



Dialogic[®] Event Service API

Programming Guide

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: 05-1918-004

Contents

	Revision History	5
	About This Publication	7
	Purpose	7
	Applicability	7
	Intended Audience	7
	How to Use This Publication	8
	Related Information	8
1	Product Description	9
2	Event Handling	13
3	Error Handling	15
4	Application Development Guidelines	17
5	Building Applications	19
5.1	Compiling and Linking	19
5.1.1	Include Files	19
5.1.2	Required Libraries	19
5.1.3	Variables for Compiling and Linking	20
	Index	21

Figures

1	Event Notification Framework.	10
2	Event Service API Classes.	11

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1918-004	March 2008	Made global changes to reflect Dialogic brand and changed title to “Dialogic® Event Service API Programming Guide.”
05-1918-003	October 2005	Global change: Added information about BRIDGING_CHANNEL for Dialogic® Host Media Processing Software release
05-1918-002	November 2003	Chapter 4, “Application Development Guidelines”: Added note stating that the peripheral hot swap procedure can only be used with CompactPCI boards. Minor editorial changes.
05-1918-001	November 2002	Initial version of document.

Revision History

About This Publication

The following topics provide information about this *Dialogic® Event Service API Programming Guide*:

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

Purpose

This publication provides guidelines for using the Dialogic® Event Service API to register an application with the event notification framework in a Windows® programming environment.

This publication is a companion guide to the *Dialogic® Event Service API Library Reference*, which provides details on the classes, functions, and events in the Event Service library.

Applicability

This document version (05-1918-004) is published for Dialogic® Host Media Processing Software Release 3.0WIN.

Intended Audience

This publication is intended for the following audience:

- 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 hardware and the system software that 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, “Product Description”](#) provides an overview of the Event Service API.
- [Chapter 2, “Event Handling”](#) describes how to use the Event Service API to register an application with the event notification framework and handle system administration events.
- [Chapter 3, “Error Handling”](#) describes the error handling capabilities provided by the Event Service API.
- [Chapter 4, “Application Development Guidelines”](#) provides guidelines for using the Event Service API to build highly available applications.
- [Chapter 5, “Building Applications”](#) contains information about the required header files and library files included with the Event Service API.

Related Information

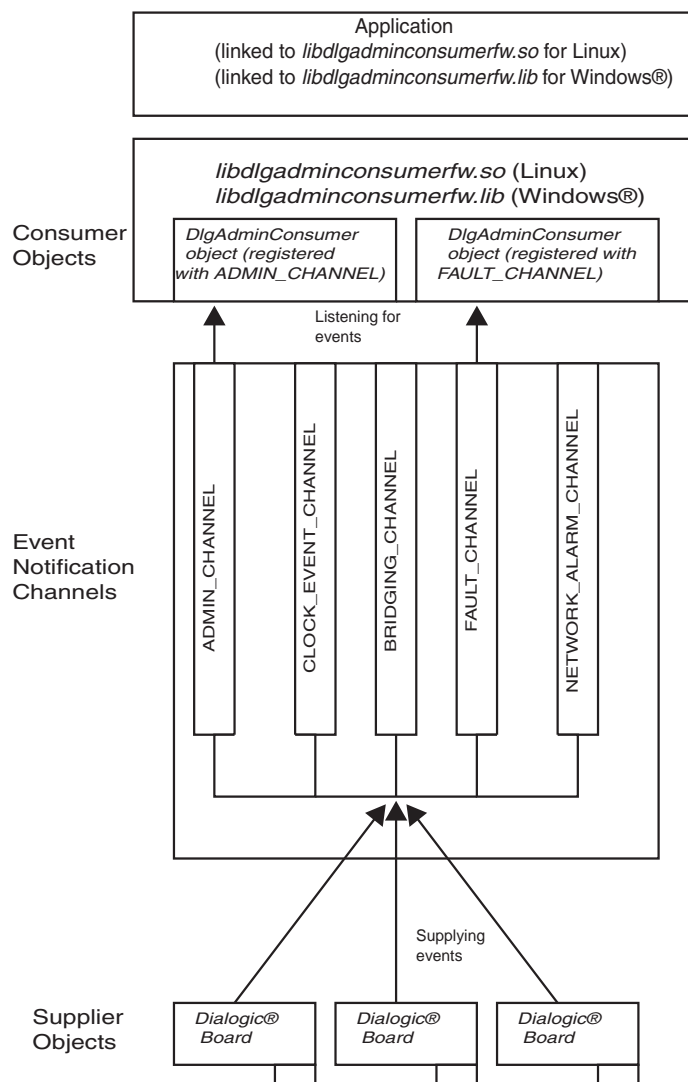
See the following for additional information:

- <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)

This chapter provides a description of the Dialogic® Event Service API and the event notification framework.

The Dialogic Event Service API provides an interface for registering your application with the system release event notification framework. The event notification framework is the subsystem for sending asynchronous system administration events to applications. The framework is implemented using supplier objects, consumer objects, and event notification channels as shown in Figure 1:

Figure 1. Event Notification Framework



Note: The CLOCK_EVENT_CHANNEL, NETWORK_ALARM_CHANNEL and BRIDGING_CHANNEL are idle in the figure shown above. These three channels are available, but do not have consumer objects registered to listen for events.

Dialogic® Host Media Processing (HMP) Software components, such as device drivers and firmware, are the supplier objects. They generate events that are broadcast to consumer objects via the event notification channels. The DlgAdminConsumer class allows you to instantiate consumer objects and register them to receive events from one of the event notification channels. The CEventHandlerAdaptor class allows you to instantiate user-defined event handler objects that are invoked when consumer objects receive events.

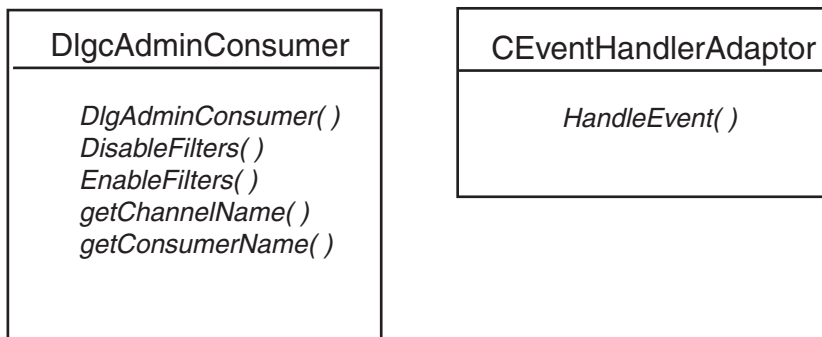
The framework contains the following event notification channels, each of which carries specific types of events:

- ADMIN_CHANNEL
- BRIDGING_CHANNEL (HMP system software only)
- CLOCK_EVENT_CHANNEL
- FAULT_CHANNEL
- NETWORK_ALARM_CHANNEL

Note: Each DlgAdminConsumer object can only monitor one event notification channel for incoming events.

The Event Service API contains two C++ classes. A DlgAdminConsumer class for instantiating event consumer objects and a CEventHandlerAdaptor virtual class for implementing event handler objects. The two Event Service API classes, along with their member functions, are shown in Figure 2:

Figure 2. Event Service API Classes



Product Description

This chapter provides information about receiving and handling asynchronous events that are transmitted via the event notification framework.

For your application to receive events from the event notification framework, you must follow these steps:

1. Define and implement a class that is derived from the `CEventHandlerAdaptor` class.
2. Provide an implementation of the `CEventHandlerAdaptor::HandleEvent()` function.
3. Define an array of filters for use by the `DlgAdminConsumer` object using `DlgEventService::AdminConsumer::FilterCallbackAssoc`. Each `DlgAdminConsumer` object references an array of filters. The elements in the array determine the following:
 - the event handler that is associated with the `DlgAdminConsumer` object. The implementation of the `CEventHandlerAdaptor::HandleEvent()` function is called when one of the events that is included in the array is received.
 - the client data that is returned to the application after the associated event handler object has been invoked.
 - the events that are allowed to pass to the `DlgAdminConsumer` object. If an event does not have an element in the filter array, the event is discarded and the `DlgAdminConsumer` object will not receive it.
 - whether the event filter is enabled or disabled.
4. Use the `DlgAdminConsumer::DlgAdminConsumer()` function to instantiate a consumer object. A pointer to the array initialized in step 3 is used as a parameter for the function.
5. Call the `DlgAdminConsumer::StartListening()` function so that the consumer object can begin monitoring its associated event notification channel for events. When the `DlgAdminConsumer::StartListening()` function is called, the consumer object creates and runs in its own thread, allowing it to monitor its associated event notification channel without blocking the main application thread.

When a `DlgAdminConsumer` object receives an event through its associated event notification channel, it compares the event's `msgId` field to its filter array. If a matching filter is found, the associated event handler object is invoked.

Note: When a `DlgAdminConsumer` object receives more than one event that is associated with the same event handler object, the `DlgAdminConsumer` object must return from the `CEventHandlerAdaptor::HandleEvent()` function before it can process the next event.

Event Handling

This chapter explains the error handling capabilities of the Event Service API.

Many of the Dialogic® Event Service API functions return error codes to indicate that a given function call has failed. The following list explains the error codes returned by the Event Service API functions:

DlgAdminConsumer::DlgAdminConsumer()

This function is a class constructor. It does not have a return value.

DlgAdminConsumer::getChannelName()

Returns NULL for a failure.

DlgAdminConsumer::getConsumerName()

Returns NULL for a failure.

DlgAdminConsumer::StartListening()

Returns Boolean value of False for failure.

Error Handling

Application Development Guidelines

4

This chapter provides information about using various Dialogic® libraries, including the Event Service library, to develop applications that support peripheral hot swap.

Note: Peripheral hot swap can only be performed using CompactPCI boards.

The board replacement could be driven by an application (for example, if the statistics are showing a degradation of service) or could be driven by an operator (for example, periodic hardware maintenance).

The following procedure provides information about developing applications that support Basic Hot Swap of peripheral hardware:

1. Register your application to receive events from the event notification framework according to the procedure outlined in [Chapter 2, “Event Handling”](#).
2. When a CP/SP fault, CT Bus clocking fault or network alarm event is transmitted via the event notification framework, the event’s payload contains the AUID of the board that generated the event.
3. Once the application has a board’s AUID, use the **SRLGetVirtualBoardsOnPhysicalBoard()** and **SRLGetSubDevicesOnVirtualBoard()** functions to retrieve the list of virtual devices (“dxxx”, “dti” etc.) on a board. This allows the application to cross-reference physical boards with virtual devices. For more information about these functions and the difference between physical and virtual boards, refer to the *Dialogic® Standard Runtime Library API Library Reference*.

Note: As a general high availability/peripheral hot swap application rule, you should design your application to accommodate the dynamic insertion/removal of virtual devices. Do not include permanent references to virtual device names within your application, instead, depend on the Standard Runtime Library (SRL) device mapper functions to get virtual device names.
4. The Standard Runtime Library functions can be used to determine the virtual devices on the board with the specified AUID. The virtual devices can be closed using the **dx_close()**, **dt_close()**, etc.
5. When all virtual devices on a board have been closed, you can either use the Dialogic® Configuration Manager (DCM) **Device > Stop Device** option or call the **NCM_StopBoard()** function to stop the board. When a board has been successfully stopped, a DLGC_EVT_BLADE_STOPPED event is generated on the ADMIN_CHANNEL.
6. Use the DCM’s **Device > Remove/Uninstall Device** option to remove the board’s configuration information from the DCM database. When the **Remove/Uninstall** option is selected from the DCM, a DLGC_EVT_BLADE_ABOUT_TO_REMOVE event is generated on the ADMIN_CHANNEL.
7. Physically remove the board from the chassis.

Application Development Guidelines

8. Insert a new board into the chassis and configure it according to the procedures in the appropriate configuration guide. When the Dialogic Plug and Play subsystem detects the inserted board, a DLGC_EVT_BLADE_DETECTED event is generated on the ADMIN_CHANNEL.
9. Use the DCM's **Device > Start Device** option or invoke the **NCM_StartBoard()** function to start the board.
10. When a board has been successfully started, a DLGC_EVT_BLADE_STARTED event is transmitted on the event notification framework's ADMIN_CHANNEL. Use the AUID from the event's payload, along with the Standard Runtime Library functions and the virtual device functions (**dx_open()**, **dt_open()**, etc.) to determine the virtual devices that are on the newly inserted board and open the board's virtual devices.
Note: If the board fails to start, a DLGC_EVT_BLADE_START_FAILED event will be transmitted on the ADMIN_CHANNEL.
11. Once a board's virtual devices are open, the board can be used by your application.

Refer to the *Dialogic® Native Configuration Manager API Library Reference* and the *Dialogic® Native Configuration Manager API Programming Guide* for information about the functions in the NCM API.

For information about configuring Dialogic® boards, refer to the *Dialogic® Host Media Processing Software for Windows Configuration Guide*.

This chapter provides general information for building applications that use the Dialogic® Event Service API library. The following topics are included:

- [Compiling and Linking](#) 19

5.1 Compiling and Linking

An application that uses the Event Service API library must include references to the Event Service API header files and must link to the appropriate library files. This information is provided in the following topics:

- [Include Files](#)
- [Required Libraries](#)
- [Variables for Compiling and Linking](#)

5.1.1 Include Files

The following header files are required by applications that receive and process events from the event notification framework:

dlgadminconsumer.h
defines the DlgAdminConsumer class

dlgadminmsg.h
defines the CEventHandlerAdaptor class

dlgevents.h
header file that includes references to all event-specific header files

dlgeventproxydef.h
defines the generic data structure for events

Note: You are not required to include the *dlgctypes.h* file in your application. However, if the file is included in your application, you will receive numerous warnings when the application is compiled. The warnings are Standard Template Library-related warnings and can be ignored.

5.1.2 Required Libraries

The following event service library file must be linked to applications that receive and process events from the event notification framework:

libdlgadminconsumerfw.lib
library file that contains the event consumer objects

5.1.3 Variables for Compiling and Linking

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

INTEL_DIALOGIC_INC

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

INTEL_DIALOGIC_LIB

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

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

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

Note: Developers should use these variables when compiling and linking applications, since it is planned that these variables will be required in future releases. It is also planned that the name of the variables will remain constant, but the values may change in future releases.

Index

A

array of filters 13
AUID 17

C

C++ classes 11
CEventHandlerAdaptor 13
channels of the event framework 11
client data 13
consumer objects 9

D

dlgadminconsumer.h 19

E

event handler 13

F

filter array 13

I

INTEL_DIALOGIC_INC 20
INTEL_DIALOGIC_LIB 20

L

libdlgadminconsumerfw.lib 19

M

msgId 13

N

NCM_StartBoard() 18
NCM_StopBoard() 17

S

SRLGetSubDevicesOnVirtualBoard() 17

SRLGetVirtualBoardsOnPhysicalBoard() 17
supplier objects 9

