, are boolean variables. The value of *true* indicates that the clocking agent supports the capability, the value of *false* indicates that the capability is not available. The maxClockMasterSourceListSize is an integer that indicates the maximum number of clock master fallback sources that can be defined for the clocking agent.

# AGENTTDMCONFIGURATION

```
typedef struct
{
      long                structVersion;   /*set to 0 for current release*/
      bool                onTelephonyBus;
      short               busType;
      CLOCKRATE           scBusClockRate;
      CLOCKRATE           group1ClockRate;
      CLOCKRATE           group2ClockRate;
      CLOCKRATE           group3ClockRate;
      CLOCKRATE           group4ClockRate;
      bool                MVIPCompatLinesActive;
      bool                SCBusCompatLinesActive;
      short               clockMasterRole;
      long                clockMasterSource;
      std::vector <long>  clockMasterSourceList[16];
      CLOCKRATE           clockMasterRate;
      short               clockingMode1;
      short               primaryLines;
      bool                isProvidingNetref;
      long                netrefSource;
      NETREFCLOCKRATES    netrefClockRate;
      bool                isProvidingNetref2;
      long                netref2Source;
      NETREFCLOCKRATES    netref2ClockRate;
      bool                termination;

} AGENTTDMCONFIGURATION, *LPAGENTTDMCONFIGURATION;
```

## Description

The AGENTTDMCONFIGURATION data structure defines the TDM bus configuration of a board's clocking agent. The data structure can be used to query the current configuration of a clocking agent (**ICTClockAgent::GetTDMConfiguration( )** function) or to set the configuration of a clocking agent (**ICTClockAgent::SetTDMConfiguration ( )** function).

## Field Descriptions

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

structVersion
: Specifies the version number of the structure. This is set to 0 for the current release.

onTelephonyBus
: Indicates whether or not the clocking agent is connected to the TDM bus. Possible values are as follows:
  - True – put on bus
  - False – do not put on bus

busType
: Indicates the TDM bus type. Possible values are as follows:
  - CT_BUSTYPE_MVIP

  ***Note:*** The MVIP bus is not supported.
  - CT_BUSTYPE_SCBUS

  ***Note:*** The SCBus is not supported.

- CT_BUSTYPE_H100
- CT_BUSTYPE_H110 (this value is supported for cPCI boards only)

ScBusClockRate
The SCBus is not supported.

group1ClockRate
If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field determines the clock rate for the first group of CT Bus lines. Possible values are as follows:

- CT_CLOCKRATE_NONE
- CT_CLOCKRATE_2MHZ
- CT_CLOCKRATE_4MHZ
- CT_CLOCKRATE_8MHZ

group2ClockRate
If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field determines the clock rate for the second group of CT Bus lines. Possible values are as follows:

- CT_CLOCKRATE_NONE
- CT_CLOCKRATE_2MHZ
- CT_CLOCKRATE_4MHZ
- CT_CLOCKRATE_8MHZ

group3ClockRate
If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field determines the clock rate for the third group of CT Bus lines. Possible values are as follows:

- CT_CLOCKRATE_NONE
- CT_CLOCKRATE_2MHZ
- CT_CLOCKRATE_4MHZ
- CT_CLOCKRATE_8MHZ

group4ClockRate
If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field determines the clock rate for the fourth group of CT Bus lines. Possible values are as follows:

- CT_CLOCKRATE_NONE
- CT_CLOCKRATE_2MHZ
- CT_CLOCKRATE_4MHZ
- CT_CLOCKRATE_8MHZ

MVIPCompatLinesActive
If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field indicates whether or not the MVIP compatibility lines are active. Possible values are as follows:

- True – activate MVIP compatibility lines
- False – do not activate MVIP compatibility lines

***Note:*** The MVIP bus is not supported by the current system release; so the MVIPCompatLinesActive value is not supported.

SCBusCompatLinesActive

If the busType field is set to CT_BUSTYPE_H100 or CT_BUSTYPE_H110, this field indicates whether or not the SCbus compatibility lines are active. Possible values are as follows:

- True – activate SCbus compatibility lines
- False – do not activate SCbus compatibility lines

clockMasterRole

Indicates the CT Bus clocking status assigned to the clocking agent. Possible values are as follows:

- CT_PRIMARY – The clock agent resides on the board that provides the clocking signal to the CT Bus primary line.
- CT_SECONDARY – The clock agent resides on the board that provides the clocking signal to the CT Bus secondary line.
- CT_SLAVE – The clock agent resides on a board that derives its clocking signal from the CT bus primary line.
- CT_REFERENCE – The clock agent resides on a board that provides the clocking signal to a CT Bus network reference line.

clockMasterSource

Indicates the clocking source used by the clock master. Possible values are as follows:

- CT_DISABLED
- CT_NETREF
- CT_NETREF2
- CT_INT_CLK

clockMasterSourceList

Provides a fallback list of onboard Digital Network Interfaces (DNI) where the board will acquire the clock from when it is assigned to provide the clock on one of the CT bus lines (Primary, Secondary, Network Reference). Some boards are capable of having more than one clock source and will automatically fallback to the next source if the current clock source fails. Possible values are as follows:

- CT_DNI_1
- CT_DNI_2
- CT_DNI_3
- CT_DNI_4
- CT_DNI_5
- CT_DNI_6
- CT_DNI_7
- CT_DNI_8
- CT_DNI_9
- CT_DNI_10
- CT_DNI_11
- CT_DNI_12
- CT_DNI_13
- CT_DNI_14
- CT_DNI_15

- CT_DNI_16
- CT_MAXDNI

clockMasterRate

Determines the clock rate of the incoming master clock (Primary, Secondary or NetRef). Possible values are as follows:

- CT_CLOCKRATE_NONE
- CT_CLOCKRATE_1_536MHZ
- CT_CLOCRATE_1_544MHZ
- CT_CLOCKRATE_2_048MHZ

clockingModel

Indicates the mode of the clocking agent. Possible values are as follows:

- CT_NORMAL
- CT_HOLDOVER
- CT_FREE_RUN
- UNKNOWN
- CT_AUTO_HOLDOVER
- CT_AUTO_FREE_RUN

***Note:*** Only the CT_NORMAL or CT_AUTO_HOLDOVER modes can be set by the application. The clocking agent automatically switches to CT_HOLDOVER, CT_FREE_RUN or CT_AUTO_FREE_RUN modes based on internal processes, without application intervention.

***Note:*** If the **ICTClockAgent::GetTDMConfiguration( )** function is called immediately after the clockingModel field is switched into CT_HOLDOVER or CT_AUTO_HOLDOVER mode, the field may still display its previous value. This is due to a brief delay while the firmware waits for the clock to synchronize. Your application should account for this delay before invoking the **ICTClockAgent::GetTDMConfiguration( )** function.

primaryLines

Specifies which CT Bus line is the primary line. Possible values are as follows:

- CT_CLKA
- CT_CLKB
- CT_DIS

isProvidingNetref

Determines whether or not the clocking agent is providing a clocking signal to the CT bus network reference (NetRef) line. Possible values are as follows:

- True – Provide NetRef on the bus
- False – Not providing NetRef on the bus

netrefSource

If the isProvidingNetref field is set to True, this field determines the source of the NetRef clocking signal.

netrefClockRate

Determines the clock rate of the network reference line (NetRef) if the busType field is set to either CT_BUSTYPE_H100 or CT_BUSTYPE_H110. Possible values are as follows:

- CT_CLOCKRATE_8KHZ
- CT_CLOCKRATE_1_536MHZ
- CT_CLOCKRATE_1_544MHZ
- CT_CLOCKRATE_2_048MHZ

isProvidingNetref2

***Note:*** The isProvidingNetref2 field is not supported.

netref2Source

***Note:*** The netref2Source field is not supported.

netref2ClockRate

***Note:*** The netref2ClockRate field is not supported.

termination

Specifies whether or not the clocking agent terminates the TDM bus. Possible values are as follows:

- True – terminate the bus
- False – do not terminate the bus

# ClockingFaultMsg

```
typedef struct ClockingFaultMsgT
{
     AUID   auid;
     short  nPhysicalBusNumber;
     char   szDescription[MAX_EVENT_DESCRIPTION];
} ClockingFaultMsg, *ClockingFaultMsgPtr;
```

## Description

The ClockingFaultMsg data structure defines the payload format of events that are carried on the CLOCK_EVENT_CHANNEL.

Refer to Chapter 3, "Events" for a list of event channels.

## Field Descriptions

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

auid
: AUID of the clocking agent that generated the event

nPhysicalBusNumber
: physical bus number that the board is on

szDescription
: ASCIIZ, NULL-terminated string that contains a description of the clocking fault

# CTPLATFORMVERSIONINFO

```
typedef struct_tagCTPlatformVersionInfo
{
      long   lSystemReleaseMajorReleaseNumber;
      long   lSystemReleaseMinorReleaseNumber;
      long   lSystemReleaseSubMinorReleaseNumber;
      long   lSystemReleaseBuildNumber;

      long   lSystemServicePackNumber;
      long   lSystemServicePackBuildNumber;

      long   lSystemFeaturePackNumber;
      long   lSystemFeaturePackBuildNumber;

      long   lServiceUpdateNumber;
      long   lServiceUpdateBuildNumber;
      long   lFeatureReleaseNumber;
      long   lFeatureReleaseBuildNumber;

} CTPLATFORMVERSIONINFO;
```

## Description

The CTPLATFORMVERSIONINFO data structure provides information about the Intel Dialogic System Software that is installed on a node. System release version information contained in this data structure can be accessed with the **GetSystemReleaseVersionInfo( )** function.

***Note:*** Applications using the CTPLATFORMVERSIONINFO data structure in the current release must be recompiled. The CTPLATFORMVERSIONINFO has changed in the current release; the lServiceUpdateNumber, lServiceUpdateBuildNumber, lFeatureReleaseNumber and lFeatureReleaseBuildNumber data fields have been added.

## Field Descriptions

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

lSystemReleaseMajorReleaseNumber
system software major release number

lSystemReleaseMinorReleaseNumber
system software minor release number

lSystemReleaseSubMinorReleaseNumber
system software sub-minor release number

lSystemReleaseBuildNumber
system software build number

lSystemServicePackNumber
if you have a service pack installed on the node, this field indicates the service pack number

lSystemServicePackBuildNumber
if you have a service pack installed on the node, this field indicates the service pack build number

lSystemFeaturePackNumber
if you have a feature pack installed on the node, this field specifies the feature pack number

lSystemFeaturePackBuildNumber
: if you have a feature pack installed on the node, this field indicates the feature pack build number

lServiceUpdateNumber
: if you have a service update installed on the node, this field specifies the service update number

lServiceUpdateBuildNumber
: if you have a service update installed on the node, this field indicates the service update build number

lFeatureReleaseNumber
: if you have a feature release installed on the node, this field specifies the feature release number

lFeatureReleaseBuildNumber
: if you have a feature release installed on the node, this field indicates the feature pack build number

# DevAdminEventMsg

```
typedef struct DevAdminEventMsgT
{
      AUID   auid;
      char   szDescription[MAX_EVENT_DESCRIPTION];
} DevAdminEventMsg, *DevAdminEventMsgPtr;
```

## Description

The DevAdminEventMsg data structure defines the payload format for events that are carried on the ADMIN_CHANNEL.

Refer to Chapter 3, "Events" for a list of event channels.

## Field Descriptions

The field of the DevAdminEventMsg data structure is described as follows:

auid
: AUID of the board that was started, stopped, inserted, removed, about to be removed or detected

szDescription
: ASCIIZ, NULL-terminated string that contains a description of the administration fault

***Note:*** The following events are system-level events; therefore their payloads will not contain AUIDs and your application should ignore the value in the AUID field:

- DLGC_EVT_SYSTEM_STARTED
- DLGC_EVT_SYSTEM_STOPPED
- DLGC_EVT_SYSTEM_ABOUTTO_START
- DLGC_EVT_SYSTEM_ABOUTTO_STOP

# DlgEventMsgType

```
typedef struct AdminCallbackMsg
{
     unsigned long      msgId;
     wchar_t*           supplierName;
     char*              node;
     int                payLoadLen;
     unsigned char*     pPayLoad;
     int                conversion;
     char*              description;
     char*              date;
     char*              time;
     } AdminCallbackMsgType;
```

## Description

The DlgEventMsgType is an alias for the AdminCallbackMsgType data structure. The AdminCallbackMsgType data structure defines the generic format of all events that are generated by supplier objects and received by consumer objects. A pointer to the event's payload is included in the pPayLoad field.

***Note:*** Aliases for AdminCallbackMsgType and AdminAllbackMsgType* are defined by the following two lines in the *dlgeventproxydef.h* file:

```
typedef AdminCallbackMsgType     DlgEventMsgType;
typedef AdminCallbackMsgType*    DlgEventMsgTypePtr;
```

DlgEventMsgTypePtr is the data type for the **CEventHandlerAdaptor::HandleEvent( ) evMsg** parameter.

## Field Descriptions

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

msgId
: name of the event. The consumer object uses the msgId to correctly typecast the message payload. Refer to Chapter 3, "Events" for a list of events.

supplierName
: name of the supplier object that generated the event.

  ***Note:*** Values shown in the supplierName field are for informational purposes only and subject to change in future Intel® Dialogic® System Software releases.

node
: IP address of the node containing the supplier object that generated the event

payLoadLen
: size of the event message in bytes

pPayLoad
: pointer to the messages payload data structure. The payload format for events varies according to the event notification channel that carries the event. Refer to the *OA&M API for Linux Programming Guide* for complete information about event notification channels.

conversion
: used to indicate if conversion is needed

description
provides a description of the event

date
provides the date the event was sent

time
provides the time the event was sent

# NetworkEventMsg

```
typedef struct NetworkEventMsgT
{
     AUID  auid;
     int   externalRef;
     short nPhysicalBusNumber;
     char  szDescription[MAX_EVENT_DESCRIPTION];
} NetworkEventMsg, *NetworkEventMsgPtr;
```

## Description

The NetworkEventMsg data structure defines the payload format for events that are carried on the NETWORK_ALARM_EVENT channel.

Refer to Chapter 3, "Events" for a list of event channels.

## Field Descriptions

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

auid
: AUID of the clocking agent that generated the event

externalRef
: the trunk number that generated the network alarm

nPhysicalBusNumber
: the physical bus number that the board is on

szDescription
: ASCIIZ, NULL-terminated string that contains a description of the alarm

# ProcessorFaultMsg

```
typedef struct ProcessorFaultMsgT
{
     AUID   auid;
     short  nProcessorNumber;
     char   szDescription[MAX_EVENT_DESCRIPTION]
} ProcessorFaultMsg, *ProcessorFaultMsgPtr;
```

## Description

The ProcessorFaultMsg data structure defines the payload format for events that are carried on the FAULT_CHANNEL.

Refer to Chapter 3, “Events” for a list of events.

## Field Descriptions

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

auid
: AUID of the board that contains the Digital Signal Processor (DSP) that generated the fault

nProcessorNumber
: number of the DSP that the fault occurred on

szDescription
: ASCIIZ, NULL-terminated string that contains a description of the fault

# Error Codes 6

This chapter provides reference information about the error code descriptions that are included in CCTDasiException objects.

Functions in the CCTDomain, ICTNode, ICTBoard and ICTClockAgent classes generate a CCTDasiException object if an error occurs. CCTDasiException objects contain an error id and an error description. The following list includes all error descriptions for CCTDasiException objects:

DLG_INVALID_BOARDAUID
: Application referenced an incorrect board AUID.

DLG_INVALID_PARAM
: Function was passed an invalid parameter.

DLG_INVALID_SERVER
: Function was passed incorrect name server information.

DLG_DB_INTERNAL_ERROR
: An error occurred in the internal OA&M API database.

DLG_CLKAPI_OPENFAIL
: OA&M API was unable to open an object.

DLG_CLKAPI_PATHOPENFAIL
: OA&M API was unable to open a path between objects.

DLG_CLKAPI_COMMUNICATIONSFAIL
: Communications failure between objects in the system.

DLG_CLKAPI_MESSAGEFAIL
: Message transmission between objects failed.

DLG_CLKAPI_WRONGREPLYFAIL
: Object received an incorrect reply.

***Note:*** The CCTDasiException class is defined in the *CCTDasiException.h* header file.

# Index