



# Dialogic<sup>®</sup> Event Service API for Windows<sup>®</sup>

Library Reference

---

*March 2008*

Copyright © 2002-2008 Dialogic Corporation. All rights reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. **Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.**

Dialogic, Dialogic Pro, Brooktrout, Cantata, SnowShore, Eicon, Eicon Networks, Eiconcard, Diva, SIPcontrol, Diva ISDN, TruFax, Realbloccs, Realcomm 100, NetAccess, Instant ISDN, TRXStream, Exnet, Exnet Connect, EXS, ExchangePlus VSE, Switchkit, N20, Powering The Service-Ready Network, Vantage, Connecting People to Information, Connecting to Growth, and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement. Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. Other names of actual companies and products mentioned herein are the trademarks of their respective owners.

Publication Date: March 2008

Document Number: 05-1905-006

# Contents

---

	<b>Revision History</b> .....	5
	<b>About This Publication</b> .....	7
	Purpose .....	7
	Applicability .....	7
	Intended Audience .....	7
	How to Use This Publication .....	8
	Related Information .....	8
<b>1</b>	<b>Function Summary by Category</b> .....	9
	1.1 DlgAdminConsumer Class Functions .....	9
	1.2 CEventHandlerAdaptor Class Functions .....	9
<b>2</b>	<b>Function Information</b> .....	11
	2.1 Function Syntax Conventions .....	11
	DlgAdminConsumer::DisableFilters( ) .....	12
	DlgAdminConsumer::DlgAdminConsumer( ) .....	15
	DlgAdminConsumer::EnableFilters( ) .....	18
	DlgAdminConsumer::getChannelName( ) .....	21
	DlgAdminConsumer::getConsumerName( ) .....	24
	DlgAdminConsumer::StartListening( ) .....	27
	CEventHandlerAdaptor::HandleEvent( ) .....	30
<b>3</b>	<b>Events</b> .....	33
	3.1 ADMIN_CHANNEL Events .....	33
	3.2 BRIDGING_CHANNEL Events .....	35
	3.3 CLOCK_EVENT_CHANNEL Events .....	35
	3.4 FAULT_CHANNEL Events .....	36
	3.5 NETWORK_ALARM_CHANNEL Events .....	36
<b>4</b>	<b>Data Types</b> .....	37
<b>5</b>	<b>Data Structures</b> .....	39
	ClockingFaultMsg – payload format of CLOCK_EVENT_CHANNEL events .....	40
	DevAdminEventMsg – payload format of ADMIN_CHANNEL events .....	41
	DevBridgingEventMsgT – payload format of BRIDGING_CHANNEL events .....	42
	DlgEventMsgType – generic event message .....	43
	NetworkEventMsg – payload format of NETWORK_ALARM_CHANNEL events .....	45
	ProcessorFaultMsg – payload format of FAULT_CHANNEL events .....	46
<b>6</b>	<b>Error Codes</b> .....	47
	<b>Index</b> .....	49

**Contents**

# Revision History

---

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

Document No.	Publication Date	Description of Revisions
05-1905-006	March 2008	Made global changes to reflect Dialogic brand and changed title to “Dialogic® Event Service API for Windows® Library Reference.”
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 <i>dlgsyserrors.h</i> with <i>dlgeventproxydef.h</i> .
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 Dialogic® Host Media Processing release. DevBridgingEventMsgT section: Noted that this data structure is only supported for Dialogic® 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 Dialogic®Host Media Processing release. DevBridgingEventMsgT section: Added bridging device event data structure for Dialogic® Host Media Processing release.
05-1905-002	November 2003	Minor editorial changes.
05-1905-001	November 2002	Initial version of document.

## ***Revision History***

# About This Publication

---

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

- [Purpose](#)
- [Applicability](#)
- [Intended Audience](#)
- [How to Use This Publication](#)
- [Related Information](#)

## Purpose

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

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

## Applicability

This document version (05-1905-006) is published for Dialogic® Host Media Processing Software Release 3.0WIN.

This document may also be applicable to other software releases (including service updates) on Windows® operating systems. Check the Release Guide for your software release to determine whether this document is supported.

## 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 Dialogic® hardware and system software release, which includes the Dialogic® Event Service API 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

See the following for additional information:

- <http://www.dialogic.com/manuals/> (for Dialogic® product documentation)
- <http://www.dialogic.com/support/> (for Dialogic technical support)
- <http://www.dialogic.com/> (for Dialogic® product information)



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

- [DlgAdminConsumer Class Functions](#) . . . . . 9
- [CEventHandlerAdaptor Class Functions](#) . . . . . 9

**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.

## **Function Summary by Category**

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.

This chapter provides a reference to the functions in the Dialogic® Event Service API 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
<b>pfilters</b>	points to the array of filters that will be disabled
<b>iCount</b>	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.

## ***DlgAdminConsumer::DisableFilters()***

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

/*user defined header file that extends the CEventHandlerAdaptor 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();
}
```

## ***DlgAdminConsumer::DisableFilters()***

```
//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
<b>szChannelName</b>	determines which event notification channel the consumer object registers with and monitors for events.
<b>szConsumerName</b>	indicates the unique name of the consumer object
<b>pFilters</b>	points to an array of filter to event handler object associations.
<b>iFilterCnt</b>	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

## **DlgAdminConsumer::DlgAdminConsumer()**

“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;
```



## ***DlgAdminConsumer::DlgAdminConsumer()***

```
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( )

# 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
<b>pFilters</b>	points to the array of filters that will be enabled
<b>iCount</b>	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)
```

## ***DlgAdminConsumer::EnableFilters()***

```
{
    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::EnableFilters( )***

- ***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();
}
```

## **DlgAdminConsumer::getChannelName( )**

```
    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( )**

## 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 **getConsumerName( )** function returns the name of the DlgAdminConsumer object. The DlgAdminConsumer object name is determined by the **szConsumerName** parameter when the DlgAdminConsumer 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();
}
```



## ***DlgAdminConsumer::getConsumerName()***

```
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;
}
```

## ***DlgAdminConsumer::getConsumerName()***

### ■ **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();
}
```

## DlgAdminConsumer::StartListening()

```
    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::StartListening()***

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

## ***CEventHandlerAdaptor::HandleEvent( )***

# **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.

<b>Parameter</b>	<b>Description</b>
<b>evMsg</b>	points to the actual event ( <a href="#">DlgEventMsgType</a> data structure) that is sent by the supplier object
<b>clientData</b>	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>

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

## ***CEventHandlerAdaptor::HandleEvent()***

```
#include "dlgeventproxydef.h"
#include "dlgevents.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

***CEventHandlerAdaptor::HandleEvent()***



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

- ADMIN\_CHANNEL Events . . . . . 33
- BRIDGING\_CHANNEL Events . . . . . 35
- CLOCK\_EVENT\_CHANNEL Events . . . . . 35
- FAULT\_CHANNEL Events . . . . . 36
- NETWORK\_ALARM\_CHANNEL Events . . . . . 36

- 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 *Dialogic® Event Service API 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 *Dialogic® Native Configuration Manager API 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 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 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.

## Events

### DLGC\_EVT\_BLADE\_STARTED

Generated immediately after an individual board has been successfully started. The board 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 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 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 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 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 Dialogic® Host Media Processing (HMP) 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 Dialogic® HMP Software 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.

## Events

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 a Dialogic® board.

DLGC\_EVT\_SP\_FAILURE

Generated when a Signal Processor failure occurs on a Dialogic 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.

This chapter contains reference information about the various data types used by the Dialogic® 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 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 Dialogic system software releases.

## ***Data Types***

This chapter includes reference information about the data structures that are used by the Dialogic® Event Service API. Information is provided for the following data structures:

- [ClockingFaultMsg](#) . . . . . 40
- [DevAdminEventMsg](#) . . . . . 41
- [DevBridgingEventMsgT](#) . . . . . 42
- [DlgEventMsgType](#) . . . . . 43
- [NetworkEventMsg](#) . . . . . 45
- [ProcessorFaultMsg](#) . . . . . 46

## 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 Dialogic® Host Media Processing (HMP) 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 Dialogic® software releases.

#### node

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

#### payLoadLen

size of the event message in bytes

## ***DlgEventMsgType*** — generic event message

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 *Dialogic® 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 Dialogic® 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

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.

## ***Error Codes***



# Index

---

## A

ADMIN\_CHANNEL 33  
array of filters 12, 15  
AUID 37, 40, 41

## B

bridge device 42  
BRIDGING\_CHANNEL 35, 42

## C

CEventHandlerAdaptor 9  
channel name 21  
ClientDataType 37  
CLOCK\_EVENT\_CHANNEL 35  
ClockingFaultMsg 40

## D

DevAdminEventMsg 41  
DevBridgingEventMsgT 42  
disable an individual filter 12  
DlgAdminConsumer 9  
DlgAdminConsumer() 15  
dlgadminconsumer.h 9  
dlgadminmsg.h 9  
DLGC\_EVT\_BLADE\_ABOUT\_TO\_REMOVE 33  
DLGC\_EVT\_BLADE\_ABOUT\_TOSTART 33  
DLGC\_EVT\_BLADE\_ABOUT\_TOSTOP 33  
DLGC\_EVT\_BLADE\_DETECTED 33  
DLGC\_EVT\_BLADE\_REMOVED 33  
DLGC\_EVT\_BLADE\_START\_FAILED 33  
DLGC\_EVT\_BLADE\_STARTED 34  
DLGC\_EVT\_BLADE\_STOPPED 34  
DLGC\_EVT\_BRIDGE\_DEVICE\_DETECTED 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_FAILED 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_HMP\_CLOCK\_MASTE  
R 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_REMOVED 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_STARTED 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_STARTING 35

DLGC\_EVT\_BRIDGE\_DEVICE\_STOPPED 35  
DLGC\_EVT\_BRIDGE\_DEVICE\_STOPPING 35  
DLGC\_EVT\_CP\_FAILURE 36  
DLGC\_EVT\_CT\_A\_LINESBAD 35  
DLGC\_EVT\_CT\_B\_LINESBAD 35  
DLGC\_EVT\_EXTERNAL\_ALARM\_RED 36  
DLGC\_EVT\_EXTERNAL\_ALARM\_RED\_CLEAR 36  
DLGC\_EVT\_EXTERNAL\_ALARM\_YELLOW 36  
DLGC\_EVT\_EXTERNAL\_ALARM\_YELLOW\_CLEAR  
36  
DLGC\_EVT\_EXTERNAL\_LOSS\_OF\_SIGNAL 36  
DLGC\_EVT\_EXTERNAL\_LOSS\_OF\_SIGNAL\_CLEAR  
36  
DLGC\_EVT\_LOSS\_MASTER\_SOURCE\_INVALID 35  
DLGC\_EVT\_MVIP\_COMPAT\_LINESBAD 35  
DLGC\_EVT\_NETREF1\_LINEBAD 36  
DLGC\_EVT\_SCBUS\_COMPAT\_LINESBAD 36  
DLGC\_EVT\_SP\_FAILURE 36  
DLGC\_EVT\_SYSTEM\_ABOUTTOSTART 34  
DLGC\_EVT\_SYSTEM\_ABOUTTOSTOP 34  
DLGC\_EVT\_SYSTEM\_STARTED 34  
DLGC\_EVT\_SYSTEM\_STOPPED 34  
DlgEvent\_CHANNEL\_ERROR 47  
DlgEvent\_ERROR\_INIT 47  
DlgEvent\_NOT\_READY 47  
DlgEvent\_READY 47  
DlgEventMsgType 43  
dlgeventproxydef.h 47  
DlgEventService 9  
DlgFilterType 37

## E

EnableFilters() 18  
event handler 15, 30  
event payload  
    administrative events 41  
    clocking events 40  
    network alarm events 45  
    processor fault events 46

## F

FAULT\_CHANNEL 36  
filter array 12, 15

## G

getChannelName() 21  
getConsumerName() 24

## H

HandleEvent() 30

## I

IpAddressStringType 37

## L

listening for events 27

## M

monitoring a channel 27

## N

namespace 9, 10  
NETWORK\_ALARM\_CHANNEL 36  
NetworkEventMsg 45

## P

PayloadDataType 37  
ProcessorFaultMsg 46

## S

StartListening() 27  
SupplierNameType 37