Dialogic®

Application Note

MRCP V2 Client Library User's Guide

Executive Summary

This application note provides detailed information about the classes and commands that make up the MRCP V2 client library. Dialogic developed this library to provide a starting point for integrating MRCP-based applications with Dialogic® products to create next-generation speech solutions.

The client library and a sample application are available for download with this application note.

Table of Contents

Introduction 4

Client Library Overview 4

Client Library Feature Set 4

- Generic MRCP Commands 4
- TTS Commands 5
- ASR Commands 5

Client Library Architecture 6

- MrcpSystem Object 7
- Session Manager Object 8
- Session Object 8
- Signaling Objects 8
- RTP Manager Object 9
- RTP Object 9
- Audio Stream Object 9
- Command Objects 9
- Command ID Manager Object 9
- Handle Manager Object 9
- Task Processor Objects 9

Using the Client Library 10

Summary 11

Acronyms 11

References 11

Introduction

The Media Resource Control Protocol (MRCP) is a communications protocol that allows applications to control media service resources in the network. These resources reside on servers and provide support for speech synthesis, recognizers, verifiers, and identifiers.

MRCP is implemented in a client/server architecture. Client applications communicate requests for media resources to MRCP servers, which provide resources using MRCP signaling and commands. The MRCP servers respond and supply the media services requested. This document describes the use of an MRCP client library created by Dialogic engineers to assist in the development of MRCP client applications.

This application note serves as a reference for the Media Resource Control Protocol Version 2 (MRCP V2) Client Library (hereafter referred to as the "client library") and is intended to help developers understand the classes, commands, and operations defined within it. Developers may choose to customize and extend the client library to suit their individual requirements.

A Zip file containing the client library, a sample application, and other components can be downloaded at http://sourceforge.net/projects/openmrcpclient.

Client Library Overview

The client library provides commands for supporting automatic speech recognition (ASR) and text-to-speech (TTS) using MRCP V2. Support for other MRCP V2 functionality such as verification, authentication, and recorder functions could be added easily to the library because it is implemented using object-orientated techniques. Additionally, the client library was developed on the Microsoft Windows platform, but platform-specific application programming interfaces (APIs) were kept to a minimum to ensure easy porting to other operating systems such as Linux.

The client library does not support the means to control telephony operations, but rather relies on the application using the client library and Dialogic® APIs to provide the needed telephony functionality. For example, the application is expected to provide a mechanism to collect caller speech and play back the audio received. This implementation approach allows the client library to work across the Dialogic product line.

Status logging is provided via a logging object, with output currently being written to both the application's console window and to a log file. Addition of a configuration utility to control features such as logging is left as a task for the users of the library. Finally, while the library contains its own SIP and RTP implementations, Real Time Control Protocol (RTCP) support has not been included. RTCP packet reports are neither generated nor processed by the client library.

Due to the nature of MRCP, all commands execute asynchronously. All client library API commands return a success or failure synchronously when called. Success means that the library was properly started and is ready to process a command. The client library uses callbacks to call the client application when a command status value is received from the MRCP server.

Client Library Feature Set

The client library provides support for a subset of the ASR, TTS, and generic commands defined within the MRCP specification. Client library command state machines ensure that commands are sent to the server only when the MRCP session is in the proper state. This section provides a summary of the command set. The IETF MRCP specification contains additional information about each of the commands discussed. For a link to this document, see the *Reference* section at the end of this application note.

Generic MRCP Commands

Commands supported as MRCP Generic Methods in the MRCP specification include both **SetParams**() and **GetParams**().

- **GetParams**() — returns the MRCP parameters currently set for an MRCP session from the MRCP server. The client library implementation supports both returning all parameter values (wildcard query) and returning specific parameter values.
- **SetParams**() — sets the session parameters on the MRCP server. These session parameters are used for all MRCP commands unless overriding parameters are supplied in other relevant MRCP commands. For example, **SetParams**() can set a confidence level for ASR for the entire session while a **Recognize**() command can override the confidence level.

TTS Commands

The MRCP specification defines TTS as both speech synthesis and streaming playback of pre-recorded messages. TTS commands within the client library include: **Speak**(), **Stop**(), **BargeinOccurred**(), **Pause**(), **Resume**().

- **Speak**() — requests speech-based audio from the MRCP server. This command can include the text of the audio to be spoken, the URL of a file to be streamed, or the relative directory location of a file to be streamed. A variety oif XML formats can be used to format the TTS or streaming-file-play requests.
- **Stop**() — stops any synthesis or streaming-file play currently underway for a session
- **BargeinOccurred**() — used when the client application detects that a barge-in event has occurred. Events include both DTMF and speaker utterances. This command is typically used when the ASR and TTS resources are located on separate servers.
- **Pause**() — causes the MRCP server to pause any speech output currently in progress. This command works regardless of whether the audio is synthesized or is a stream from a pre-recorded file.
- **Resume**() — resumes previously paused audio from the MRCP server

ASR Commands

ASR is the recognition of audio by the MRCP server based on predefined parameters. Typically, a grammar (list of terms for recognition) is defined in advance of the request for recognition. ASR commands within the client library include: **DefineGrammar**(), **Recognize**(), **Stop**(), **GetResult**(), **StartInputTimers**(), **and Interpret**().

- **DefineGrammar**() — identifies the grammar to be used for recognition. Actual grammar can be defined, or the URI of a file containing the grammar can be specified.
- **Recognize**() — starts the MRCP server's recognition process for the session. Grammar to be used for the recognition can be passed to the server as part of this command; however, grammar compilation necessitated by specifying grammar in this command may impede the recognition process.
- **Stop**() — tells the MRCP server to stop the recognition process. This command is syntactically the same as the **Stop**() command discussed in the *TTS Command* section above.
- **GetResult**() — requests the results of a recognition request. Results are returned without using this command; however, using this command returns additional recognition information.
- **StartInputTimers**() — signals the MRCP server that a kill-on-barge-in prompt has completed. This command is frequently used when the ASR and TTS MRCP servers are located on different physical servers. Input timers are commonly used to notify the client when no input has occurred within the time allotted.
- **Interpret**() — requests a semantic interpretation of audio sent to the MRCP server. This command is similar to **Recognize**(), but only portions of the result relevant to acoustic matching are excluded from the result. This command can work in conjunction with **DefineGrammar**().

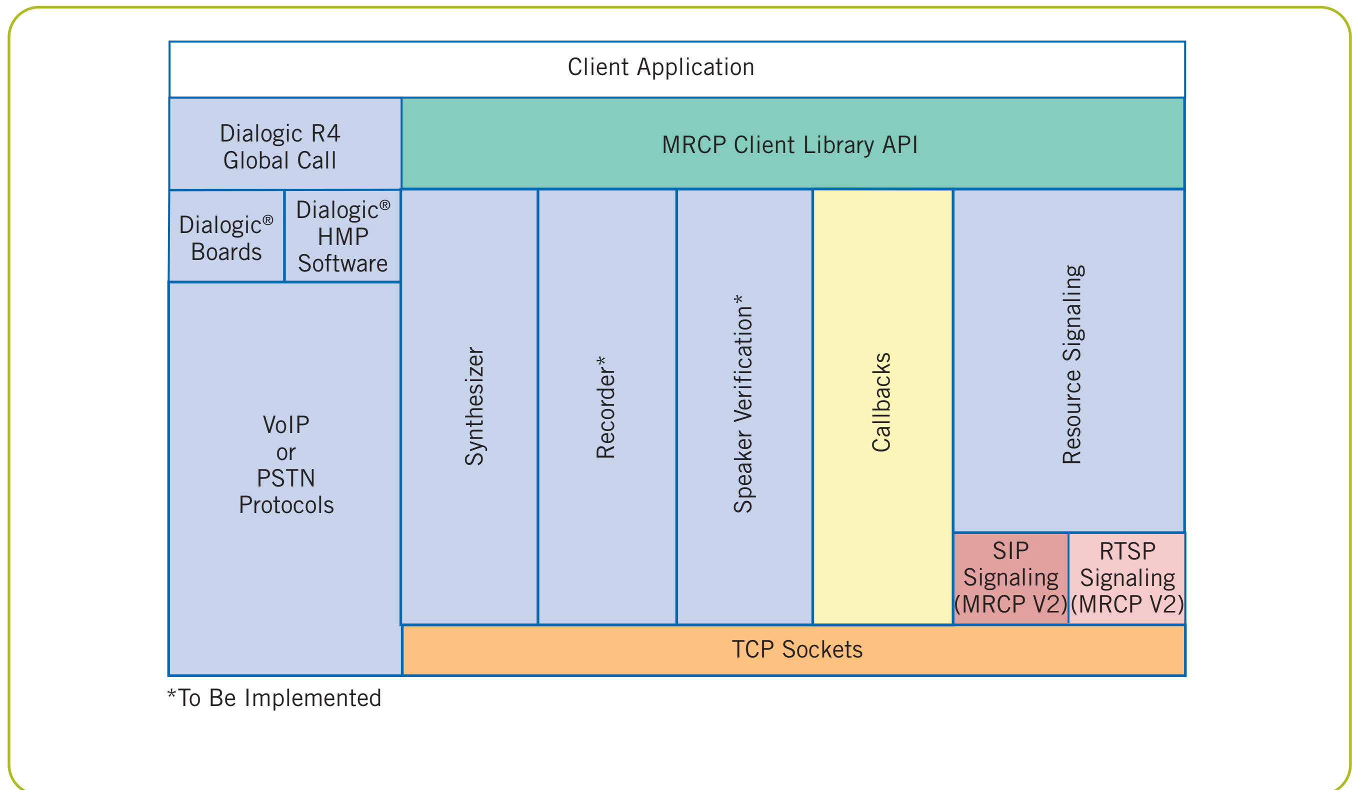

Figure 1. Client Application Architecture

Client Library Architecture

The client library is an object-oriented C++ implementation of the client library functionality defined in the MRCP V2 specification. Several of the design patterns defined in the book *Design Patterns: Elements of Reusable Object-Orientated Software* were used in the client library implementation. For more information about these sources, see the *Reference* section at the end of this document.

For an illustration of how the client library fits within a client application architecture, see Figure 1.

The API for the client library is contained within the MrcpSystem class. All client application commands for opening and closing sessions, working with media, or sending ASR or TTS commands are made via this class. The MrcpSystem class is an implementation of the singleton design pattern. As such, there is a single instance of the MrcpSystem object per application utilizing
the library.

All of the manager objects are invoked when the MrcpSystem object's start command is called by the client application using the library. Like the MrcpSystem object, the manager objects (Command ID Manager, Handle Manager, Session Manager, and RTP Manager) are singleton class implementations. Similarly, all of the manager objects are stopped when the MrcpSystem object's stop command is called. See Figure 2 for an illustration of the relationship of MrcpSystem to the handlers and other library components.

Each of the manager objects has a unique responsibility. They support object encapsulation and provide unique MRCP parameter values to the MRCP server.

- **Session Manager** — opens and closes sessions and notifies the MrcpSystem object when all sessions have exited on shutdown.
- **Command ID Manager** — assigns unique command IDs for each MRCP command and maps a callback routine to the command.
- **RTP Manager** — instantiates both the RTP and audio stream objects needed to support an MRCP session. Like the Session Manager, the RTP manager notifies the MrcpSystem object when all RTP and audio objects have exited during shutdown.
- **Handle Manager** — Assigns unique numerical values (handles) to be passed back to the client application in lieu of actual object pointers. A map of the handles

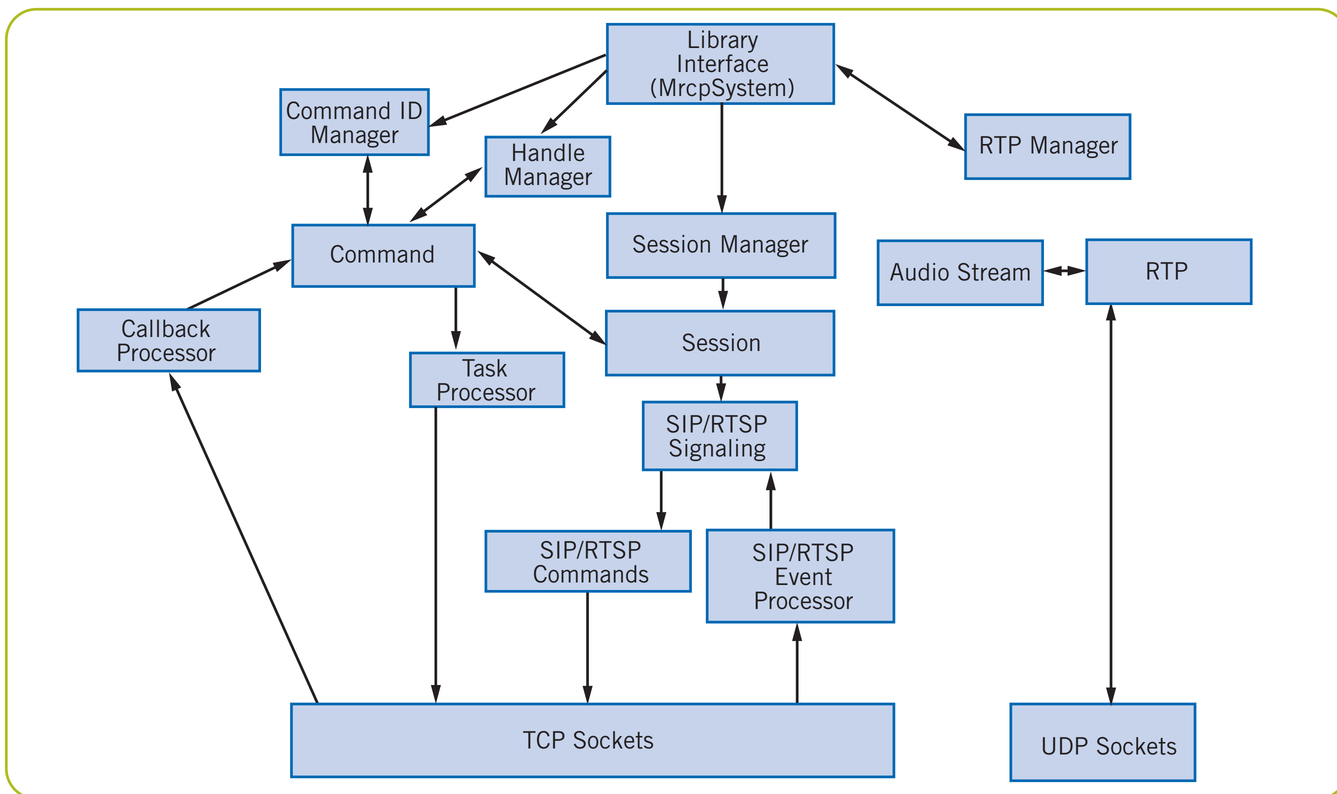

Figure 2. Client Library Architecture

to object pointers is maintained by the handle manager, as are the commands needed to maintain, assign, and retrieve from the map.

Communication between the various manager objects, the MrcpSystem object, and various dependant objects is accomplished with the Observer Design Pattern. This pattern implements a "publish/subscribe" architecture whose benefit is that the notifying objects do not need to know anything about the subscriber objects.

All API commands are made to the MrcpSystem object, and they are processed using the command design pattern in conjunction with the state design pattern. Command objects are instantiated for the API requested and queued for processing. Objects processed from the queue are then passed through the state design pattern where they are run if the library, session, and state are proper to support the command.

Additional details regarding each of the client library objects are presented in the following sections.

MrcpSystem Object

The MrcpSystem class commands are used to open, change, and close sessions and to support all MRCP commands. The procedure to open a session is:

1. The start command must be called with the MrcpSystem object's instance retrieved with the instance command.
2. The open session command is called to set up sessions with MRCP server(s). Required parameters for this command are defined in the "clientinterfacedefs" include file. The parameters are passed by referencing the open session command.

When the session has been opened, the configuration parameters are updated with session handle values. The session handles are used on all calls to change and close session calls and in all MRCP command calls.

All commands return either success (value of zero) or error (value of -1). An error status is returned for calls to MRCP-related commands if a session has not been created to an MRCP server. For commands other than stop and start, a return value of success means that the requested API has been queued for processing and not that the command has completed.

When an application finishes with an MRCP session, a call to the close session command must be made to

terminate the session with the MRCP server. Then the application must call the stop command. Stop performs a uniform shutdown of all objects and sessions created by the library, reducing both memory and resource leaks.

Session Manager Object

The session manager is responsible for opening and closing session objects and for adding resources to existing session objects. The client library requires that each session be limited to one ASR and one TTS server session. However, multiple sessions are supported. When opening a session, the session manager processes the configuration parameters to determine which type of MRCP resource server is required (ASR, TTS, or both) and whether separate server sessions are needed for ASR and TTS. After parsing the configuration parameters, the open session command instantiates the session objects needed to support the client application's MRCP session.

To add resources to a session or close a session, the session handle parameter is used to assure that the correct session is modified or closed. Adding resources to a session can be done if a single session to an MRCP server is using only ASR or only TTS functionality and the session receiving the resources is not currently active. Adding resources to a session will impact performance when the protocol signaling occurs that is required to add the resource.

Close session uses the session handle to retrieve a pointer to the MRCP session, and then uses that pointer to stop the session.

Session Object

The session object is responsible for starting, stopping, adding to sessions, processing MRCP client commands, and starting the callback process. When a session starts, the session pointer is registered with the handle manager and the handle field in the configuration parameters, passed by reference, is updated. ASR and/or TTS handles are defined when a session starts based on the resources requested for a session. The ASR and/or TTS session handles are required values for all MRCP commands. A configuration parameter contains a callback that will be used when a session has been established.

Adding resources to a session results in the issuing of a SIP re-invite command to the MRCP server. The re-invite is necessary to signal the addition of a resource to the server.

The stop session command un-registers the session object with the handle manager, instantiates and queues a disconnect command to the task processor, and waits to receive observer notification that the SIP session has been closed to the server. Logic has been added to the wait loop to handle conditions when an acknowledgement of a request to end a session is not received. The current code base forces the session to shut down after a delay of five seconds. Additional code in the routine, but commented out, can be used to resubmit the shutdown request in the event that no response is received.

Session handles are used to send MRCP API command requests received in the MrcpSystem object to the session object. A command object is instantiated for each API call, and a command ID is requested from the command ID manager. The command object parameters are set, and the object is queued to the task processor for processing. A command ID is requested from the command ID manager and is set in the parameter values passed by reference. This allows the client application to associate the callback status received with the MRCP command issued. The command ID and associated callback are registered with the command ID manager.

Signaling Objects

The signaling objects provide the SIP (MRCP V2) or RTSP (MRCP V1) signaling necessary to establish, modify, and tear down sessions with MRCP servers. While MRCP V1 support is not fully implemented in the client library version discussed in this application note, outlines of many of the objects required to support it are included. Because the signaling object must work closely with the TCP and UDP socket layers, the signaling object is responsible for processing network traffic received for both signaling and MRCP command responses. A call to the signaling object's start command establishes a socket connection to the MRCP server that is used to send all signaling commands. Conversely, a call to the stop command signals the MRCP server (SIP BYE) that the session is ending and closes the socket.

The session object instantiates signaling objects and queues them to the task processor. If the signaling object is in the proper state to support the signaling command, the signaling command's primitive function (prim) is called to generate the signaling command. The signaling object parses the responses to commands, and the state of the object is updated based on responses received.

RTP Manager Object

The RTP manager object is responsible for setting up sessions, stopping sessions, and assigning the unique TCP sockets that are needed to send and receive audio packets from the MRCP server. Like all manager objects in the client library, the RTP manager object has a singleton design, which safeguards the TCP sockets assigned per session. During the session setup process, the RTP manager object instantiates both an audio stream object and an RTP object. The mapping of audio stream object to RTP object is used to coordinate the stopping and shutdown of audio stream and RTP objects.

RTP Object

The RTP object is responsible for establishing the inbound and outbound audio sockets and managing the RTP headers for all audio sent or received. For inbound RTP packets, the RTP object receives the audio packet, removes its RTP header, and places the packet in the audio-stream-objects processing queue. An RTP header is applied to outbound audio buffers received in the outbound processing command, and they are then sent via the socket to the MRCP server.

The RTP object opens the RTCP sockets, but does not send or process the packets received.

The shutdown command closes all sockets and causes the inbound and outbound processing threads to exit.

Audio Stream Object

The audio stream object provides support for inbound and outbound audio. Inbound audio is added to the inbound audio queue by the RTP object. The inbound audio processor command retrieves the audio from the inbound audio queue and calls the session's audio callback to send the audio to the callback routine. The outbound audio processor command is called via the MrcpSystem's add outbound audio command, which uses the audio handle to retrieve a pointer to the audio stream object. The audio stream's add outbound audio command is then called to pass the audio buffer (collected from the caller) for processing.

Command Objects

Command objects are those objects in the client library that are designed to work in conjunction with task processor objects. A command object's responsibility is to delegate the execution of a command to an object without knowing anything about that object. All command objects have an execute command that calls the first of a series of objects that comprise the state machine of the object for which the command was issued.

Command objects are used in signaling and MRCP commands and in results and callback processing objects. Command objects typically do the following:

1. Instantiate an object
2. Set the object's parameters
3. Queue the object to the task processor object

The task processor's queue processing command calls the queued task's execute command and then deletes the command object.

Command ID Manager Object

The command ID manager object registers and un-registers objects using the unique MRCP request IDs it generates. As singleton objects, the reference IDs are guaranteed to be unique. Callbacks associated with each object are mapped in the command ID manager object, and the callback-to-object map is used to process callbacks for registered objects. The callback called is the callback identified by the client application when issuing a command.

Callback processing involves retrieving the callback based on a command ID, setting the result's structure information, and queuing the callback task command for processing.

Handle Manager Object

The handle manager object's responsibility is to assign unique integers to be used in place of object pointers for general sessions and audio sessions. The object maintains a map of assigned handles to pointers to register, un-register, and retrieve the object pointers using the handles. The use of handles instead of object pointers prevents the client application from using call object commands, which it should not be using via the object pointer.

Task Processor Objects

Task processor objects provide the queuing function needed to marshal commands and events received. This guarantees the processing of commands and the return of events in the order they are received. The task processors support the command pattern, providing object

encapsulation, which frees objects from having to know about the other objects it needs to process commands.

There are two task processor implementations and queues within the client library. One queues and processes command requests and events received. The second queues and executes the callbacks registered for commands and sessions. Each instance of the task processor is multi-threaded with the main thread providing the queue command and the worker thread providing the processing for objects in the queue. The separation of the task and event processor from the callback processor ensures minimum impact to the client library in the case of a client application callback block.

The task processor's threaded command pulls the head item from the queue, calls the object's execute command, and then deletes the object. Exception handling is provided within this processing thread to safeguard the thread from conditions that might cause it to exit unexpectedly. When the stop command is called during shutdown, all queued objects are discarded when the processing thread exits.

Using the Client Library

The interface to the client library is contained in the MrcpSystem class, and client applications must include the MrcpSystem header file. In addition, the client library defines a namespace of "MrcpV2RefLib," which must be used in the client application.

Boost is an open-source, platform-independent, proven software library for C++, and Boost libraries are used throughout the client library. As a consequence, the development environment for client applications must contain the Boost libraries. For a web address that accesses information about obtaining the Boost libraries, see the *Reference* section at the end of this application note.

Because the MrcpSystem class is a singleton, the class's instance command is called to retrieve a pointer instead of its constructor. Instance will instantiate a MrcpSystem object if one does not exist or will return a pointer to an existing object if one exists.

The next step is to set the configuration information that the client library needs to communicate with MRCP servers. ClientInterfaceDefs contains two structures that set the configuration information. The resource configuration structure is the primary structure and contains ASR and TTS server configuration structures and a callback parameter that defines the audio callback for the client application. Audio callback is the routine that is called to deliver inbound audio. The macro `INIT_RESOURCE_CFG_STRUCT` should be called to initialize the structure properly.

Next, the Mrcp Server Config structure should be configured for each of the MRCP servers to which the client application will be connecting. A single MRCP session can connect to one ASR and one TTS server, but there is no limit to the number of sessions or servers that the client library can support. The macro `INIT_MRCP_SERVER_CONFIG` should be called to initialize the structure. The ServerAddress value contains either the IP address or DNS name of the MRCP server, and the server port field contains the MRCP signaling port configured on the MRCP server. The callback value should be set to the callback routine, which will receive a callback when the session or sessions have been established with the MRCP server.

Next, the client library is started with a call to the MrcpSystem start command. This call starts the various manager objects in the client library and completes synchronously. The client application then calls the MrcpSystem open session command, passing in the resource configuration structure by reference. When the session has been established, the callback specified in the configuration parameters is called, and the configuration parameter structure is updated with the ASR and/or TTS handles that the client library has assigned for the session. The client application must use these handles for all MRCP-related commands.

MRCP commands supported by the client library are located in the MrcpSystem class. Each of the commands executes synchronously and requires a session handle, a callback function, and a reference number. The reference number is passed by reference and will contain a number that associates the callback received to the command issued. It is strongly recommended that a callback be specified for each command. Enough information is provided in the callback structure to allow a single callback routine to handle the callback functionality for processing multiple commands.

Some of the MRCP commands contain a separate structure that contains parameters and callback function and reference number values. Each of the structures is defined within the ClientInterfaceDefs header file.

The client library calls the registered callback when any of the commands completes. A structure is returned to the callback routine in the client application that contains the reference number, the status value for the call received from the MRCP server, the ASR and TTS handles, and the entire MRCP packet received from the MRCP server.

Audio callbacks receive the same structure as the MRCP command callbacks with the exception of the return packet parameter, which contains audio instead of MRCP packet information.

When a client application is finished with the library, it calls the MrcpSystem close session command for each of its sessions and passes in the session handle. Finally, a call to the MrcpSystem stop command gracefully closes the library.

Summary

The MRCP V2 client library provides support for commonly used MRCP ASR and TTS commands. Additional ASR and TTS commands can be added easily, and parameters expanded or enhanced with additional MRCP functionality for verification or recording. The client library can provide a jumpstart when creating an MRCP client, but may need to be supplemented to create an entire client for an application.

Acronyms

API Application Programming Interface

ASR Automatic Speech Recognition

IETF Internet Engineering Task Force

MRCP V2 Media Resource Control Protocol Version 2

RTCP Real Time Control Protocol

RTSP Real Time Signaling Protocol

SIP Session initiation Protocol

TTS Text To Speech

XML eXtensible Markup Language

References

Boost Library (Source for Open Source Software Required for Client Library Source Code Provided with this Application Note) http://boost.org/more/getting_started.html

Gamma, Erich, Ralph Johnson, Richard Helm, and John Vlissides. *Design Patterns: Elements of Reusable Object-Orientated Software*. Addison-Wesley, 1994.

A Media Resource Control Protocol (MRCP) Developed by Cisco, Nuance, and Speechworks (IETF RFC 4463) http://www.rfc-archive.org/getrfc.php?rfc=4463

Dialogic®

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 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.

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

Dialogic is a registered trademark of Dialogic Corporation ("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.

The 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.

Copyright © 2007 Dialogic Corporation All rights reserved.

04/07 10285-001