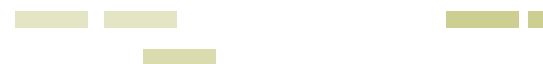


MRCP V2 Client Library Programmer's Guide



Executive Summary

This application note provides information about developing applications using the classes and commands that make up the MRCP V2 client library. Dialogic developed the MRCP V2 client 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.....	2
MRCP Client Library Usage Overview.....	2
Callback Processing	2
Configuration and Startup.....	3
Configuring the Client Library	3
Start()	4
Stop().....	4
OpenSession().....	4
AddSessionResources().....	4
CloseSession().....	5
Generic MRCP Commands.....	5
GetParams().....	5
SetParams()	5
ASR MRCP Commands.....	5
DefineGrammar()	5
Recognize().....	6
Interpret()	6
StartInputTimers()	6
GetResult().....	6
StopRecog()	7
TTS MRCP Commands.....	7
Speak()	7
StopSpeak()	7
Pause()	7
Resume().....	8
BargeIn()	8
Audio Commands.....	8
StartAudio().....	8
StopAudio().....	8
AddOutboundAudio()	9
References	9
Acronyms.....	9
For More Information.....	9

Introduction

This document serves as a programmer's reference for the Media Resource Control Protocol Version 2 (MRCP V2) Client Library (referred to in this application note as the "client library"). It is intended to help developers understand the classes, commands, and operation of the client library. Commands within this library provide support for automated speech recognition (ASR) and text-to-speech (TTS) commands commonly used in telephony applications.

The client library was architected and built around the design patterns documented in the book *Design Patterns: Elements of Reusable Object-Oriented Software* [Gamma, Helm, Johnson and Vlissides]. Design patterns provide a high level of encapsulation, which results in an easily extensible library. Developers may choose to customize and extend the 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/openmrpcclient>.

MRCP Client Library Usage Overview

The client library application programming interface is contained within a single class, the CMrcpSystem class. All configuration and MRCP client commands, and audio interfaces are provided within this class. This class is instantiated as a singleton object, one instance per application, in order to ensure the uniqueness of values required in MRCP client requests. Command calls in the CMrcpSystem class instantiate additional objects and threads needed to support multiple sessions to one or more MRCP servers within a single client application.

Parameter structures used in the client function calls are defined in the ClientInterfacesDef header file.

Commands not related to the configuration of the client library execute asynchronously. Client library API commands synchronously return a success or failure when called. Success means that the library was properly started and ready to process the command. Callbacks are utilized by the MRCP V2 Client Library to call back to the client application when information related to a command has been received from the MRCP server.

The boost libraries (www.boost.org) are used throughout the library. As a result, the development environment for client applications must have the boost libraries installed. Boost is an open source, platform independent, proven software library for C++.

While not part of the CMrcpSystem class, the client library contains a logger object. The logger object currently logs runtime information to both the console window and a log file. Information logged includes signaling messages sent to set up a session, success and failure information, and error conditions encountered.

The client library defines a namespace of "MrcpV2RefLib" that must be used in the client application.

Callback Processing

All asynchronous commands in the client library use a reference number and a callback function parameter. When an asynchronous command completes, the callback function is called passing partially parsed results and the complete MRCP packet received. The following structure is sent to the callback routine identified as a command parameter:

```
typedef struct{
    int referenceNumber;
    int MrcpReturnValue;
    MrcpSessionHandle asrSessionHandle;
    MrcpSessionHandle ttsSessionHandle;
    char MrcpReturnPkt [ 10000] ;
} MRCP_RESULT_STRUCT;
```

A description of each element of the structure follows:

- **ReferenceNumber** — The number assigned by the library and should be used to track command completion. Reference numbers are passed by reference from the client application and assigned by the client library.
- **MrcpReturnValue** — The MRCP numerical status value received for the command Execution.
- **asrSessionHandle** — The handle assigned to the ASR session issuing the command. Zero if no ASR session active.
- **ttsSessionHandle** — The handle assigned to the TTS session issuing the command. Zero if no TTS session active.
- **MrcpReturnPkt** — The complete MRCP status packet received.

Configuration and Startup

Configuring the Client Library

Each client library session can be configured to specify one TTS and one ASR server. These resources can be located on the same or separate MRCP servers. The following structure, contained in the ClientInterfaceDefs header file, is used to configure the server information:

```
typedef struct{
    int    mrcpVersion;
    char*  serverAddress;
    int    serverPort;
    MrcpCallBackFunction  callBack;
} MRCP_SERVER_CONFIG;
```

A description of each element of the structure follows:

- **mrcpVersion** — The MRCP version, 1 or 2, used to communicate with the MRCP server. Version 1 support was not available when this document was written.
- **serverAddress** — The IP address or DNS name of the MRCP server used
- **serverPort** — The network port defined on the MRCP server that receives session setup requests.
- **callBack** — A routine in the client application that is called when the MRCP session has been set up.

One or more of these structures is needed, depending on the MRCP server resources needed and whether or not the resources are on the same server. The MRCP_SERVER_CONFIG structure(s) are then used in the following configuration structure:

```
typedef struct {
    char*  clientAddress;
    MRCP_SERVER_CONFIG *asrServerConfig;
    MrcpSessionHandle  asrSessionHandle;
    MrcpAudioHandle    outboundAudioHandle;
    MRCP_SERVER_CONFIG *ttsServerConfig;
    MrcpSessionHandle  ttsSessionHandle;
    MrcpAudioHandle    inboundAudioHandle;
    MrcpCallBackFunction  audioCallback;
} RESOURCE_CFG_STRUCT;
```

A description of each element of the structure follows:

- **clientAddress** — The IP address of the computer hosting the client application.
- **asrServerConfig** and **ttsServerConfig** — These fields contain pointers to the MRCP_SERVER_CONFIG structures defined above. If the same server is used for ASR and TTS, the same structure pointer can be reused.
- **asrSessionHandle** and **ttsSessionHandle** — These fields are passed by reference. The client library assigns and populates the handle value.
- **outboundAudioHandle** and **inboundAudioHandle** — These fields are passed by reference and assigned an inboundAudioHandle value by the client library.
- **audioCallback** — These fields identify the callback responsible for receiving and processing all inbound TTS audio.

Users of the library are encouraged to use the INIT_MRCP_SERVER_CONFIG and the INIT_RESOURCE_CFG_STRUCT macros to ensure that the structures are properly initialized before use.

Start()

Inputs: None

Returns: MrcpSuccess(0) if successful
MrcpError (-1) if failure

Mode: Synchronous

The **Start()** function initializes the client library for use. Start initializes the state machine, instantiates the singleton objects, and creates the callback and task processors. This function must be called before any other client library function and should only be called one time per client application.

A value of -1 is returned if this function has been previously called in the application. Otherwise, an MrcpSuccess (0) is returned.

Stop()

Inputs: None

Returns: MrcpSuccess(0) successful

Mode: Synchronous

The **Stop()** function stops all of the singleton objects, sets the state machine to stop, and completes an orderly shutdown of the client library. The function is

called when the client application has finished using the client library.

The recommended steps for stopping the library involve the client application closing all open MRCP sessions before calling the **Stop()** function. Sessions are closed using the **CloseSession()** function.

OpenSession()

Inputs: RESOURCE_CFG_STRUCT by reference

Returns: MrcpSuccess (0) if successful
MrcpError (-1) if error

Mode: Asynchronous

The **OpenSession()** function establishes MRCP sessions with the MRCP server. A value of MrcpSuccess indicates that the library was in the proper state to schedule the asynchronous opening of a session. The callback function defined in the MRCP_SERVER_CONFIG structure is called when the session has been opened, or failed to open, with the MRCP server.

The RESOURCE_CFG_STRUCT is updated with values for the asrSessionHandle and/or the ttsSessionHandle parameters. Values for each of these handles are dependant on whether ASR or TTS resources were requested for the MRCP session. Use the macros INIT_MRCP_SERVER_CONFIG and INIT_RESOURCE_CFG_STRUCT to ensure that the configuration parameter structures are properly initialized. All MRCP functions called for the session to be opened must include the handle value returned.

AddSessionResources()

Inputs: MrcpSessionHandle
MrcpResourceType

Returns: MrcpSuccess(0) if successful
MrcpError(-1) if failure

Mode: Asynchronous

The **AddSessionResources()** function adds either an ASR or TTS resource to an existing MRCP session. This function should only be used to add an ASR or TTS resource to an MRCP session if the resource was not requested when the session was opened. This function triggers the generation of a SIP re-invite message to be sent to the MRCP server. The client application must wait for the callback indicating the resource was added before invoking functions for the session.

This function returns `MrcpError` if the library has not been started or if the `MrcpSessionHandle` specified was not found in the open session's list.

CloseSession()

Inputs: `MrcpSessionHandle`

Returns: `MrcpSuccess(0)` if successful
`MrcpError(-1)` if error

Mode: Synchronous

The `CloseSession()` function performs shutdown processing for an MRCP session. A SIP BYE message is sent to the MRCP server, and all resources and entries for the closed session are removed from the client library. Close all sessions with this function before stopping the library and/or exiting the client application to ensure that licensed MRCP resources are freed on the MRCP server.

This function returns `MrcpError` if the session handle is not found in the client library's registered handles list.

Generic MRCP Commands

GetParams()

Inputs: `MrcpSessionHandle`
`MrcpCallBackFunction`
`GET_PARAMETERS`
Reference Number – by reference

Returns: `MrcpSuccess(0)` if successful
`MrcpError(-1)` if error

Mode: Asynchronous

The `GetParams()` function retrieves the MRCP session parameters set for the MRCP session identified by the session handle. Values for parameters listed in the `GET_PARAMETERS` structure are returned. All parameters for the MRCP server are returned if the `GET_PARAMETERS` structure is empty. Returning all MRCP server parameters generates multiple responses from the MRCP server and impacts performance. The number of parameters that can be requested is unlimited.

A value of `MrcpSuccess` is returned if the function can be scheduled for processing. `MrcpError` is returned if the `MrcpSessionHandle` used cannot be found. One or more calls to the `MrcpCallBackFunction` are made to return the requested parameter values to the client application. The `MrcpCallBackFunction` is called when the MRCP message indicating the success or failure of the request is received.

SetParams()

Inputs: `MrcpSessionHandle`
`MrcpCallBackFunction`
`SET_PARAMETERS`
Reference Number – by reference

Returns: `MrcpSuccess(0)` if successful
`MrcpError(-1)` if error

Mode: Asynchronous

The `SetParams()` function sets the MRCP server parameters for a session. Parameters set by this function are used for all MRCP functions unless an overriding parameter is supplied in the MRCP function call. This command may be used to set parameters that are not currently supported in the MRCP functions contained within the client library. The number of parameters that can be set using this function is unlimited.

A value of `MrcpSuccess` is returned if the function can be scheduled for processing. `MrcpError` is returned if the `MrcpSessionHandle` used cannot be found. The `MrcpCallBackFunction` is called when the MRCP message indicating the success or failure of the request is received.

ASR MRCP Commands

DefineGrammar()

Inputs: `MrcpSessionHandle`
`DEFINE_GRAMMAR_STRUCT`

Returns: `MrcpSuccess(0)` if successful
`MrcpError(-1)` if error

Mode: Asynchronous

The `DefineGrammar()` function defines the grammar to be used for a recognition or interpret MRCP command. Grammar is specified in the `gramContent` field of the `DEFINE_GRAMMAR_STRUCT` structure. This field can contain the actual grammar or a URI to predefined and/or precompiled grammars. Grammars can also be specified as part of the MRCP `Recognize()` or `Interpret()` functions.

The `MrcpCallBackFunction` is called when the MRCP message indicating the success or failure of the request is received. A value of `MrcpError` is returned if the session handle is not found in the client library's list of active sessions. `MrcpSuccess` is returned if the command was successfully queued for processing.

Recognize()

- Inputs:** MrcpSessionHandle
 RECOGNIZE_STRUCT – passed by reference
- Returns:** MrcpSuccess(0) if successful
 MrcpError(-1) if error
- Mode:** Asynchronous

The **Recognize()** function signals the MRCP server to begin recognition of audio. Audio is sent to the MRCP server on the network ports defined by the MRCP server when the session was established (refer to the **AddOutboundAudio()** function for additional information). The gramContent field (all the fields referenced in the **Recognize()** function are in located in RECOGNIZE_STRUCT) can be used to define the grammar to be used in the recognition process. Passing the grammar in this field will delay recognition as the grammar is compiled or downloaded for use. The contentType field is used to identify the format of the grammar to be used in the recognition process. The cancelQueue field tells the MRCP server to cancel the recognition if the request must be queued and not processed immediately.

The MrcpCallBackFunction is called when an in-progress response is received from the MRCP server, and again when the recognition-complete message is received. A value of MrcpError is returned if the session handle is not found in the client library's list of active sessions. MrcpSuccess is returned if the command was successfully queued for processing.

Interpret()

- Inputs:** MrcpSessionHandle
 MrcpCallBackFunction
 SET_PARAMETERS
 INTERPRET_STRUCT
 Reference Number – by reference
- Returns:** MrcpSuccess(0) if successful
 MrcpError(-1) if error
- Mode:** Asynchronous

The **Interpret()** function signals the MRCP server to begin recognition of audio. Audio is sent to the MRCP server on the network ports defined by the MRCP server when the session was established. While this function is

similar to the **Recognize()** function, the **Interpret()** function is different in that the results relevant to acoustic matching are not returned. The IETF MRCP specification has additional information on the differences between Interpret and Recognize (for a link to this document, see For More Information). The content field, located in the INTERPRET_STRUCT, can be used to define the grammar to be used in the interpret process.

The MrcpCallBackFunction is called when an MRCP In-progress is received and again when the Interpret-complete message is received. A value of MrcpError is returned if the session handle is not found in the client library's list of active sessions. MrcpSuccess is returned if the command was successfully queued for processing.

StartInputTimers()

- Inputs:** MrcpSessionHandle
 Reference Number – by reference
- Returns:** MrcpSuccess(0) if successful
 MrcpError(-1) if error
- Mode:** Asynchronous

The **StartInputTimers()** function starts the MRCP session recognition timer. This timer is used to determine if no audio input has been received for recognition within the pre-defined duration. StartInputTimers() functions are used for MRCP deployments with separate MRCP ASR and TTS servers to signal the start of the recognition process. Developers using the client library may experience time-out errors from the MRCP server. This time-out condition, if it occurs, is most likely caused by the client application taking too long processing the audio collected.

The MrcpCallBackFunction is called when the MRCP server responds with a success or failure for the command. A value of MrcpError is returned if the session handle is not found in the client library's list of active sessions. MrcpSuccess is returned if the command was successfully queued for processing.

GetResult()

- Inputs:** MrcpSessionHandle
 MrcpCallBackFunction
 Reference Number – by reference
- Returns:** MrcpSuccess(0) if successful
 MrcpError(-1) if error
- Mode:** Asynchronous

The **GetResult()** function requests the results of a recognition request when the MRCP server is in the recognized state. **Note:** This command might not be supported on all MRCP server implementations.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

StopRecog()

Inputs: **MrcpSessionHandle**
MrcpCallBackFunction
 Reference Number – by reference

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **StopRecog()** function stops a recognition in process. This function is equivalent to the MRCP Stop command. This function is called when the client application elects to stop a recognition in process.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

TTS MRCP Commands

Speak()

Inputs: **MrcpSessionHandle**
SPEAK_STRUCT

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **Speak()** function signals the MRCP server's TTS resources to deliver audio. This function supports both speech synthesis and streaming of pre-recorded audio files. For speech synthesis, the content field (all the fields referenced in the **Speak()** function are in located in **SPEAK_STRUCT**) contains the text to be synthesized and streamed to the client application. The same content

field contains a URI pointer to a pre-recorded file to be streamed to the client application. The content-type field is used to identify the format of the content. The **killOnBargeIn** field is used to indicate whether or not barge in is permitted. **VoiceName** allows the client application to specify a specific voice resource for synthesis. The **speechLanguage** field identifies the language to be spoken for synthesis. **VoiceName** and **speechLanguage** are dependant on the MRCP supporting the values requested. The IETF MRCP specification has additional information on valid values (for a link to this document, see For More Information).

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

StopSpeak()

Inputs: **MrcpSessionHandle**
MrcpCallBackFunction
 referenceNum – passed by reference

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **StopSpeak()** function stops an audio stream. This function is equivalent to the MRCP **Stop()** function. The **StopSpeak()** function is likely to be used more often for those environments having separate ASR and TTS servers.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

Pause()

Inputs: **MrcpSessionHandle**
MrcpCallBackFunction
 referenceNum – passed by reference

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **Pause()** function tells the TTS resource to pause speech output. Unlike the StopSpeak command, the audio stream can be resumed at the point it was paused using the resume function.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

Resume()

Inputs: **MrcpSessionHandle**
MrcpCallBackFunction
referenceNum – passed by reference

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **Resume()** function resumes an audio stream previously interrupted by the pause function. A status of success is returned if the client application calls this function while an audio stream is active regardless of whether or not a previous call to the pause function was made.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

BargeIn()

Inputs: **MrcpSessionHandle**
MrcpCallBackFunction
referenceNumber – passed by reference

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Asynchronous

The **BargeIn()** function signals the TTS resource that a barge in event has occurred. Like the **Stop()** function, a **BargeIn** call stops all audio currently being streamed to the client application. This function would more likely be used in environments where the ASR and TTS MRCP

resources are on separate servers where a client application would have to alert the TTS server to a barge in condition.

The **MrcpCallBackFunction** is called when the MRCP server responds with a success or failure for the command. A value of **MrcpError** is returned if the session handle is not found in the client library's list of active sessions. **MrcpSuccess** is returned if the command was successfully queued for processing.

Audio Commands

StartAudio()

Inputs: **MrcpAudioHandle**

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Synchronous

The **StartAudio()** function sets up the audio stream objects and sets their state for an MRCP session. This function must be called prior to working with any of the MRCP audio functions. The **MrcpAudioHandle** is the handle returned when the **OpenSession()** function is called and must be used for all audio related functions

This function returns a value of **MrcpError** if the audio handle is not found. Otherwise, a value of **MrcpSuccess** is returned.

StopAudio()

Inputs: **MrcpAudioHandle**

Returns: **MrcpSuccess(0)** if successful
MrcpError(-1) if error

Mode: Synchronous

The **StopAudio()** function stops the audio stream objects and sets their state for an MRCP session. This function must be called when the client application is done with audio for a session, usually before a **CloseSession()** function call is made. The **MrcpAudioHandle** is the handle returned when the **OpenSession()** function is called and must be used for all audio related functions.

This function returns a value of **MrcpError** if the audio handle is not found. Otherwise, a value of **MrcpSuccess** is returned.

AddOutboundAudio()

Inputs: MrcpAudioHandle
audioBuffer – passed by reference

Returns: MrcpSuccess(0) if successful
MrcpError(-1) if error

Mode: Asynchronous

The **AddOutboundAudio()** function sends an audio buffer to the MRCP server. Audio buffers are sent to the MRCP server for MRCP ASR Recognize and Interpret command processing. This function has been tested and verified with a buffer size of 160 bytes. Audio buffers received are pushed onto the outbound audio queue for processing.

This function returns a value of MrcpError if the audio handle is not found. Otherwise, a value of MrcpSuccess is returned.

References

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

Acronyms

API	Application Programming Interface
ASR	Automatic Speech Recognition
Client Application	Application built using the MRCP V2 Client Library
IETF	Internet Engineering Task Force
IP	Internet Protocol
MRCP V2	Media Resource Control Protocol Version 2
SIP	Session Initiation Protocol
TTS	Text-to-Speech
URI	Uniform Resource Identifier — Generic set of names/addresses that are short strings that refer to web resources

For More Information

Boost library information — http://boost.org/more/getting_started.html

Media Resource Control Protocol (MRCP) specification — <http://www.rfc-archive.org/getrfc.php?rfc=4463>

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.

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.

This document discusses various open source products, systems and releases. Dialogic is neither responsible for your decision to use open source in connection with Dialogic products including without limitation those referred to herein, nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, business, or intellectual property rights.