



Running Applications Using Dialogic® Global Call Software as Windows® Services



Executive Summary

This application note explains how to control the startup and shutdown of a telephony application using Dialogic® Global Call Software from a Windows® service. An overall sample architecture is discussed, and descriptions of the processes involved are given. Instructions and code for running the SampleService and Win32App demo programs are provided as well.



Table of Contents

Introduction.....	2
Environment.....	2
Downloads	2
Architecture	2
Service Control Process	3
Avoiding Simultaneous Shutdown	4
System Setup, Build, and Execution of Sample Code.....	5
Setting up the System	5
Building the Applications.....	6
Running the Applications	6
Acronyms.....	7
For More Information.....	7

Introduction

There does not appear to be a “best” method for running a telephony application using Dialogic® Global Call Software as a Windows® system service. Certain considerations are important to keep in mind to work with the Dialogic® Global Call API libraries correctly. For example, incorrect procedures may cause difficulties in shutting down the Dialogic Global Call API libraries. This document discusses a method for avoiding these difficulties. C/C++ code and two Visual C++® projects are included in demo programs to illustrate the method.

Environment

The hardware and software used in the test environment for the method described in this document consist of the following components:

- Windows® 2000 Professional with Service Pack 4
- Microsoft® Visual C++® 6.0 with Service Pack 5
- Dialogic® System Release 6.0 PCI for Windows®
- Dialogic® DM/V960A-4T1 Media Board, with a crossover cable connecting the first and second T-1 spans. The board has a 5ESS ISDN firmware load, as distributed in the enhanced voice media load file for the DM3 architecture.

Downloads

Two demo programs are available for use with this application note:

- **SampleService**
- **Win32App**

These program demos enable you to understand the techniques explained in this document and also provide additional implementation information. See the *For More Information* section for the current location to download the Zip file containing these demo programs.

Architecture

A telephony application using Dialogic Global Call Software operates as a process that requires minimal operating system interaction to shut it down. Far fewer complications occur if the shutdown stimulus comes from the application itself through a `gc_Stop()` API call rather than as a shutdown (kill) signal or message from the operating system. To accomplish this, an architecture, such as the one illustrated in Figure 1, can be used for controlling the application from a system service.

The service control process for the telephony application functions as an intermediary between the service management console and the telephony application. To control the application from a system service, install the telephony application as a Win32 service rather than the telephony application itself.

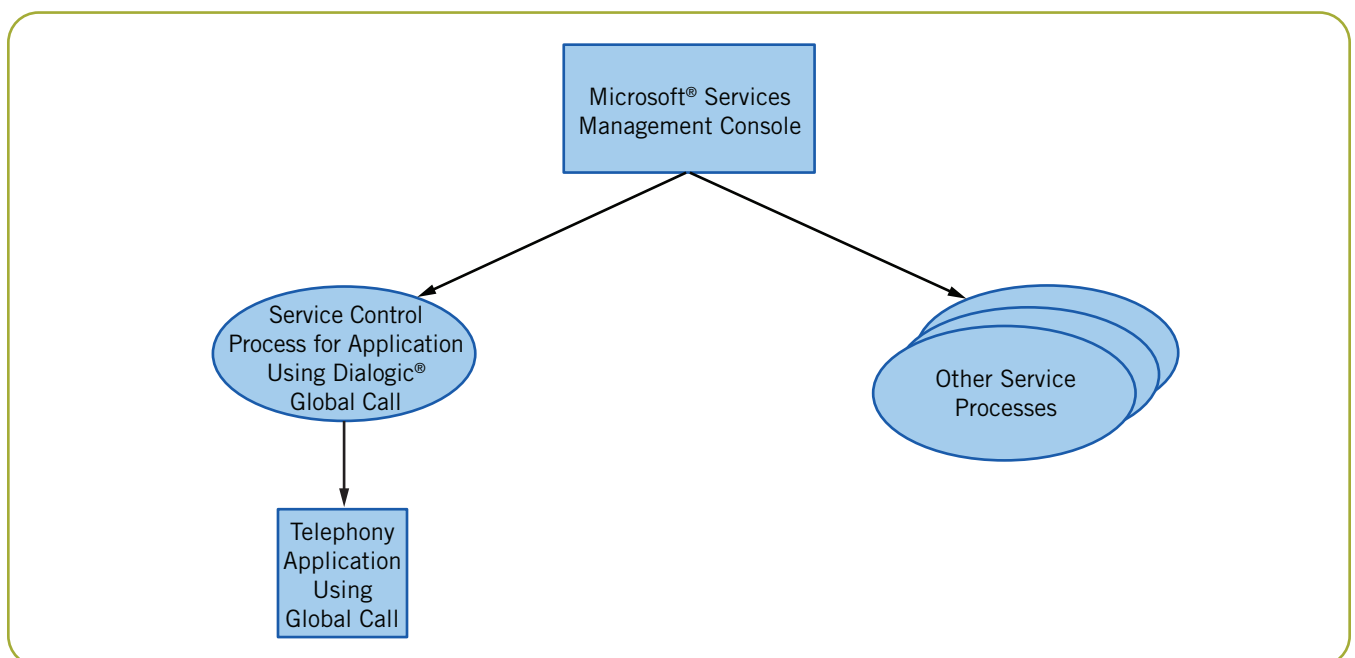


Figure 1. Architecture for Controlling an Application from a System Service

Service control operations, such as Start, Stop, and Restart, are initiated from the Windows® service management console and sent to the service control process. These operations result in actions that start and stop the telephony application, which resides in its own Win32 process.

Service Control Process

The service control process is shown in Figure 2 and operates as follows:

1. The service control process, in its main Run () method, issues CreateProcess () to start the telephony application using Global Call Software, and its process ID is saved for later use. The service control process then remains in a GetMessage () loop until forced to break out of it.
2. On receiving a SERVICE_CONTROL_STOP message from the management console, the service control

process sends a WM_QUIT message to itself using PostMessage () .

3. The GetMessage () loop is exited, and EnumWindows () is called. This system call invokes a handler for each window in the system, including the window running the application using Dialogic Global Call Software.
4. Each handler matches the saved process ID of the application using Dialogic Global Call Software against the ID of the application running in the window. When the correct window is found, shutdown can begin.
5. A message from the Windows® operating system in the WM_USER (application-defined) range is sent to the correct window, and the application running in the window interprets this event as a signal to begin shutting down.

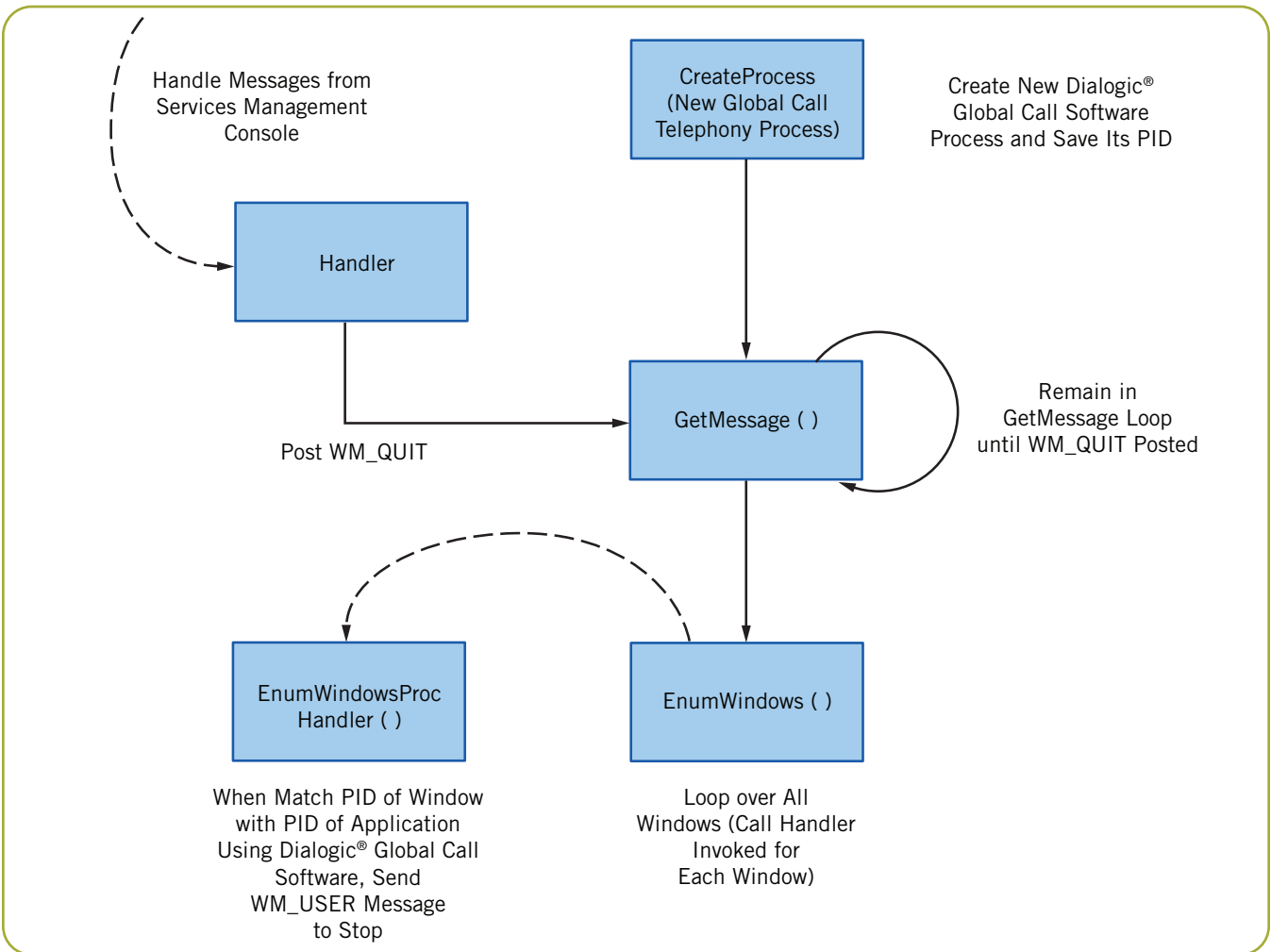


Figure 2. Service Control Process

Avoiding Simultaneous Shutdown

The Win32App demo program described in this document is a variation on the Dialogic Global Call API demo program, gc_basic_call_model, which is discussed in the *Global Call API Demo Guide* (see the *For More Information* section).

The gc_basic_call_model application has been modified so that it runs as a Win32 application rather than as a console application, as it did in the original demo. The modification allows the application to receive window

manager events, which are used as an Inter-Process Communications (IPC) mechanism. An event is sent from the service control process (SampleService) to the sample application (Win32App) to signal the application using Dialogic Global Call Software to shut itself down cleanly.

If the application using Dialogic Global Call Software were a simple console application, this strategy would fail. Sending a WM_CLOSE to the external console process results in a signal that is caught by the application console

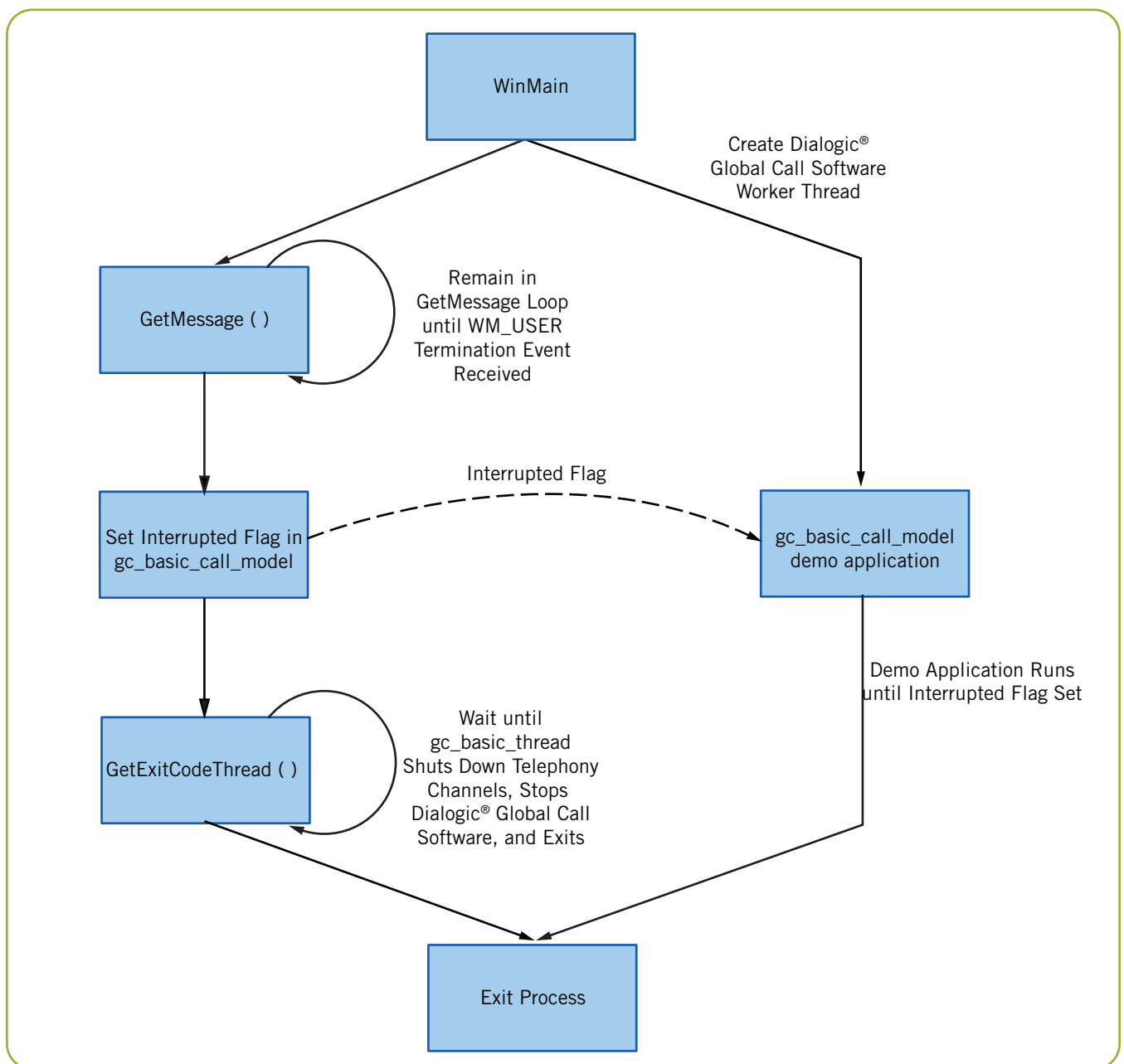


Figure 3. gc_basic_call_model as a Win32 Application

control handler set by `SetConsoleCtrlHandler()`. The shutdown process would begin, but a problematic interaction with the underlying Dialogic Global Call API libraries would occur. The console would direct the Dialogic Global Call API libraries to shut down, but there would be no coordination between the console shutting down the libraries and the application also shutting down the libraries with `gc_Stop()`. The likely result would be a “hang” in the execution of the `gc_Stop()` function.

To avoid simultaneous shutdown activities and the resulting instability, the event that initiates the shutdown must exist in the normal operating-system-to-application-event range. If this technique is used, signaling is strictly between the two processes, and will not have undesirable effects. A Windows manager event value of “WM_USER + 1, #defined as SERVICE_TERMINATION” in both applications, accomplishes this.

An illustration of how `gc_basic_call_model` operates when Win32 is used is shown in Figure 3.

The `main()` function in `gc_basic_call_model` is replaced by a `WinMain()` with changes to threading and signaling. The result is:

1. A worker thread that encapsulates the normal `gc_basic_call_model` is spawned using `CreateThread()`.
2. A `GetMessage()` loop is entered, and `WinMain()` remains there, checking messages received until it sees the termination message sent from the service control process.
3. An interrupt-received flag, monitored by the `gc_basic_call_model` thread, is set. `Gc_basic_call_model` then enters its shutdown sequence, closing individual channels and devices, and finally issuing a `gc_Stop()`.
4. While the `gc_basic_call_model` thread is shutting down, its exit status is monitored in `WinMain()` until the thread is no longer active.
5. When `WinMain()` exits, the entire process ends.

For more information, refer to the Win32App and SampleService demo programs.

Using an application-defined shutdown signal passed between the service and telephony processes can be accomplished by using IPC mechanisms other than the one shown in this application note. Other possibilities include shared memory, system semaphores, and named pipes.

System Setup, Build, and Execution of Sample Code

To execute the SampleService and Win32App sample programs, complete several tasks, including setting up the system, performing a build, and running the sample code.

Note: Although the demo discussed in this application note uses Dialogic® DM3 Media Boards and System Release 6.0 PCI for Windows, it is understood that the programming, Windows system services setup, and usage are applicable to the Dialogic® HMP Software-based Windows® system as well.

Setting up the System

The basic system setup consists of the following:

1. Install one of the Windows operating systems supported by Dialogic System Release 6.0 PCI for Windows.
2. Install System Release 6.0 PCI for Windows.
3. Install a Dialogic® DM/V480A-2T1 Media Board (supports T-1) or Dialogic® DM/V960A-4T1 Media Board (supports T-1) with a back-to-back crossover cable (RJ-48-C terminated cable where pin 1 connects to pin 4, pin 2 connects to pin 5) between spans one and two.
4. Configure the installed Dialogic® board using the Dialogic® Configuration Manager (DCM), selecting the `ml2_dsa_5ess` (DM/V480-2T1) or `ml2_qsa_5ess` (DM/V960-4T1) as the media load for the FCD/PCD files. Note that the files themselves are usually located in the `C:\Program Files\Dialogic\data` directory.
5. In the `.config` file that corresponds to the `.pcd` and `.fcd` files, locate the sections marked [CCS.1] and [CCS.2]. Parameter `0x17` is set to 0 in both, indicating that the ISDN spans operate in User (TE) mode, and expect a Network (NT) connection on their other end. Set one of the spans to Network mode, and the remains in User mode as follows:

```
[ CCS.1]
Setparm=0x17,0      # Network Mode.1=
NETWORK  0=USER
      .
      .
```

```
[ CCS.2]
Setparm=0x17,1      # Network Mode.1=
NETWORK  0=USER
      .
      .
```

Note: For Windows operating systems, the FCD file is automatically created when the PCD file and modified CONFIG file are downloaded to the board.

7. Start the system using DCM.

Building the Applications

Microsoft® Visual C++ 6.0 is used to build both SampleService and Win32App. Workspace and project files (.dsw and .dsp) are provided for each application.

Begin by building Win32App in Visual C++ 6.0.

Next, compile SampleService. Before doing so, set the two #defines at the top of the source file to appropriate values:

- **#define GCAPP_COMMAND_LINE** — The full path to the executable (for example, “C:\AppnoteExamples\Win32App\Debug\Win32App.exe”). Note that double backward slashes are used.
- **#define GCAPP_DIRECTORY** — The full path to the directory in which the gc_basic_call_model.cfg (.config file) can be found (for example, “C:\AppnoteExamples\Win32App”).

Running the Applications

Once SampleService is built, install it into the Windows service manager by executing `SampleService -Service` in its Debug directory. When the service manager is started, SampleService appears in the service listing.

SampleService can be uninstalled by executing `SampleService -UnregServer`.

A visible window on the desktop is needed to run the application using Dialogic Global Call Software. Configure the window by doing the following:

1. Open the service manager.
2. Double click on “SampleService”.
3. Go to the “Log On” tab.
4. Check off “Allow service to interact with desktop”.
5. Select “OK” to save.

Now start SampleService from the service manager by right clicking on the SampleService line and selecting “Start”. A window running Win32App appears. Two ISDN channels running back-to-back are configured, and the counters indicating the number of inbound and outbound calls handled increment as calls are made.

The normal log information detail produced by gc_basic_call_model is available in the log files in the same directory as the gc_basic_call_model.cfg file. Problems in starting Win32App from SampleService can be diagnosed from errors and events that appear in the application log in the system event viewer.

To stop SampleService from the service manager, right click on the SampleService line and select “Stop”. The Win32App window displays a “stopping” message, and then disappears. A clean exit should be noted in the inbound and outbound log files.

Acronyms

DCM	Dialogic Configuration Manager
FCD	Feature Configuration Description
HMP	Host Media Processing
IPC	Inter-Process Communication
PCD	Product Configuration Manager

For More Information

A Zip file containing the SampleService and Win32App demo programs can be downloaded at <http://www.dialogic.com/goto/?10684>

Dialogic® System Release 6.0 PCI for Windows® — <http://www.dialogic.com/manuals/sr60winpci/>

Dialogic® DM/V480A-2T1 Media Boards, Dialogic® DM/V600A-2E1 Media Boards, Dialogic® DM/V960A-4T1 Media Boards, Dialogic® DM/V1200A-4E1 Media Boards — http://www.dialogic.com/products/tdm_boards/media_processing/

Global Call API Demo Guide — http://www.dialogic.com/manuals/docs/globalcall_demo_v2.pdf

To learn more, visit our site on the World Wide Web at <http://www.dialogic.com>.

Dialogic Corporation
9800 Cavendish Blvd., 5th floor
Montreal, Quebec
CANADA H4M 2V9

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH PRODUCTS OF DIALOGIC CORPORATION OR ITS SUBSIDIARIES ("DIALOGIC"). 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.

Dialogic may make changes to specifications, product descriptions, and plans at any time, without notice.

Dialogic is a registered trademark of Dialogic Corporation. 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.

Microsoft, Windows, and Visual C++ are a registered trademarks 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. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement their concepts or applications, which licenses may vary from country to country.