Event Service API for Windows Operating Systems

Library Reference

April 2006

05-1905-005

Publication Date: April 2006

Document Number: 05-1905-005

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 and Compute Products website at:
http://www.intel.com/design/network/products/telecom

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

Contents

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1905-005	April 2006	CEventHandlerAdaptor::HandleEvent( ): Removed redundant lines from Example: include "dlgcevents.h". NetworkEventMsg section: Rewrote description of AUID. (PTR 32157) Error Codes: Replaced *dlgsyserrors.h* with *dlgeventproxydef.h.*
05-1905-004	October 2005	NetworkEventMsg section: Added note to the auid data structure field about the channel returning invalid AUIDs. Events chapter: Noted that Bridging events are only supported for Intel NetStructure Host Media Processing release. DevBridgingEventMsgT section: Noted that this data structure is only supported for Intel NetStructure Host Media Processing release.
05-1905-003	October 2005	Global change: Minor editorial updates.
05-1905-003-01	September 2005	Events chapter: :Added bridging device events for Intel NetStructure Host Media Processing release. DebBridgingEventMsgT section: Added bridging device event data structure Intel NetStructure Host Media Processing release.
05-1905-002	November 2003	Minor editorial changes.
05-1905-001	November 2002	Initial version of document.

About This Publication

The following topics provide information about this *Event Service API for Windows Operating Systems Library Reference*:

Purpose

This publication provides a reference to all classes, functions, data structures, and events in the Event Service library for Intel® telecom products.

This publication is a companion document to the *Event Service API for Windows Operating Systems Programming Guide*, which provides guidelines for designing applications that are capable of receiving system administrative events via the event notification framework.

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 publication after you have installed the Intel telecom hardware and Intel Dialogic system software release, which includes the Event Service library.

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

The information in this publication is organized as follows:

- Chapter 1, "Function Summary by Category" introduces you to the categories of functions in the Event Service library. Functions are organized by C++ class membership.
- Chapter 2, "Function Information" provides a reference to the Event Service API functions.
- Chapter 3, "Events" lists names and descriptions of the events that are transmitted via the various channels of the event notification framework.
- Chapter 4, "Data Types" includes reference information about the data types that are defined by the Event Service API.
- Chapter 5, "Data Structures" defines the data structures used by the Event Service API.
- Chapter 6, "Error Codes" lists and describes the error codes that are used by the Event Service API.

Related Information

Refer to the following for more information:

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

Function Summary by Category 1

The Event Service API functions are grouped into the following categories:

Note: Each category corresponds to a C++ class. The DlgAdminConsumer class is defined in the *dlgadminconsumer.h* file. The CEventHandlerAdaptor class is defined in the *dlgadminmsg.h* file.

1.1 DlgAdminConsumer Class Functions

You must instantiate at least one DlgAdminConsumer object to receive asynchronous events from the event notification framework. The DlgAdminConsumer class member functions are used to instantiate event consumer objects and return information about the objects.

The DlgAdminConsumer class contains the following functions:

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

DlgAdminConsumer::DlgAdminConsumer( )
Instantiates a DlgAdminConsumer object.

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

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

DlgAdminConsumer::getConsumerName( )
Returns the user-defined name of the DlgAdminConsumer object.

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

Note: The DlgAdminConsumer class uses the DlgEventService namespace.

1.2 CEventHandlerAdaptor Class Functions

Applications must implement a class derived from the CEventHandlerAdaptor class to instantiate event handler objects. The CEventHandlerAdaptor class member function is invoked by the DlgAdminConsumer object when an event is received from the event notification framework.

The CEventHandlerAdaptor class contains the following virtual function:

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

Note: The CEventHandlerAdaptor class uses the DlgEventService namespace.

Function Information 2

This chapter provides a reference to the functions in the Event Service library. Functions are listed according to C++ class membership.

2.1 Function Syntax Conventions

The Event Service functions use the following syntax:

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

where:

return_type
: indicates the data type of the return type or return field

function_name
: represents the function name

parameter1
: refers to the first parameter

parameterN
: refers to the last parameter

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"

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 event notification framework. Topics include:

Notes: ***1.*** Refer to Chapter 5, "Data Structures" for information about the data structures and payload formats used by the events.

2. Refer to the *Event Service API for Windows Operating Systems Programming Guide* for information about enabling applications to receive and handle events.

3.1 ADMIN_CHANNEL Events

The following events are transmitted on the ADMIN_CHANNEL:

Note: Refer to the *Native Configuration Manager API for Windows Operating Systems Library Reference* for complete information about the NCM functions discussed in this section.

DLGC_EVT_BLADE_ABOUT_TO_REMOVE
: Generated when the **Device > Remove/Uninstall Device** option is selected in the Intel® Dialogic® Configuration Manager (DCM).

DLGC_EVT_BLADE_ABOUT_TOSTART
: Occurs when an individual board start command has been issued (either through the DCM's **Device > Start Device** option or programmatically with the **NCM_StartBoard( )** function).

DLGC_EVT_BLADE_ABOUT_TOSTOP
: Occurs when an individual board stop command has been issued (either through the DCM's **Device > Stop Device** option or programmatically with the **NCM_StopBoard( )** function).

DLGC_EVT_BLADE_DETECTED
: Indicates that a newly inserted board has been detected by the Intel Dialogic system software release and its initial configuration information has been stored in the NCM database.

DLGC_EVT_BLADE_REMOVED
: Generated when a board has been removed from the system and its configuration information has been deleted from the NCM database.

DLGC_EVT_BLADE_START_FAILED
: Occurs if an individual board start sequence has failed. The board start sequence can be initiated through DCM's **Device > Start Device** option or programmatically with the **NCM_StartBoard( )** function.

DLGC_EVT_BLADE_STARTED
Generated immediately after an individual board has been successfully started. The board start start can be initiated through DCM's **Device > Start Device** option or programmatically with the **NCM_StartBoard( )** function.

DLGC_EVT_BLADE_STOPPED
Generated immediately after an individual board has been successfully stopped. The board stop can be initiated through DCM's **Device > Stop Device** option or programmatically with the **NCM_StopBoard( )** function.

DLGC_EVT_SYSTEM_ABOUTTOSTART
Occurs when a Intel Dialogic system start command has been issued (either through the DCM's **System > Start System** option or programmatically with the **NCM_StartDlgSrv( )** function).

DLGC_EVT_SYSTEM_ABOUTTOSTOP
Occurs when a Intel Dialogic system stop command has been issued (either through the DCM's **System > Stop System** option or programmatically with the **NCM_StopDlgSrv( )** function).

DLGC_EVT_SYSTEM_STARTED
Generated immediately after the Intel Dialogic system has been successfully started. The system start can be initiated through DCM's **System > Start System** option or programmatically with the **NCM_StartDlgSrv( )** function.

DLGC_EVT_SYSTEM_STOPPED
Generated immediately after the Intel Dialogic system has been successfully stopped. The system stop can be initiated through DCM's **System > Stop System** option or programmatically with the **NCM_StopDlgSrv( )** function.

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_TOSTART	DevAdminEventMsg	sizeof(DevAdminEventMsg)
DLGC_EVT_BLADE_ABOUT_TOSTOP	DevAdminEventMsg	sizeof(DevAdminEventMsg)
DLGC_EVT_BLADE_DETECTED	AUID	sizeof(AUID) = 4 bytes
DLGC_EVT_BLADE_REMOVED	AUID	sizeof(AUID) = 4 bytes
DLGC_EVT_BLADE_START_FAILED	DevAdminEventMsg	sizeof(DevAdminEventMsg)
DLGC_EVT_BLADE_STARTED	DevAdminEventMsg	sizeof(DevAdminEventMsg)
DLGC_EVT_BLADE_STOPPED	DevAdminEventMsg	sizeof(DevAdminEventMsg)
DLGC_EVT_SYSTEM_ABOUTTOSTART	NULL	0 bytes
DLGC_EVT_SYSTEM_ABOUTTOSTOP	NULL	0 bytes
DLGC_EVT_SYSTEM_STARTED	NULL	0 bytes
DLGC_EVT_SYSTEM_STOPPED	NULL	0 bytes

3.2 BRIDGING_CHANNEL Events

The following events are transmitted on the BRIDGING_CHANNEL:

Note: BRIDGING_CHANNEL events are only supported on Intel NetStructure® Host Media Processing system software releases.

DLGC_EVT_BRIDGE_DEVICE_DETECTED
Sent when a new bridge device is detected. A new bridge device can be detected when the DCM is started, a board is inserted into a system that supports hot swap, or when the user requests re-detection of devices.

DLGC_EVT_BRIDGE_DEVICE_FAILED
Sent when the system has determined that a bridge device failed.

DLGC_EVT_BRIDGE_DEVICE_HMP_CLOCK_MASTER
Occurs when a bridge device HMP clock master is selected by the bridge controller.

DLGC_EVT_BRIDGE_DEVICE_REMOVED
Sent when a bridge device is removed from the system. This event is generated when DCM is started or when a board is removed from a system that supports hot swap.

DLGC_EVT_BRIDGE_DEVICE_STARTED
Occurs when a bridge device is started.

DLGC_EVT_BRIDGE_DEVICE_STARTING
Occurs when a bridge device is in the process of starting.

DLGC_EVT_BRIDGE_DEVICE_STOPPED
Occurs when a bridge device is stopped.

DLGC_EVT_BRIDGE_DEVICE_STOPPING
Occurs when a bridge device is in the process of stopping.

3.3 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 Line A fails.

DLGC_EVT_CT_B_LINESBAD
Occurs if the signal on the CT Bus Line B 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 NetRef1 line has failed.

DLGC_EVT_SCBUS_COMPAT_LINESBAD
Occurs if the SCbus compatibility line fails.

3.4 FAULT_CHANNEL Events

The following events are transmitted on the FAULT_CHANNEL:

DLGC_EVT_CP_FAILURE
Generated when a Control Processor failure occurs on an Intel NetStructure® board.

DLGC_EVT_SP_FAILURE
Generated when a Signal Processor failure occurs on an Intel NetStructure board.

3.5 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 (local) end of a T1 or E1 line has detected a loss of signal or frame alignment in the incoming data.

DLGC_EVT_EXTERNAL_ALARM_RED_CLEAR
Signifies that the condition which caused the red alarm has recovered and the alarm has been cleared.

DLGC_EVT_EXTERNAL_ALARM_YELLOW
Generated when a transmit circuit fails in the data transmission path. The device that detects the failed circuit sends a yellow alarm to the device that contains the failed circuit.

DLGC_EVT_EXTERNAL_ALARM_YELLOW_CLEAR
Indicates that a yellow alarm is no longer being sent to the device.

DLGC_EVT_EXTERNAL_LOSS_OF_SIGNAL
Indicates that a signal is not being detected on the incoming T1 or E1 line.

DLGC_EVT_EXTERNAL_LOSS_OF_SIGNAL_CLEAR
Occurs when a signal has been detected on the incoming T1 or E1 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 Event Service API.

The following list contains data types that are defined in the Event Service API:

AUID
: Long integer data type that indicates the Addressable Unit Identifier of a system component. The Intel Dialogic system software release assigns a unique AUID to each system component with which communications can be initiated.

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 Event Service API. Information is provided for the following data structures:

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

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

DevBridgingEventMsgT

```
typedef struct DevBridgingEventMsgT
{
      AUID auid;
      char szDescription[MAX_EVENT_DESCRIPTION];    /*Comment foe second field */
} DevBridgingEventMsg, *DevBridgingEventMsgPtr;
```

Description

The DevBridgingEventMsgT data structure defines the payload format for events that are carried on the BRIDGING_CHANNEL.

Note: The DevBridgingEventMsgT data structure is only supported on Intel NetStructure® Host Media Processing system software releases.

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

Field Descriptions

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

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

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

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 *Event Service API for Windows 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 dtiB<n> virtual board that generated the network alarm (n starting with 1)

Note: The network alarms are against Digital Telephony Interface (DTI) interfaces. Therefore, the event that is broadcast concerns a dtiB<n> virtual board, and the event contains the AUID of the virtual board. Note that Intel NetStructure® DM/T and DM/V boards can have up to 16 network interfaces, so <n> can be an integer between 1 and 16 (dtiB<n>). Clock Fallback concerns the ability of the board to be Clock Master, so this event is sent with the AUID of the physical board.

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 describes the error codes that can be returned by the **CEventHandlerAdaptor::HandleEvent( )** function.

The **CEventHandlerAdaptor::HandleEvent( )** can return any one of several integer values. Each returned integer value is associated with a defined error message. Returned integer values and their associated error messages are listed and described below:

5000
: Equivalent to the DlgEvent_READY message. This message indicates that the consumer object is ready to receive incoming events.

5001
: Equivalent to the DlgEvent_ERROR_INIT message. This message indicates that an error occurred during initialization of the consumer object.

5002
: Equivalent to the DlgEvent_NOT_READY message. This message indicates that the consumer object has been instantiated but not initialized.

5003
: Equivalent to the DlgEvent_CHANNEL_ERROR message. This message indicates that an error occurred within the consumer object's associated event channel. Refer to Chapter 3, "Events" for a list of event channels.

The DlgEvent_READY, DlgEvent_ERROR_INIT, DlgEvent_NOT_READY and DlgEvent_CHANNEL_ERROR messages are defined in the *dlgeventproxydef.h* file.