Application Note

# Media Server Solution Recipe: Conferencing Applications

## Executive Summary

This application note describes a conferencing application developed using the Dialogic® Communications Services Framework (CSF). (See http://sourceforge.org/projects/commsvcfw).

This note covers conferencing in the CSF from an application point of view. General application call flow concepts are covered first and apply to any application written using the CSF. The conference server demo application is then examined from both an external (user) point of view and, in more internal detail, through looking at the application state machine. Finally, instructions are given on how to build, configure, and run the demo application on both the Windows® and Linux operating systems.

Implementations of specific conferencing features using different Dialogic® products are covered in other application notes available at http://www.dialogic.com/products/applnote.htm.

## Table of Contents

## Introduction

This application note will help the reader understand the technical details of an object-oriented approach to conferencing using Dialogic® hardware and software. It can be used, along with the sample code (see http://sourceforge.net/projects/commsvcfw/) it describes, as a basis for building a production-quality conferencing server application. It will also help the reader learn and understand how to use the Dialogic® API and resources when building any conferencing application.

The reader is assumed to be familiar with object-oriented programming and conversant with basic telephony concepts. For more information on these topics, see For More Information and visit the Dialogic Web site at http://www.dialogic.com/support/helpweb/dxall/.

## Communication Services Framework Application Flow

Any CSF-based application will have its application logic based in a set of objects known as a call flow. The first section of this note describes some principles that apply to any call flow.

This set of objects is contained in a call flow directory. A call flow may be used with various telephony network interfaces, and the logic follows the standard Answer Demo shipped with Dialogic® system releases:

1. Accept an inbound call
2. Play a prompt
3. Record a message
4. Play the message back
5. Hang up
6. Make an outbound call
7. On answer, repeat the prompt/record/playback cycle

Demonstrating a conference server requires a different, more complicated application; however, the way a CSF call flow is used is identical.

Since a call flow is used to demonstrate telephony applications, its on-screen user interface is very simple: a console application with no GUI. Keeping the on-screen interface uncomplicated also helps to achieve compatibility with the Windows® and Linux operating systems and simplicity in areas that are not concerned with telephony.

There are two main parts to the CSF application layer. The first application component handles startup, initialization, and shutdown. The second component, an object-oriented finite state machine, specifies the application logic itself. More information on the CSF design and usage can be found at http://sourceforge.net/docman/?group_id=92480.

Note that there is no overwhelming reason that an asynchronous, objected-oriented state machine must be used to drive the application logic. The application layer is loosely coupled to the rest of the framework and could easily be replaced with an application logic driver of the reader's choice. This particular application model, however, has been found to be useful in writing demo programs for the CSF. It contains a good example of an object-oriented state machine.

CSF demos do not typically contain any non-Dialogic technologies, such as a complete GUI, DBMS interaction, or email integration, that would be needed for a complete commercial application.

Before describing how the conference application works from an external point of view, initialization, shutdown, and the object-oriented state machine will be examined.

## Main Program Module

All CSF demo applications use a similar simple main() module:

- The system event processor is created.
- The system event/error logger is created.
- The application object (described below) is created.
- This main thread blocks waiting for an interrupt signal (control-C from keyboard).
- On receipt of the signal, the application is cleanly shut down. Shut-down consists of destroying all application objects created either in main() function or in the AppObj described below.

## AppObj

In the case of the conference application, there is a single application object called ConfAppObj. ConfAppObj implements network, media, and conferencing device opening and initialization as follows:

- Network and voice devices for each channel are created as a CSF-managed object. Devices are specified in a configuration file.

- One ConfCallflow object per network channel is created as a CSF-managed object.
- DSP conferencing resource objects are created as CSF-managed objects. Note that these objects are abstractions of either hardware-based (DSP) conferencing facilities or, in the case of Dialogic® Host Media Processing (HMP) Software, of software-based facilities. Again, they are specified via a configuration file.
- States for the application are instantiated and each Callflow state machine is set to an *uninitialized* state.
- A single instance of the conference manager object is created to manage the specified set of conferencing resource objects.

## Application State Machine Implementation

Applications linked directly to the CSF (as opposed to those that may have an intervening interpreter/adapter layer) are always asynchronous and may be best modeled as state machines. The application state machine follows the State Machine pattern used elsewhere in the CSF (for example, for voice or network devices). For a comprehensive explanation on pattern use in object-oriented programming, see *Design Patterns: Elements of Reusable Object-Oriented Software* [Gamma, Helm, Johnson, and Vlissides].

The operation of the application state machine can be simply explained. The application, or stateful object, is always in one particular state. Within this state, there are a number of valid events of which it may be notified. When one of these events is received, a state transition is made (the next state selection being dependent upon the triggering event) and a corresponding action is taken. The action invoked by the original event will likely result in another event. For example, a play completion event invokes a routine that starts a record. This eventually generates a completion event of its own, which will be a valid event for the new state.

This design pattern allows for states that handle very few events, often only two or three events each. This will lead to a large number of states for any sizable application and will require managing a corresponding number of C++ files. However, this design pattern also makes it easy to understand the application, modify its flow, and debug in the event of problems.

State definition begins with the abstract base class, AdlgcConfCallflowState. Here, virtual actions taken upon receiving application-related events are defined; however, no real actions are taken in the virtual functions. Rather, they default to unhandled and simply log an error message. An example of this would be a call offered event received on a channel that was in a playing state. The application state machine should not react to this, since it is not a valid event for this state. Its occurrence should be logged for future diagnosis, particularly if it is not a one-time event.

Concrete classes for each application state are then defined for each such state that is identified. Each event handler in the concrete state class overrides the base event handlers defined in the abstract state class and provides a place to add programmatic responses taken when the particular event occurs.

There is also an active call substate, AdlgcConfCallflowActiveState. This substate contains a set of event handlers that are always available when a call is active. These events/actions are OnCallDropped and OnCallDisconnected, handling hang-ups on both sides of the call. This simplifies all active call states by moving repetitive event handling needed by many states into an abstract base class that they all inherit.

Figure 1 shows the relationship between the abstract states and a typical concrete state based on them.

Using the previous state as an example, the following steps through an example of a transition scenario in the conference server:

1. While in an idle state, a channel responds to a *CallAnswered* event in its *OnCallAnswered* action method where it first transitions to a *CollectMenu1Choice* state.
2. A play is initiated with the prompt used for Menu1. When the play is terminated, either by the full prompt having been played or by a DTMF tone for the menu choice having been entered, the *OnPlayComplete* action method launches a *GetDigit* operation to fetch the menu choice.
3. *GetDigit* either gets a pre-existing DTMF value out of the digit buffer (if a digit detection was the termination condition for the play) or waits until the tone is detected and returns its value.
4. Other valid events have been specified in the active call substate to allow for reacting to a *CallDisconnected* or *CallDropped* event by cleaning up the application call flow for the channel.

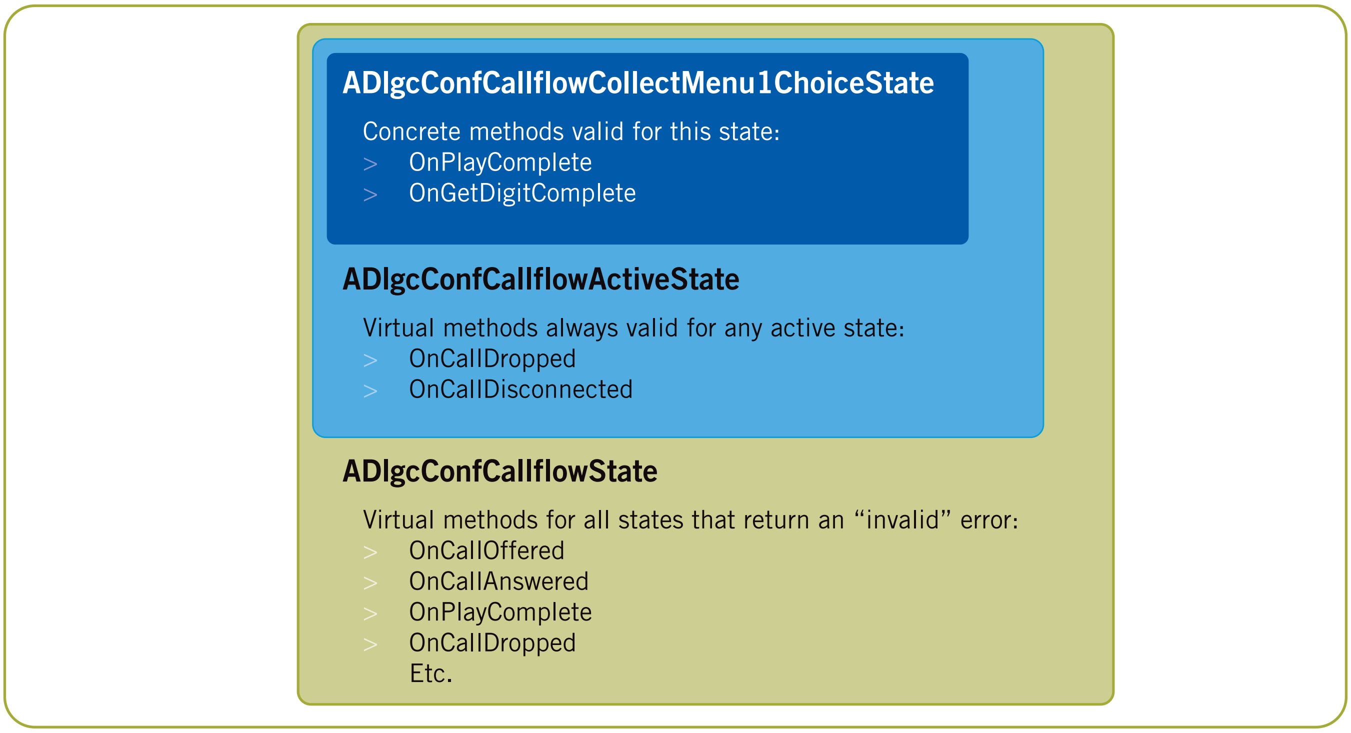
ADlgcConfCallflowCollectMenu1ChoiceState
Concrete methods valid for this state:
> OnPlayComplete
> OnGetDigitComplete
ADlgcConfCallflowActiveState
Virtual methods always valid for any active state:
> OnCallDropped
> OnCallDisconnected
ADlgcConfCallflowState
Virtual methods for all states that return an "invalid" error:
> OnCallOffered
> OnCallAnswered
> OnPlayComplete
> OnCallDropped
Etc.

*Figure 1. Valid Actions for Active Application State AdlgcConfCallflowCollectMenu1Choice*

## Conference Server Demo Application

### Conferencing Boards and APIs Supported

The conference application call flow demonstrates the functionality currently implemented using the CSF conference server objects. Refer to "Media Server Solution Recipe: Conferencing Objects," another Dialogic® conference server application, which explains the basic conferencing objects themselves. The code developed for this application note is available on the SourceForge website and you can ask questions about the code on the Dialogic Helpweb forums. For a current copy of the entire application note, contact Dialogic support.

The demo application uses standard interactive voice response (IVR) voice prompting and DTMF responses to elicit menu choices and information from the caller.

Three main features are shown:

- Ad hoc conference setup
- Join an already active conference
- Schedule a conference for a future time

The following Dialogic® conferencing products are supported:

- Dialogic® Modular Station Interface (MSI) boards with the MSI conferencing API (based on using MSI boards from Dialogic).

**Note:** The MSI product line has been retired. It has been replaced by the newer Dialogic® HDSI Station Interface Boards and Dialogic® DISI Switching Boards. Some modifications may be necessary if you are using a later version.

- The Dialogic® Springware-based DCB Series Conferencing Boards with DCB API have been retired. The following are available with the DCB conferencing API (some modifications may be necessary if you are using a later version):

  — Dialogic® DM3-based conferencing boards. This is a software emulation of the original Dialogic® DCB board, running on the Dialogic® DM3 media resource family of boards.

  — Dialogic® Host Media Processing (HMP) Software. This is a software emulation of the Dialogic DCB board, running as software on a host Intel architecture processor, such as an Intel Pentium or Xeon processor.

Since this application runs on conferencing technologies, it also forms a set of migration paths between the technologies for conferencing services.

When this application was created, the MSI station sets as conference members were not yet supported. The board was used only for its conferencing abilities in this application.

The type of conferencing supported is specified in the CConfObjApp class. CConfObjApp initializes the appropriate DSP resources according to the conferencing devices specified in the application's configuration file. For information on the format and contents of the configuration file, see the section Configuring the Conference Application.

### Network Interfaces Supported

There are two different network interfaces supported with the conference application:

- Time Division Multiplexed (TDM) using the Dialogic® Global Call control API. Any protocol supported by Global Call should work with the conference server, including Internet-based protocols as they become available. Testing has been done with 5ESS ISDN.
- Voice over Internet Protocol (VoIP) using host-based Session Initiation Protocol (SIP) call control and Dialogic® DM/IP Board-based Real Time Protocol (RTP) streaming. The Vovida Open Source stack is used in this example as the SIP user agent.

As with the conferencing DSP resources, the network interfaces are initialized according to the entries specified in the application's configuration file. For information on the format and contents of the configuration file, see the section Configuring the Conference Application.

### Application Options: External View

An external (user-oriented) view of the call flow for an inbound call to the conference server is pictured in Figure 2. The operation of each of the three major options follows.

### Ad Hoc Conference Setup

An inbound call is able to activate and participate in a conference, which can then be joined by other conferees. After picking this choice, the caller is asked to specify the number of conference participants. Note that only a single digit is collected, limiting the number of participants to 9. Moving beyond a demo, it would be necessary to enter larger numbers.

The CSF resource manager is queried as to the availability of on-board DSP conferencing resources. If sufficient capacity is present, the conference ID is spoken back to the conference initiator. The conference ID is also printed out as part of CSF logging. This ID begins at 100 when the server is first started, increasing by one for each subsequent conference. A single-user conference is created and available for call-in by the rest of the conference participants.

### Conference Scheduling

This option allows a caller to set up a conference at a future time for a selectable number of participants. When that time arrives, the application activates a conference ready for call-in by the requested number of conferees.

Times are entered as four digits in 24-hour time format. Conference start time is entered first, followed by end time. Times are validated as being later than the current time, with the end time being later than the start time. Note that the demo application only supports times in the current day. A usable application would require entering month and day with appropriate validations.

After times are entered, the size of the conference is requested. Again, only a single digit is collected for ascertaining the conference size, thereby artificially limiting the number of participants to nine. Availability of on-board DSP resources for conferencing is then checked.

If sufficient capacity is present, a conference is scheduled and the conference ID is spoken back to the conference requestor. It is also printed out as part of CSF logging. This ID begins at 100 when the server is first started and increases by one for each subsequent conference. Once the number of participants is obtained and the conference is scheduled, the application session terminates and the caller is disconnected.

Very little is done in this demo for maintaining conference scheduling based on network and conference resource availability. This sort of scheduling is not trivial and is left to the builder of a production application or a future version of this application note.

### Joining an Existing Conference

The option to join an existing conference is valid when a conference is already active, either because an ad hoc conference has been created or the time period for a scheduled conference has begun. The caller is asked to key in the ID of the conference to be joined. Validations are done to make sure that the ID belongs to an active conference and that the conference has not exceeded the requested size. If the validation tests are passed, the caller is added to the conference.

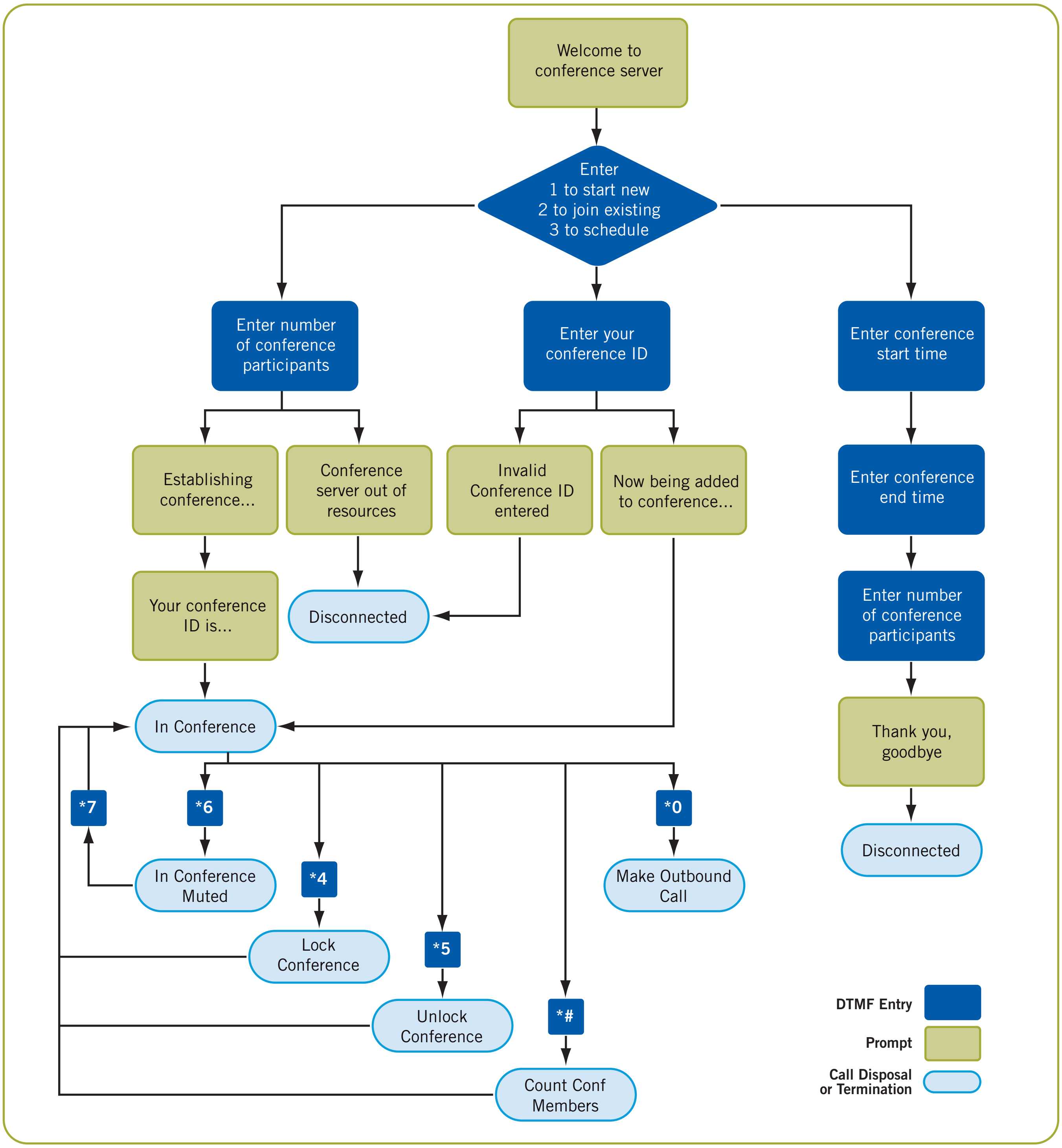
Welcome to conference server
Enter 1 to start new 2 to join existing 3 to schedule
Enter number of conference participants
Enter your conference ID
Enter conference start time
Establishing conference...
Conference server out of resources
Invalid Conference ID entered
Now being added to conference...
Enter conference end time
Your conference ID is...
Disconnected
Enter number of conference participants
In Conference
Thank you, goodbye
*7
*6
*0
Disconnected
In Conference Muted
*4
Make Outbound Call
Lock Conference
*5
Unlock Conference
*#
Count Conf Members
DTMF Entry
Prompt
Call Disposal or Termination

*Figure 2. Conference Application Call Flow*

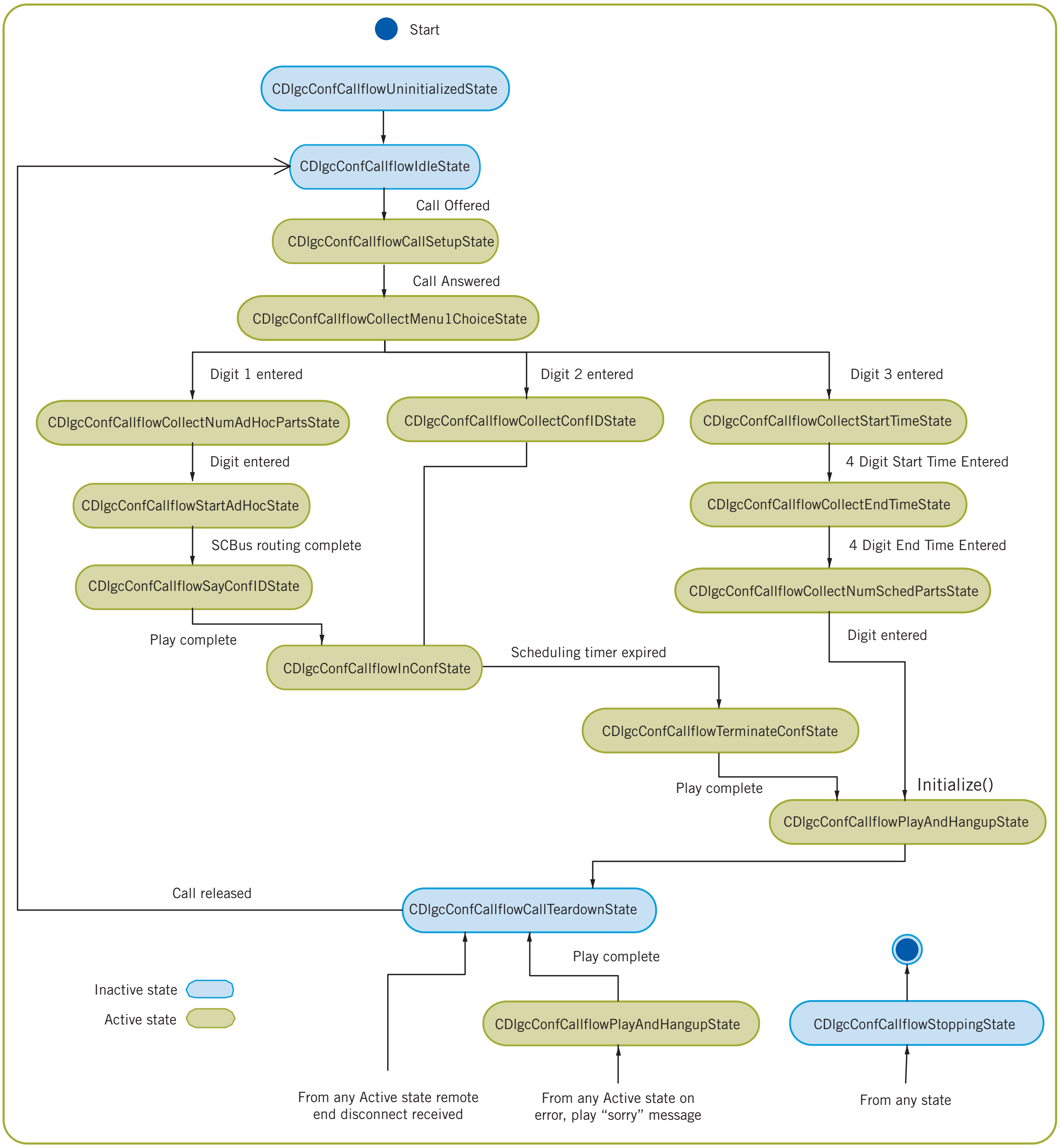
Start
CDlgcConfCallflowUninitializedState
CDlgcConfCallflowIdleState
Call Offered
CDlgcConfCallflowCallSetupState
Call Answered
CDlgcConfCallflowCollectMenu1ChoiceState
Digit 1 entered
Digit 2 entered
Digit 3 entered
CDlgcConfCallflowCollectNumAdHocPartsState
CDlgcConfCallflowCollectConfIDState
CDlgcConfCallflowCollectStartTimeState
Digit entered
4 Digit Start Time Entered
CDlgcConfCallflowStartAdHocState
CDlgcConfCallflowCollectEndTimeState
SCBus routing complete
4 Digit End Time Entered
CDlgcConfCallflowSayConfIDState
CDlgcConfCallflowCollectNumSchedPartsState
Play complete
Digit entered
CDlgcConfCallflowInConfState
Scheduling timer expired
CDlgcConfCallflowTerminateConfState
Play complete
Initialize()
CDlgcConfCallflowPlayAndHangupState
Call released
CDlgcConfCallflowCallTeardownState
Play complete
Inactive state
Active state
CDlgcConfCallflowPlayAndHangupState
CDlgcConfCallflowStoppingState
From any Active state remote end disconnect received
From any Active state on error, play "sorry" message
From any state

*Figure 3: Internal State Machine Used to Implement the Application*

The participants of a scheduled conference are terminated when the conference's time period expires. A warning message is played to all conferees before they are disconnected by the application. A full application might give several warnings and offer an option of extending the conference if sufficient conference resources were available.

## Options within a Conference

There are several options available to a conferee while a conference is in progress. These are selected via DTMF tone combinations as follows:

*4 **Lock** the current conference so that additional callers are not allowed access. Inbound callers are then informed that the conference is closed to further conferees and disconnected.

*5 **Unlock** the current conference to allow access. Note that the lock/unlock functionality is available to any conference participant. In a production application, there might be a designated moderator who would be the only one allowed to lock/unlock the conference.

*6 **Mute** the transmit side of the conferee. It is still possible to hear the ongoing conference.

*7 **Unmute** the transmit side of the conferee. The conference member will return to the member's usual full duplex mode.

*0 **Outdial** to an operator. This option initiates an outbound call to a number specified in the configuration file. On answer, the standard conferencing application is started. The party can then choose option 2 to join the ongoing conference and must enter the conference ID.

*# **Count** and report on the number of participants currently in the conference. Only the conferee requesting the count hears the message.

### Application Call Flow: Internal State Machine

The state transition diagram in Figure 3 shows the internal state machine implementation behind the user options of Figure 2. A separate state object represents each state in the application. Events that cause a transition to the next state are shown. Three transitions from multiple states are shown with a single arrow for clarity:

- Any *active* state (pictured in green) will react to a remote end disconnect event.
- Any *active* state (pictured in green) will react to an application error (including those from the Dialogic® API) by playing a message and disconnecting the caller.
- Any state will react to an *application stopped* event.

## Building the Conference Application

Before building the CSF-based conference application, several base packages used by the application must be added to the platform:

- **Boost Threading Libraries** — This is an OS neutral (Linux/Windows®) set of libraries meant to extend and improve on the Standard C++ libraries. See the file Boost_win.txt (Windows®) or Boost_linux.txt (Linux) in the CSF source code directory for installation instructions. The libraries themselves can be obtained from http://www.boost.org.
- **Vovida Open Source SIP** — The conference server may be configured for either Dialogic® Global Call/ISDN, Global Call/SIP, or VoIP with external SIP call control. The external SIP package used is from Vovida, and may be downloaded from http://www.vovida.org. Under the Linux operating system, follow the standard installation and build procedure outlined in the README file for the Linux download.

  Vovida installation for the Windows® operating system is described in the vovida_win2k.txt file. Linux instructions are in vovida_linux.txt. Both can be found in the CSF source code directory
- **CURL Libraries** — The CURL libraries contain date/time functions that are used for conference scheduling. The libraries are part of the standard Red Hat Linux distribution, but must be downloaded and installed on the Windows® operating system. They are available at http://www.curl.haxx.se and their installation is described in the Curl_win2k.txt file that can be found in the CSF source code directory.
- **Log4CPlus Libraries** — The Log4Cplus library supplies an error/event logging subsystem used by CSF. It may be downloaded from http://log4cplus.sourceforge.net/.

  Log4Cplus installation for the Windows® operating system is described in log4cplus_win2k.txt, and Linux instructions are in log4cplus_linux.txt.
- **Xerces XML Parser** — Although the conferencing application does not need the XML parsing facilities in the Xerces library, other CSF modules that are linked in require it. It may be found at http://xml.apache.org/xerces-c.

  Xerces installation for the Windows® operating system is described in the xerces_win2k.txt file and Linux instructions are in xerces_linux.txt.
- **Dialogic® System Release or Dialogic® Host Media Processing (HMP) Software** — Use an appropriate release for the operating system being used — either a standard Dialogic System Release, or a Dialogic HMP Software release.

## Linux Operating System

The conferencing server is built using the *makefile* supplied in the ConfObj directory. If the additional packages and the Dialogic System Release (or Dialogic

HMP Software) have been properly installed, the server will compile and link without problems.

**Note:** There is a set of environment variables in the makefile that parallel those that are recommended when installing the base packages. Some adjustment may be necessary, depending on where the packages were installed.

### Windows® Operating System

The project files for a Visual C++ 6.0 application are provided in the ConfObj directory. Make sure the additional packages have been properly installed and environment variables mentioned in the installation instructions have been set before building and running the application.

## Configuring the Conference Application

Network, voice, and conferencing devices used by the application must be specified in a standard CSF configuration file. The file, named ConfObjCfg.txt, is located in the ConfObj directory. Two types of devices are supported: DEVICES and CONFDEVICES.

The generic DEVICES type specifies both the network and voice devices available for each channel. Note that a voice device is not optional; one must be assigned to each network device. The column definitions for the DEVICES type follow:

| Column | Entry | Description |
|---|---|---|
| 1 | Group ID | Not used (enter *none*) |
| 2 | Protocol | Several protocols are supported:<br>gc:isdn - ISDN under Global Call<br>gc:H323 - H323 under Global Call<br>gc:SIP - SIP under Global Call<br>Vovida SIP using IPML RTP media is specified in the following format:<br>ipm:host_IP_address:SIP_Port_number |
| 3 | Network Device Name | For Global Call ISDN: dtiBxTy<br>For Global Call SIP/H323: iptBxTy<br>For DM/IP: ipmBxCy |
| 4 | Voice Device Name | dxxxBxCy |
| 5 | Default Address for Outbound Call | Not used |
| 6 | Other Info | For Vovida SIP using IPML RTP media and Global Call ISDN: not used<br><br>For Global Call SIP and H.323: CallInfo_X. This refers to an entry in the .config file [CALLINFORMATION] that specifies a set of codecs and DTMF type. The codecs are in turn described under[CODEC]. |

The CONFDEVICES type specifies a DCB or MSI conferencing DSP device. Column definitions for the CONFDEVICES type follow:

| Column | Entry | Description |
|---|---|---|
| 1 | Conference Board/API type | MSI or DCB. Note that all entries MUST be of one type or the other. A conference server with mixed boards is not supported. |
| 2 | Conferencing Device | For MSI, *msiBx*<br>For DCB, *dcbBxDy* |
| 3 | Other Info | Not used |

### Sample Configuration Files

The configuration file that is shipped with the source code is set up for four channels of SIP under Dialogic® Global Call. The protocols described above are also in the file, commented out.

### Additional System Configuration and Setup

If VoIP/SIP with a Dialogic® DM/IP Board is used with the conferencing application, some configuration of the DM/IP Board is necessary to set up the board for in-band DTMF. This is done as follows:

- System service must be in a *stopped* state, and can be done using *dlstop* for Linux, and using the Dialogic® Configuration Manager (DCM) for Windows® operating systems.
- Edit the .config file used for the DM/IP Board. (This would typically be a file such as ipvs_evr_isdn_5ess_311.config.) Change the PrmDTMFXferMode to 1 for in-band DTMF.
- For Windows® and Linux operating systems, the FCD file is automatically created when the PCD file and modified CONFIG file are downloaded to the board.
- Start the telephony boards using *dlstart* for Linux, and using the DCM for Windows® operating systems.

## Running the Conference Application

The application is run without specifying any parameters on the command line.

When configured for ISDN, calls into the system should be delivered through an ISDN channel bank test system or some other ISDN connection.

When configured for SIP, almost any available SIP softphone or SIP phone set may be used.

Another method for providing compatible calls to the conferencing server is to use a CSF-based application (either one of the media gateways or the satellite PBX) to generate SIP/RTP calls.

## Potential Conference Application Improvements

Additions to the conferencing server application may include:

- Further enhancement of the “standard” * command set found on most conferencing servers
- Remote and non-DTMF invocation methods for the * commands
- Inclusion of DSP spanning capabilities on DM3 boards
- Addition of the CNF API
- Performance and capacity characterization for specific recipes

## References

[Gamma, Helm, Johnson and Vlissides] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Orientated Software*, Addison-Wesley, 1997.

## Acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **Asynchronous** | A type of architecture where telephony API commands do not block waiting for completion. Instead, the application is notified when completion occurs and it reacts appropriately. |
| **Callflow/ConfCallflow** | The application logic part of a Framework 2-based telephony application |
| **CSF** | Communications Services Framework |
| **CURL** | A client that groks the URLS |
| **DBMS** | Data Base Management System |
| **DCB** | Dialogic Conferencing Board |
| **DCM** | Dialogic Configuration Manager |
| **DSP** | Digital Signal Processor |
| **DTMF** | Dual Tone Multi-Frequency |
| **IVR** | Interactive Voice Response |
| **MSI** | Modular Station Interface |
| **RTP** | Real Time Protocol (Internet Engineering Task Force RFC 1889) |
| **SIP** | Session Initiation Protocol (Internet Engineering Task Force RFC 2543) |
| **Synchronous** | A type of architecture where telephony API commands block waiting for completion. On completion, the application proceeds to its next step. |
| **TDM** | Time Division Multiplexing |
| **URL** | Universal Resource Locator |
| **VoIP** | Voice over Internet Protocol |

## For More Information

**Dialogic® System Release Software Downloads for the Windows® and Linux Operating Systems** — http://www.dialogic.com/products/tdm_boards/system_release_software/default.htm

**Dialogic® Products** — http://www.dialogic.com

**Developer Resource Centre** — http://www.dialogic.com/forums/category-view.asp

**Red Hat Linux** — http://www.redhat.com

**CURL** — http://curl.haxx.se/

**Vovida SIP Stack** — http://vovida.org/

**Boost Threading Libraries** — http://boost.org/

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