Vocera Messaging Interface Guide

Version 5.2.3

Notice

Vocera Communications, Inc.
www.vocera.com
tel :: +1 408 882 5100
fax :: +1 408 882 5101

Last modified: 2019-06-19 11:27

Docs_VS_523Rel build 177

Contents

Getting Started

The Vocera Messaging Interface (VMI) is an application programming interface (API) that enables text messaging between external systems and Vocera badges via the Vocera Voice Server. VMI allows a *client* (for example, a nurse call system) to send a text message to a badge, and to receive acknowledgements that describe the delivery status of the message, along with optional responses from a message *recipient*.

About the VMI Documentation

This documentation provides the information you need to develop applications using the Vocera Messaging Interface (VMI). It describes the C++ classes and methods that are implemented in the VMI libraries, provides information on configuring VMI, and discusses the sample code included in the Vocera Developer Kit.

VMI Features

The VMI provides the following features:

- C++ interface for developing client applications.

  Client applications communicate with the Vocera Voice Server via a dynamic link library (DLL) (32-bit or 64-bit) and header files provided by Vocera. The header files define a C++ API to the VMI.

  The API includes methods for sending a message to a badge, deleting a message from a badge, and getting information about a user or group.

  See **VMI API Reference** for detailed information about all VMI methods.
- Text messages from clients are sent directly to a badge.

  The badge plays an alert and displays the message. Urgent messages are played immediately.
- Recipients can play messages aloud.

  Using button clicks or voice commands, badge users can play text messages aloud via text-to-speech or (optionally) as audio files.

  See **Working with VMI Messages** for more information.
- Client applications can specify responses.

  A VMI message can supply any of the following: a list of responses, a callback number, audio files containing recordings of the message and responses.

  See **Using the Vocera Messaging Interface**.
- Automatic logging of interactions resulting from a message.

  VMI notifies the client when a message is delivered, when it is acknowledged, and when the recipient responds. See **Understanding the Flow of Events**.

  A client application can use this information, for example, to escalate unanswered messages to the badge of another user.
- Multiple connections supported.

VMI allows multiple client connections to the Vocera Voice Server, although each client can maintain only one connection at a time.

System Requirements

To run a VMI client application, a computer must be able to run one of the following operating systems:

- Windows 7
- Windows Server 2008 R2
- Windows Server 2012

The computer must have enough free hard disk space to store **vmi.dll**, the client application, and any application data. The computer must also have enough memory to run the client application.

In addition, a VMI-enabled license key must be installed on the Vocera Voice Server. A VMI-enabled license includes the letter N followed by a number that represents the number of allowed client connections. For example, N6 in the license key indicates that 6 VMI client connections are allowed.

When the Vocera Voice Server starts with a VMI-enabled license, it displays information similar to the following in the server logs:

```
01/23/16 00:00:05.698 [I] [281] There are currently 2 connected VMI clients
of 4 licensed.
```

Developing VMI Client Applications

VMI client applications communicate with the Vocera Voice Server via a dynamic link library (DLL) and header files provided by Vocera. The header files define a C++ API.

The VMI Directory Structure

The `VMI` directory on the Vocera Developer Kit CD contains all the information and files you need to create VMI applications.

The `VMI` directory contains the following subdirectories and files:

Table 1: VMI Directories

Directory	Content
`\VMI\docs`	VMI documentation
`\VMI\vmi`	VMI header files
`\VMI\vmi\Win32`	32-bit version of VMI DLL and library
`\VMI\vmi\x64`	64-bit version of VMI DLL and library
`\VMI\vmitest`	`vmitest` sample application source files
`\VMI\vmitest\Win32`	32-bit version of `vmitest` DLL and executable
`\VMI\vmitest\x64`	64-bit version of `vmitest` DLL and executable

How to Develop VMI Client Applications

To develop a VMI client application:

1. Copy the following files from the **VMI** directory of the **Vocera Developer Kit** CD into your development directory.

Table 2: VMI software files

Folder	File	Description
`\VMI\vmi`	`dll.h`, `listener.h`, `log.h`, `vmi.h`	Header files to include in your C / C ++ source code.
`\VMI\vmi\Win32\Release` or `\VMI\vmi\x64\Release`	`vmi.lib`	Library file for linking your code to the VMI implementation.
`\VMI\vmi\Win32\Release` or `\VMI\vmi\x64\Release`	`vmi.dll`	Dynamic link library containing the VMI implementation.

2. Write the code to implement your client application.

   When you use the VMI API, note the following requirements:

   - VMI client IDs must be unique. Each client can maintain only one connection at a time.
   - VMI message IDs must be unique for each client.

   You must have a compiler and linker that can handle libraries (.lib files) generated by Microsoft Visual C++ 2005.
3. Test your application.
4. Deploy your application.

Note: When you test and deploy your application, the `vmi.dll` file must be in the library path of the machine running the application. Also, a VMI-enabled license key must be installed on the Vocera Voice Server.

Using the Sample Application

Vocera provides a sample application in the form of a Microsoft Visual C++ 2005 project. The project, called **`vmitest`**, is completely self-contained, and includes source code (**`vmitest.cpp`** and **`vmitest.h`**) as well as **`vmi.dll`**, **`vmi.lib`**, and the header files you will need to complete your integration.

The compiled and linked application, **`vmitest.exe`**, is provided as well. You can use it as a reference, and as a test bench to make sure that the Vocera Voice Server is working properly. Before you run it, put **`vmi.dll`** on your DLL path. (You can put the DLL in the same directory as the executable program file.) For best results, have a Vocera Voice Server up and running with at least one user logged in to a badge.

Note: `vmitest` is sample software provided solely to illustrate the use of the API. Vocera provides it AS IS. You are solely responsible for verifying its suitability for any specific purpose or application.

If you copy the `VMI` directory of the **Vocera Developer Kit** CD onto drive C of your development computer, you can enter the following command to run the sample application:

```
c:\VMI\vmitest\Win32\Release\vmitest.exe
```

or

```
c:\VMI\vmitest\x64\Release\vmitest.exe
```

The application is a command-driven 32-bit or 64-bit console application. When the application starts, it queries the registry for SSL and Port options and loads those values if they exist. The application then displays a list of commands. To issue a command, type its first letter (for example, type **O** for Open).

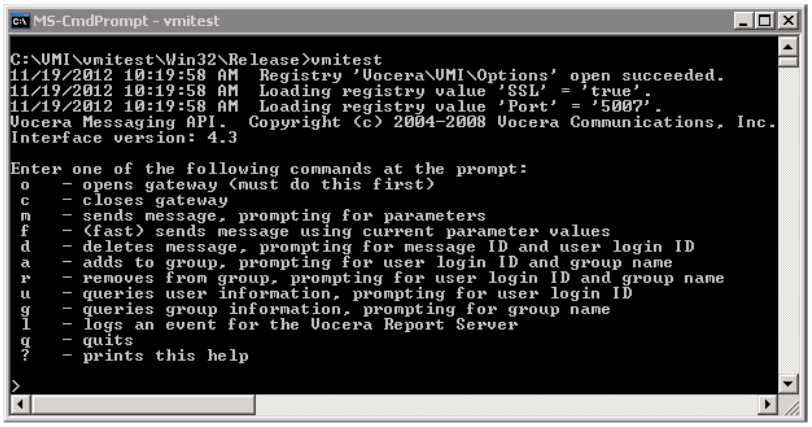

Figure 1: Vmitest application

The application prompts you for the arguments for each command. For example, when you type O, the application prompts for a string to identify the client. The value shown in brackets is the default value you get by pressing the Enter key without typing a value. To enter an empty value for a given argument, type a space and then press Enter. Values that you enter become the new defaults. The initial set of defaults can be configured, if desired, via command-line options (see the method `ParseCommandLineOptions` in `vmitest.cpp` for details).

You must issue the Open command before you can use the Message command to send a message. When prompted for the text of the message, you can use the notation `[ CR ]` to indicate a line break. After a message is sent, the application displays acknowledgment and response callback messages as these events occur (if you are testing against a live Vocera Voice Server).

Configuring VMI

This chapter describes several ways to configure VMI for your Vocera system.

Enabling TLS for VMI

By default, VMI clients make a TCP connection to the Vocera Voice Server on port 5005. Communication over this connection is not encrypted. For secure VMI communication, you can enable encryption using Transport Layer Security (TLS). This requires configuration on both the Vocera Voice Server and on the client-side VMI DLL (`vmi.dll`).

Enabling TLS for VMI on the Vocera Voice Server

There are two properties you can set in the `properties.txt` file on the Vocera Voice Server to enable VMI encryption:

- IPVMISecureEnable – enables secure VMI support within the Vocera Voice Server. When this property is set to TRUE the Vocera Voice Server opens a port to listen for secure VMI client connections. The default is FALSE.
- IPVMISecureListeningPortNo – specifies the port the Vocera Voice Server uses to listen for secure VMI client connections. The default is port 5007.

Note: The Vocera Voice Server uses an embedded self-signed certificate for authentication. You cannot specify a different certificate, such as one from a Certificate Authority.

To configure Vocera Voice Server for secure VMI connections:

1. On each Vocera Voice Server node, open the `\vocera\server\properties.txt` file in a text editor.
2. Add the IPVMISecureEnable property and set it to TRUE.
3. Add the IPVMISecureListeningPortNo property and specify the port to use for secure VMI connections.
4. Save the `properties.txt` file.
5. Stop the Vocera Voice Server and start it again. The Vocera Voice Server loads `properties.txt` into memory.

Note: If you have a Vocera Voice Server cluster, stop and start the standby nodes first, and then switch to the active node and choose Cluster > Failover in the Vocera Control Panel.

Enabling TLS for VMI Clients

After you enable a TLS connection for VMI on the Vocera Voice Server, you can configure your VMI client application to use a secure connection. The `vmi.dll` client registry interface provides two entry points that let you register your application to use a secure TLS connection with the Vocera Voice Server.

The VMI client queries the registry key "HKEY_LOCAL_MACHINE\Software\Vocera\VMI\Options" for the following values:

Table 3: VMI client registry values

Value	Data type	Value data
SSL	String	`true` or `false`
Port	String	*<port_number>*

Use the `rundll32.exe` command-line utility to set the `SSL` and `Port` options for your VMI client. To set a registry value, run `rundll32.exe` from the location where `vmi.dll` is located.

Note: When you run the `rundll32.exe` command, make sure you run it as an administrator.

Here are some examples showing how to run the `rundll32.exe` command:

Set the SSL option:	`rundll32 vmi.dll,SetOpt SSL true`
Set the Port option:	`rundll32 vmi.dll,SetOpt Port 5007`
Clear the SSL and Port options:	`rundll32 vmi.dll,ClrOpt SSL` `rundll32 vmi.dll,ClrOpt Port`

After you run each of these rundll32 commands, press Enter.

Setting Text Message Enunciation Properties

When a Vocera device receives an urgent Vocera Messaging Interface (VMI) message, the device plays an alert tone and then immediately plays the message along with the responses (if any) sent with the message.

For all other text messages, a Vocera badge or VCS application plays an alert tone and displays the text of the message; a Vocera smartphone plays an alert tone and prompts you whether to open the message.

There are two properties you can set in the `properties.txt` file on the Vocera Voice Server to control what types of text messages are played immediately on a badge or a Vocera smartphone:

- MsgEnuciateModeSmartphone – controls whether text messages received on Vocera Smartphones (Wi-Fi phones manufactured by Motorola) are played immediately. This property does not affect Vocera badges.
- MsgEnunciateMode – controls whether text messages received on Vocera badges or smartphones running Vocera Connect are played immediately.

The following table summarizes how the setting of these properties affects enunciation:

Value	Enunciated messages
0	Urgent VMI messages, high-priority VMP alerts, and urgent text messages sent by email. This setting is the default.
1	All urgent text messages.
2	All VMI messages. Urgent messages continue to take priority. Their enunciation interrupts other messsages and does not allow interruption by lower-priority messages; instead, lower-priority messages are delivered but not enunciated.
3	All text messages. Urgent messages continue to take priority. Their enunciation interrupts other messsages and does not allow interruption by lower-priority messages; instead, lower-priority messages are delivered but not enunciated.

Value	Enunciated messages
4 - 9	None.

To set text message enunciation properties:

Learn how to set text messages enunciation properties.

1. On each Vocera Voice Server node, open the `\vocera\server\properties.txt` file in a text editor.
2. Add the MsgEnunciateMode and MsgEnunciateModeSmartphone properties (if they have not already been added).

   Set each property to a value of 0 through 9, as described in **Setting Text Message Enunciation Properties** on page 10.

   For the MsgEnunciateMode property, you can also choose to enter a comma-delimited list to control text message enunciation for each VMI application or site. See **Specifying MsgEnunciateMode Per VMI Client or Site** on page 11.
3. Save the `properties.txt` file.
4. Stop the Vocera Voice Server and start it again. The Vocera Voice Server loads `properties.txt` into memory.

   Note: If you have a Vocera Voice Server cluster, stop and start the standby nodes first, and then switch to the active node and choose Cluster > Failover in the Vocera Control Panel.

Specifying MsgEnunciateMode Per VMI Client or Site

The MsgEnunciateMode property allows you to enter a comma-delimited list of values to specify the enunciate mode for a VMI client, a site, or both.

This helps you control which text messages are enunciated for each VMI application or site.

Each item in the comma-delimited list consists of three subitems delimited by colons:

ClientID : SiteName : EnunciateMode

where

- ClientID = The unique Client ID for a VMI application (optional, can be left blank)
- SiteName = The current site of the recipient of the message (optional, can be left blank)
- EnunciateMode = A one-digit numeric value representing the enunciate mode, described in **Setting Text Message Enunciation Properties** on page 10

The server processes the MsgEnunciateMode property from left to right using the following rules:

- The MsgEnunciateMode property values must be on one line. Values that run onto another line are ignored.
- A blank ClientID or SiteName subvalue serves as a wildcard.

  In the next value, the ClientID is blank, which means the value applies to all VMI client IDs:

  ```
  San Jose:1
  ```

  In the next value, the SiteName is blank, which means the value applies to all sites:

  ```
  Connexall::1
  ```

- A more specific value always takes precedence. For example, the value `Emergin:San Jose:1` takes precedence over `San Jose:2`.
- If there is a tie between two values, the leftmost value takes precedence. For example, there is a tie in the following two values, so the first one is used:

  ```
  Emergin:Santa Cruz:0, Emergin:Santa
  Cruz:4
  ```

- If a value cannot be resolved (for example, the ClientID and SiteName are specified incorrectly or the EnunciateMode is missing), the default EnunciateMode value, 0, applies.
- If you omit the optional ClientID and SiteName subvalues, you can also omit the colons. For example, the following values are all valid:

  ```
  1, San Jose:3, Emergin::4
  ```

- Examples

  `MsgEnunciateMode = 0, San Jose:3, Emergin:San Jose:4`

  The following text messages are enunciated:

  - All urgent VMI messages (0).
  - All text messages received by users in San Jose ("San Jose:3"), except those sent by Emergin, which are NOT enunciated ("Emergin:San Jose:4").

  `MsgEnunciateMode = 0, San Jose:1, Santa Clara:1, Emergin:San Francisco:2, ConnexAll:Palo Alto:2, Cupertino:3, Santa Cruz:4`

  

  Note: For the purposes of this example, the MsgEnunciateMode property spans multiple lines. However, in the actual `properties.txt` file, the MsgEnunciateMode property must appear on one line.

  The following text messages are enunciated:

  - All urgent VMI messages (0).
  - Urgent text messages received by users in San Jose ("San Jose:1")
  - Urgent text messages received by users in Santa Clara ("Santa Clara:1")
  - VMI text messages with the VMI client ID "Emergin" received by users in San Francisco ("Emergin:San Francisco:2")
  - VMI text messages with the VMI client ID "ConnexAll" received by users in Palo Alto "ConnexAll:Palo Alto:2")
  - All text messages received by users in Cupertino ("Cupertino:3")

  All text messages received by users in Santa Cruz are NOT enunciated ("Santa Cruz:4").

Enabling or Disabling the "Skip" Response to VMI Messages

By default, when users play VMI messages aloud they must either Accept or Reject the message (or say another valid response); they cannot respond by saying "Skip," which skips the message. However, you can enable the Skip response by adding the following property to the `properties.txt` file on the Vocera Voice Server:

```
MsgDisableSkipMessageResponse = False
```

Note: Regardless of this property setting, Vocera users can still press the Call button and say "Skip" to advance to the next message when they play their text messages or voice messages aloud.

To enable or disable the "Skip" response for VMI messages:

1. On each Vocera Voice Server node, open the `\vocera\server\properties.txt` file in a text editor.
2. Add the MsgDisableSkipMessageResponse property.
3. Set the value of the property to one of the following values:
   - True – disables the "Skip" response.
   - False – enables the "Skip" response.
4. Save the `properties.txt` file.
5. Stop the Vocera Voice Server and start it again. The Vocera Voice Server loads `properties.txt` into memory.

Note: If you have a Vocera Voice Server cluster, stop and start the standby nodes first, and then switch to the active node and choose Cluster > Failover in the Vocera Control Panel.

Configuring Button Responses for VMI Messages

By default, when alerts and alarms are sent to a badge via the Vocera Messaging Interface, the user must respond by either using voice commands or by selecting responses from a menu on the device. In a noisy environment, it may be difficult to respond to urgent VMI messages using voice commands. For faster and more accurate responses, you can configure the Vocera system to allow users to respond using the Call or DND buttons on the device.

Important: If you are considering enabling button responses for urgent VMI messages, note the following:

- If your Vocera system has implemented multiple VMI clients that use urgent message delivery, the response choices MUST be consistent across all VMI clients. If the response choices are different across VMI clients, DO NOT enable button responses for urgent VMI messages.
- If you choose to make this configuration change, make sure you adequately train users on the new behavior. Otherwise, users will not know how to use buttons to respond to urgent VMI messages.
- You cannot use the Call or DND buttons to respond to VMI messages that you play aloud later using the "Play Text Messages" command.

Here is an example showing several button response properties for VMI messages:

```
VMITouchCallResponse = accept
VMITouchDNDResponse = reject
VMITouchCallHoldResponse = call back
VMIResponseTimeout = 15
VMITimeoutResponse = negative
VMIResponseMapping = accept, affirmative, reject, negative
```

If you omit VMITouch* properties from the `properties.txt` file, pressing the Call or DND buttons while a VMI message is being played aloud does not send a response. Pressing the Call or DND buttons while an urgent VMI message is being played for the first time cancels message play. If you play a VMI message using the "Play Text Messages" command, pressing DND cancels message play, and pressing the Call button suspends message play and allows you to use voice commands (for example, "Save" and "Delete") to manage the message. See **Managing VMI Messages**.

To configure button responses for VMI messages:

1. On each Vocera Voice Server node, open the `\vocera\server\properties.txt` file in a text editor.
2. Add one or more of the following button response properties to the file. You do not need to add all of these properties.

Property	Description
`VMITouchCallResponse`	Enter the Vocera response phrase that is used when a user presses the Call button to respond to a new VMI message. Example: To make pressing the Call button equivalent to the "accept" response, enter the following property: `VMITouchCallResponse = accept`
`VMITouchDNDResponse`	Enter the Vocera response phrase that is used when a user presses the DND button to respond to a new VMI message. Example: To make pressing the DND button equivalent to the "reject" response, enter the following property: `VMITouchDNDResponse = reject`
`VMITouchCallHoldResponse`	Enter the Vocera response phrase that is used when a user presses and holds the Call button to respond to a new VMI message. Example: To make pressing and holding the Call button equivalent to the "call back" response, enter the following property: `VMITouchCallHoldResponse = call back` Note: Do not omit the space in "call back".
`VMIResponseTimeout`	Controls the maximum time (in seconds) that a user can be prompted to respond to a new alert or alarm. The default is 0, which means that no explicit response timeout is specified, although the speech port timeout will take effect after 3 minutes. Example: The following property sets the VMI response timeout to 15 seconds: `VMIResponseTimeout = 15`
`VMITimeoutResponse`	Controls the response that is used when a new alert or alarm reaches the specified `VMIResponseTimeout`. There is no default value. If the `VMIResponseMapping` property is specified (see below), enter a mapped response value, not a middleware response value. The "call back" response value cannot be used for this property. If no value or an invalid value is specified, no response is sent, which is equivalent to saying "skip" to skip the alert or alarm. Example: The following property sets the response to "reject" whenever the response timeout is reached for an alert or alarm: `VMITimeoutResponse = reject`

Property	Description
`VMIResponseMapping`	Maps VMI responses passed from a middleware system to other response choices. Enter a comma-separated list of values where odd-numbered values represent the middleware response choices and even-numbered values represent response choices that users will see and hear on their Vocera devices. The values are case-insensitive. Example: The following property maps "accept" and "reject" to "affirmative" and "negative," respectively: `VMIResponseMapping = accept, affirmative, reject, negative`

3. Save the `properties.txt` file.
4. Stop the Vocera Voice Server and start it again.

Note: If you have a Vocera Voice Server cluster, stop and start the standby nodes first, and then switch to the active node and choose Cluster > Failover in the Vocera Control Panel.

Configuring Urgent VMI Messages

The Vocera Voice Server plays one-way, urgent messages aloud to a large number of recipients without the need to configure your environment for multicast or turn on Vocera Voice Server properties.

In previous versions of Vocera Voice Server an urgent VMI message was sent to a group of users utilizing a unicast message. Since urgent messages are played aloud, a separate speech port was needed for each recipient.

The problem with this functionality is when your environment requires the ability to send urgent messages to a large group. In this instance, the Vocera Voice Server might run out of available speech ports, causing delivery of the message to be delayed for some recipients.

Prior to Vocera Voice Server version 5.2.0, the only way to reduce the number of speech ports, was the difficult task of configuring multicast, rather than unicast messages, between the Vocera Voice Server and Vocera client devices. In addition, you had to turn on the VMIBroadcastEnabled property in the Vocera properties file.

The Vocera Voice Server has been enhanced so that unicast messages always utilize a single speech port and you are no longer required to perform additional configuration tasks in your environment.

Configuring VMI Telephony Properties

If your Vocera installation includes Vocera Telephony Server for telephony integration, you can specify property values in the `\vocera\dialogic\telproperties.txt` file that define how a Vocera badge interacts with telephony equipment via a VMI client application.

Note: If you use Vocera SIP Telephony Gateway for telephony integration, you can configure trunk access codes (or TACs) to specify how specific dial strings are processed.

These properties are designed for use with VMI applications in the following situations:

- You need to adjust the badge volume for calls from other devices. For example, if badge users are having trouble hearing calls from bedside speakers in a nurse call system, these properties can help.
- A system requires a special key sequence to end a device-to-badge call after the badge user hangs up.

The collection of VMI telephony properties must be complete. If you comment out one property in the collection, you must comment out the entire collection. You can, however, specify one or more empty values for a property in this collection. By default, the VMI telephony properties are undefined and therefore disabled. The following table describes these properties.

Table 4: VMI telephony properties

Property	Description
TelVMIDeviceTAC	Specifies a trunk access code (TAC) to identify a device (such as a nurse call system) that connects to a PBX to communicate with a Vocera badge. By default, this property value is not defined. When it is defined, this value activates the gain specified by the corresponding TelVMIRxGain property, and the macro defined by the corresponding TelVMIHangUpMacro property. The TAC for any given device is set by the PBX administrator. To specify TACs for multiple devices, use a forward slash (/) as a separator character. White space is ignored. You can specify up to 50 TACs. If two or more TACs begin with the same sequence of characters, list them in descending order of length. For example, each of the following TACs begins with the sequence 12: 12, 123, and 12345. In the properties file, you would list them in the following order: `12345 / 123 / 12`
TelVMIRxGain	Specifies how much gain is added when a badge user chooses the callback option to respond to a VMI message. By increasing or decreasing this value, you increase or decrease the sound level (volume) of the badge speaker in 6 dB increments. For example, a value of 3 increases the volume by 18 dB (3 * 6 = 18). Valid values range from 0 to 6, inclusive. By default, this property value is not defined. Optimum values should be determined in the field by trial and error. The specified gain is applied only if the corresponding TelVMIDeviceTAC property is defined. The gain is removed when the call ends. To specify gains for multiple devices, use the forward slash (/) as a separator character. White space is ignored. You can specify up to 50 gain values.
TelVMIHangUpMacro	Specifies a sequence to dial when a Vocera badge ends a call initiated using the callback option in response to a VMI message. This property is especially useful when interacting with a device that connects to a PBX via an analog line. The required sequence varies depending on the device. For example, nurse call systems from different vendors require different hang-up sequences. Consult the device documentation for details. The specified sequence is dialed only if the corresponding TelVMIDeviceTAC property is defined. To specify more than one macro, use the forward slash (/) as a separator character. White space is ignored. You can specify up to 50 macros.

When you specify more than one value for any of these properties, the order is important:

- If two or more TACs begin with the same sequence of characters, list them in descending order of length when you specify values for the TelVMIDeviceTAC property.

  Vocera's parser processes a dial string from left to right, and when it finds a sequence of digits that matches a value specified for TelVMIDeviceTAC, it interprets that sequence as the TAC portion of the dial string. Therefore, given a dial string of 1234914087904100 and two TelVMIDeviceTAC property values listed in the order 12/1234, the parser interprets the first match, 12, as the TAC. However, when the same property values are listed in the order 1234/12, the first match is 1234.
- The TelVMIRxGain and TelVMIHangUpMacro values are associated with a TelVMIDeviceTAC value, so you must list all property values in the same order. That is, the first TelVMIDeviceTAC value corresponds to the first TelVMIRxGain value and the first TelVMIHangUpMacro value, and so on.

For example, suppose a VMI client application interacts with a nurse call system made by company NC1, a blood pressure monitoring system made by company BP, and another nurse call system made by company NC2. The following code lists some sample values for this scenario:

Specifying VMI telephony properties

```
#                    NC1   BP    NC2
#-------------------------------------
TelVMIDeviceTAC   = 835 / 7812 / 781
TelVMIRxGain      =  4  /      / 2
TelVMIHangUpMacro = ##  /      / *9*
```

In this example, a gain value of 4 and the hang-up macro ## are defined for nurse call system NC1, which has the TAC 835. Similarly, a gain value of 2 and the hang-up macro *9* are defined for nurse call system NC2, which has the TAC 781. However, the blood pressure monitor BP, which has the TAC 7812, does not define a gain value or a hang-up macro. In the properties file, such "empty" values can either be omitted or specified explicitly with spaces. Also, the TAC for BP is listed before the TAC for NC2 because both TACs begin with the sequence 781 and the TAC for BP is longer than the TAC for NC2. Listing the TACs in this order ensures that the Vocera parser will extract them correctly from a dial string.

Using the Vocera Messaging Interface

VMI client applications communicate with the Vocera Voice Server via a dynamic link library (DLL) and header files provided by Vocera. The header files define a C++ API. The API is specified by the C++ classes `VMI` and `VMIListener`, defined in `vmi.h`. The library `vmi.dll` implements VMI, and is either statically or dynamically linked to the client application. The VMI DLL communicates with the Vocera Voice Server through an internal TCP socket.

For detailed information about VMI classes and methods, see VMI API Reference. The source code for a working VMI client application is provided in the `VMI` directory on the **Vocera Developer Kit** CD. The key files are `vmitest.cpp` and `vmitest.h`.

Using the VMI Class

The VMI class contains methods for communicating with the Vocera Voice Server. The most important methods in the VMI class are:

- Open Method
- Message Method
- Close Method

Open Method

The `Open` method establishes a TCP connection to the Vocera Voice Server. It must be called before any other method. The `Open` method takes three arguments: a string that uniquely identifies the client, a comma-separated string containing the IP address(es) of the Vocera Voice Server(s), and a pointer to a `VMIListener` object that must be implemented by the client application developer. The VMI DLL uses the `VMIListener` object to send acknowledgements and other messages back to the client.

Important: Each VMI client must use a unique ID. If the same client ID is used for two different VMI connections at the same time, the Vocera Voice Server will automatically drop the earlier connection, log a warning to the server log, and send a warning email to the Vocera Voice Server alert recipient(s).

The following code, simplified for readability, instantiates a subclass of `VMIListener` and passes it to the `Open` method. It establishes a connection to a single Vocera Voice Server. See `vmitest.cpp` for a more complex example.

Opening a connection

```
#include <vmi.h>
class MyListener : public VMIListener {
public:
  void HandleAck(long iMessageID,
                 char* sLoginID,
                 int iAckCode)
  {}

  void HandleResponse(long iMessageID,
```

```
                        char* sLoginID,
                        char* sResponse)
  {}

  void HandleConnectionFailed(void)
  {}
};

  int main() {
  VMI* vmi = new VMI();
  MyListener* myL = new MyListener();
  int iResultCode = vmi->Open("MyClient", "192.168.15.10",
 myL);
  return iResultCode;
}
```

To establish a connection with a clustered Vocera Voice Server, call the `Open` method and specify a comma-separated string containing up to four IP addresses for the `sVoceraIPAddr` parameter. This ensures continued interoperability with the Vocera Voice Server after a failover occurs. The following method call establishes a connection with a cluster of three Vocera Voice Servers.

Opening a connection to a Vocera cluster

```
char* sServerIPList =
 "192.168.15.10,192.168.15.11,192.168.15.12";
int iResultCode = vmi->Open("MyClient",
                            sServerIPList,
                            myL);
```

Message Method

The `Message` method sends a message to a badge via the Vocera Voice Server. In addition to the message text, you can specify the recipient, message priority, an optional set of responses from which the recipient can choose, and an optional custom alert tone.

VMI messages sent to a badge carry the following information:

Table 5: Message data

Parameter	Description
Message ID	VMI uses the combination of message ID and client name to return acknowledgments and other messages to the sender of a message. Therefore, for any client that opens a connection to the Vocera Voice Server, message IDs must be unique. For example, if client ABC sends a message with ID 123, it should not send another message with ID 123. However, client XYZ could send a message with ID 123. The message IDs are the same, but the client names are different. A timestamp is one way to keep message IDs unique.
Login ID	One of the following: • The user ID of the recipient user, given in the user's profile on the Vocera Voice Server. • The name of a Vocera group, or a group's phone extension.
Message	The message text.
Ring tone	In the current version, this value is always 0.
Priority	Set to 2 for an urgent message; set to 1 for a high-priority message; set to 0 for a normal priority message.
Optional callback phone number	A phone number that the recipient can call to respond to the message.
Optional list of responses	A comma-separated list of up to five responses (for example: Yes, No) from which the recipient can choose using a menu on the badge. Each response choice cannot exceed 15 characters.
Optional audio file used for the alert tone	A custom audio file used for the alert tone for the message.

The following code shows how to send a message. This simplified example assumes that a pointer to the VMI class has been instantiated, and that a connection to the server has been opened.

Sending a message

```
/*
  Assumptions:
  VMI* vmi has been instantiated.
  vmi->Open succeeded.
*/
  long lMsgId = 123;
  char* sUser = "jsmith";
  char* sMsgText = "Hello";
  int iRingTone = 0;            // Always 0 in this version
  int iPriority = 0;            // Normal priority
  char* sPhoneNo = "555-1212";  // Callback phone number
  char* sResponses = "Yes,No";  // Comma-separated list
  char* sWAVFiles = "";
  int iResult = vmi->Message(lMsgId,
                             sUser,
                             sMsgText,
                             iRingTone,
                             iPriority,
                             sPhoneNo,
                             sResponses,
                             sWAVFiles);
```

The `Message` method returns an integer that represents a result code (see **VMI Result Codes** for a complete list). The code `rcAccepted` is returned to indicate success. If, for example, the `rcCouldNotConnect` code is returned, it could mean that the Vocera Voice Server IP address was incorrectly specified, or that the Vocera Voice Server is not running at the moment. The client application may want to try again after a certain time.

Custom Alert Tones and Audio Prompts

You can use custom audio files with VMI messages, either as the alert tone for the message or as an audio prompt contained within the message.

Table 6: Vocera Voice Server locations of custom alert tones and audio prompts

Folder	Description
`\vocera\config\custom\prompts`	Location of custom alert tones Note: After you place a WAV file in this folder, stop and start the Vocera Voice Server to force devices to download the custom alert tone.
`\vocera\data\prompts\custom`	Location of custom audio prompts

Required Format of Audio Prompt Files

If you create custom audio prompt files to use with Vocera, the WAV files must have the following format:

Audio Format: 16 bit Monophonic WAV PCM

Sampling Rate: 8000 samples/second

Important: Make sure audio prompt files used for alert tones are short in duration (no more than 2 seconds).

Using Custom Alert Tones

The `sWAVFiles` parameter of the `Message()` method can be used to send a VMI message with a custom alert tone. VMI alert tones behave differently depending on the priority of the message. A normal priority message plays the `sWAVFiles` audio file once, a high priority message plays the `sWAVFiles` audio file twice, and an urgent message plays the `sWAVFiles` audio file twice followed by the prompt, "Urgent Message."

In the following example, the WAV file specified for the `sWAVFiles` parameter is `normal.wav`, a custom audio file that has been placed in the `\vocera\config\custom\prompts` folder. That WAV file will be the alert tone that is played on the badge. This example assumes that a pointer to the VMI class has been instantiated, and that a connection to the server has been opened.

Sending a message using a custom alert tone

```
/*
  Assumptions:
  VMI* vmi has been instantiated.
  vmi->Open succeeded.
*/
  long lMsgId = 1011;
  char* sGroup = "N I C U Nurse";
  char* sMsgText = "REMINDER: Staff meeting at 3 p.m.";
  int iRingTone = 0;     // Always 0 in this version
  int iPriority = 0;     // Normal priority
  char* sPhoneNo = "";   // No callback phone number
  char* sResponses = ""; // No response needed
  char* sWAVFiles = "normal.wav";
  int iResult = vmi->Message(lMsgId,
                             sGroup,
                             sMsgText,
                             iRingTone,
                             iPriority,
                             sPhoneNo,
                             sResponses,
                             sWAVFiles);
```

Using Custom Audio Prompts in the Text of a Message

You can also play custom audio prompts within the text of a message. The message text in the following example includes references to two audio prompts: `bp` (the prompt for "blood pressure") and `room`.

Playing custom audio prompts

```
/*
  Assumptions:
  VMI* vmi has been instantiated.
  vmi->Open succeeded.
*/
  long lMsgId = 1011;
  char* sUser = "jdassin";
  char* sMsgText = "Please check patient bp in room 304.";
  int iRingTone = 0;            // Always 0 in this version
  int iPriority = 0;            // Normal priority
  char* sPhoneNo = "555-1212";  // Callback phone number
  char* sResponses = "Accept,Reject";  // Comma-separated
 list
  char* sWAVFiles = "";
  int iResult = vmi->Message(lMsgId,
                             sUser,
                             sMsgText,
                             iRingTone,
                             iPriority,
                             sPhoneNo,
                             sResponses,
                             sWAVFiles);
```

You can create your own custom prompt files and use them in messages. The custom prompt filename must start with an underscore character ("_") and the file must be placed in the following folder:

```
\vocera\data\prompts\custom
```

When you reference a custom prompt file in a VMI text message, leave out the initial underscore character and the filename extension. For example, if the prompt file is named `_xray.wav`, type `xray` in the message.

Vocera automatically maps some text characters in a message, such as + and @, to audio prompt files. For a list of text characters converted to prompts, see **Message**.

Close Method

The `Close` method closes the TCP connection to the Vocera Voice Server, and frees resources used by VMI.

The following code shows how to close a VMI connection. This simplified example assumes that a pointer to the `VMI` class has been instantiated, and that a connection to the server has been opened.

Closing a connection

```
/*
 Assumptions:
 VMI* vmi has been instantiated.
 vmi->Open succeeded.
*/
vmi->Close();
```

Using the VMIListener Class

The `VMIListener` class provides a callback interface that a VMI client application developer must implement as a derived class. An object of this derived class must be supplied as the second argument to the `Open` method of the `VMI` object. The methods of this class are called from the VMI DLL when an acknowledgement or response notification arrives from the Vocera Voice Server, or if the connection to the Vocera Voice Server fails.

The `VMIListener` methods are:

- HandleAck
- HandleResponse
- HandleConnectionFailed

HandleAck Method

The VMI DLL calls a client's implementation of the `HandleAck` method to send an acknowledgement (for example, when a message is delivered) as opposed to a response (when a recipient chooses from a list of responses supplied with a message).

The following code example shows how the `HandleAck` method is implemented in `vmitest.cpp`.

Handling message acknowledgments

```
void VMITest::HandleAck(long iMessageID,
                        char* sLoginID,
                        int iAckCode)
{
  static char* sAckCodes[] =
  {
    "Delivered",
    "Read",
    "Call Started",
    "Call Ended"
  };
  char sPrompt[cMaxPrompt];
  Print("\n");
  sprintf(sPrompt,
          "Ack for Login ID %s and message id %ld: %s",
          sLoginID, iMessageID, sAckCodes[iAckCode]);
  Print(sPrompt);
}
```

Delivery status notifications sent from the Vocera Voice Server back to the client carry the following information:

Table 7: HandleAck data

Parameter	Description
Message ID	Identifies the message in question.
Login ID	Identifies the recipient.
Acknowledgement code	An enumeration member that represents one of the following events: • Message was successfully delivered to the badge • Message was read (or played out loud) by the recipient • Callback phone call started • Callback phone call ended

You can implement `HandleAck` to do more than print status messages. For example, if the interval between receiving a message-delivered acknowledgement and a message-read acknowledgement is too long, a client application could take action (for example, send the message to another user).

HandleResponse Method

The VMI DLL calls a client's implementation of the `HandleResponse` method to signal that the recipient has chosen one of the response picks supplied with the message.

The following code shows how `HandleResponse` is implemented in `vmitest.cpp`.

Handling message responses

```
void VMITest::HandleResponse(long iMessageID,
                             char* sLoginID,
                             char* sResponse)
{
  char sPrompt[cMaxPrompt];
  Print("\n");
  sprintf(sPrompt,
          "Response for Login ID %s and msg id %ld: %s",
          sLoginID, iMessageID, sResponse);
  Print(sPrompt);
  Print("\n");
}
```

Notifications signaling a response choice picked by the recipient carry the following information:

Table 8: HandleResponse data

Parameter	Description
Message ID	Identifies the message in question.
Login ID	Identifies the recipient.
Response	A string representation of the recipient's response.

Recipients can respond to a message more than once. The client is notified each time a response choice is made.

VMI uses the combination of message ID and client name to return acknowledgments and other data to the sender of a message. Therefore, for any client that opens a connection to the Vocera Voice Server, message IDs should be unique. For example, if client ABC sends a message with ID 123, it should not send another message with ID 123. However, client XYZ can send a message with ID 123. A timestamp is one way to keep message IDs unique.

HandleConnectionFailed Method

The VMI DLL pings the Vocera Voice Server periodically. If it doesn't receive a response, the VMI DLL calls the client's implementation of the `HandleConnectionFailed` method to signal that the TCP connection to the Vocera Voice Server has failed.

The following code shows how `HandleConnectionFailed` is implemented in `vmitest.cpp`.

Handling a failed connection

```
void VMITest::HandleConnectionFailed(void)
{
  Print("Connection failed - socket closed\n");
  bOpen = false; // global variable indicates gateway
 status.
}
```

The following diagram shows the flow of messages and events among the client application, VMI DLL, and Vocera Voice Server as the VMI DLL tests the connection to the Vocera Voice Server and calls the client application when the connection is lost.

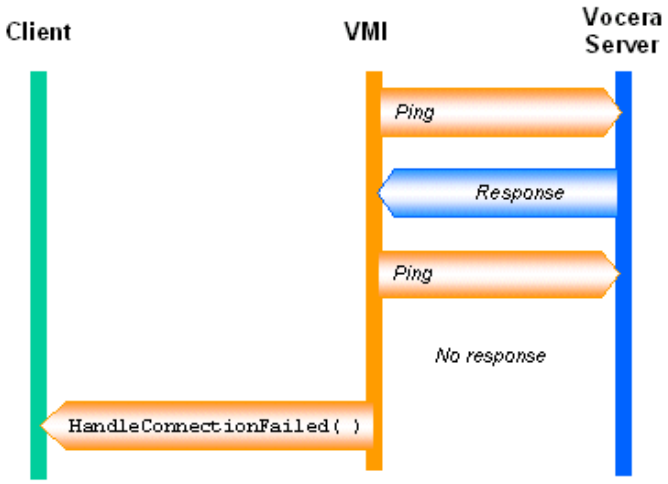

Figure 2: Flow of events for a failed connection

A client application might miss an acknowledgement or response if the connection to the Vocera Voice Server is down at the time of the event in question; notifications are not buffered in the Vocera Voice Server in such cases.

Parameter Validity Checking

The VMI DLL performs basic validity checking of VMI method parameters. If a parameter for a VMI method call in your client application is invalid because it exceeds the maximum allowed size for a value or is out of the range of valid enum values, the client will immediately return an `rcFailed` return code. Your client application can interpret the result and handle it appropriately. For the full list of VMI result codes, see **VMI Result Codes**.

Understanding the Flow of Events

The following diagram shows the flow of messages and events among the client application, VMI DLL, Vocera Voice Server, and Vocera badge when the client sends a normal-priority message and the recipient chooses one of the responses supplied with the message.

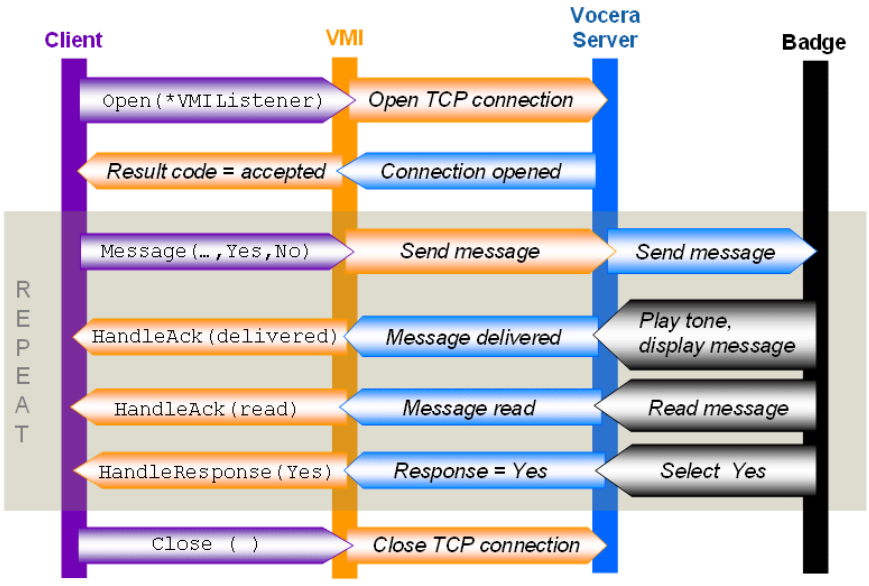

Figure 3: Flow of events for a normal priority message

Here's a description of the diagram, starting from the top left.

1. The client application opens a TCP connection to the Vocera Voice Server. One of the `Open` method parameters is a pointer to a VMIListener object implemented by the client. The VMI DLL uses the VMIListener to call HandleAck and HandleResponse.

   The client uses this connection to send text messages and receive status acknowledgements. Each client must open its own VMI connection. For example, a nurse call system and a supply monitoring application would require separate VMI connections.
2. The VMI DLL returns a result code to indicate a successful connection. The client reuses this connection; you don't have to open a new connection for each VMI message.
3. The client sends a message. The message payload includes a list of possible responses (in this example, the responses are Yes and No).
4. The Vocera Voice Server sends the message to the badge of the specified user.
5. The badge plays a tone and displays the message text.
6. The VMI DLL calls the client's `HandleAck` method with a status code indicating that the message was delivered.
7. The badge user reads the message and presses the Select button.
8. The VMI DLL calls the client's `HandleAck` method with a status code indicating that the message has been read.
9. The badge user chooses the response Yes.
10. The VMI DLL calls the client's `HandleResponse` method with a parameter that contains the user's response.
11. When the client no longer requires a VMI connection, it closes the connection.

Working with VMI Messages

VMI messages are sent to Vocera devices as text. You can read the text on the badge or smartphone, or play them aloud via text-to-speech. A VMI message can also supply audio files containing recordings of the message and response prompts.

VMI messages have three levels of call priority:

- normal—the badge plays a “klunk” and displays the message text on screen.
- high—the badge plays two “klunks” and displays the message text on screen.
- urgent—the badge plays two “klunks”, enunciates "urgent message," and then automatically plays the message aloud.

Note: You can customize the alert tones for VMI messages. See **Using Custom Alert Tones**.

See the *Vocera Badge User Guide* and the *Vocera Smartphone User Guide* for more information about using badges and smartphones, respectively.

Receiving VMI Messages

When either your badge or smartphone receives a VMI message of normal or high priority, it plays an alert tone and displays the text of the message. (The alert tone is different for normal and high priority messages.) You can read the message or play it aloud using text-to-speech (or an audio file, if supplied with the message).

When your badge receives an urgent message, it plays an alert tone and then immediately plays the message. The alert tone for an urgent message is the same as for a high-priority message.

Note: If you are using the badge or smartphone to speak with someone when a normal or high priority message arrives, you cannot play the message until the call ends. However, the badge and smartphone both play urgent messages immediately, breaking into a conversation and putting the original caller on hold if necessary.

The client application specifies the priority of a message. For example, in a nurse call application, calls that originate from a bedside handset may be designated normal, and calls originating from a shower pull or a code blue call alert may be designated urgent.

Playing VMI Messages

When you play a VMI message, the VMI DLL sends a message-read acknowledgement to the client application, and the badge gives you an opportunity to respond. You can use voice commands, button clicks, and the message list to play VMI messages.

Using Voice Commands

You can use voice commands to play VMI messages at any time. Use the same voice commands you use to play other text messages.

To play unacknowledged VMI messages:

1. Press the Call button.
2. Say, “Play text messages.”
3. The badge plays unacknowledged messages and responses (if any), starting with the newest unacknowledged message.

To play VMI messages you have already acknowledged:

1. Press the Call button.
2. Say, “Play old text messages.”
3. The badge plays acknowledged messages and responses (if any), starting with the newest acknowledged message.

Using Button Clicks on a Badge

Your badge displays the most recent message you have received until you play it or read it or press the Call button.

To play a new message using button clicks:

1. Press the Select button twice. The badge plays the message.
2. After playing the message, the badge sends an acknowledgement to the client, plays response prompts (if any), and then displays the message list.

Using the Message List

The message list displays the date and time you received each VMI message, preceded by one of the following icons:

- An open envelope icon indicates a message you have played or read.
- A closed envelope icon indicates a message you have not listened to or read.

The message list presents the most recent text message first. In some situations, however, you may want to play older messages. For example, if your badge receives two or three messages while you are too busy to acknowledge them, you may want to play all of them when you have the opportunity. In this situation, use the message list to play messages.

To play a message in the badge message list:

1. If the message list is not displayed, you can display it from the main screen by pressing the Select button twice.
2. Use the Up and Down buttons to scroll through the list of messages until you select the message you want.

   By default, the most recent message is at the top of the list, the message you received previously is next, and so on.
3. Press the Select button three times.

   The badge plays the message, then plays response prompts (if any).
4. The device displays the message list again after you play a message. You typically use the message list only to continue playing messages after listening to the most recent message.

To display the badge message list when an unanswered message is not visible:

You can display the message list when there is no unanswered message on the badge.

1. Look at the display screen to make sure the message list is not already visible.
2. Press the Select button twice to display the message list.

Reading VMI Messages

You can read VMI messages on a badge or smartphone and use menu choices to respond.

To read the most recent unacknowledged VMI message on a badge:

1. Look at the message on the display screen.
   The badge displays the most recently received message until you press the Select button or the Call button.
2. Press the Select button.
   The badge displays a menu of responses.
   The menu displayed on your badge may be slightly different. For example, if the message was sent with a list of responses, the menu would include those responses. If the message was sent without a callback number, the menu would not include the CALL option.
3. Use the Up and Down buttons to choose a response.
4. Press the Select button.
   The badge sends the response, then displays the message list.

To read a VMI message on a badge using the message list:

1. Use the Up and Down buttons to scroll through the list of messages until you find the message you want.
2. Press the Select button.
   The badge displays the text of the message.
3. Read the message, then press the Select button.
   The badge displays a menu of responses.
4. Use the Up and Down buttons to choose a response.
5. Press the Select button.
   The badge automatically sends the messaging system an acknowledgement, performs the requested action, and then displays the message list again.

To read a VMI text message immediately on a smartphone:

1. When the VMI message is received, an alert window displays the message. Press Open Message to open the message and respond to it.
2. The smartphone displays the body of the message, the sender's name or email address, and the date and time the message was received by the Vocera Voice Server.
3. To respond to the message, press Menu, and then choose one of the response choices.
   If you choose Play, the message is played aloud, and you can use voice commands to respond to the message. See **Responding to Played Messages Using Voice Commands**.

To open a VMI text message from the New Messages list on a smartphone:

1. On the Home screen, press the Vocera Apps soft key. The Vocera Apps screen appears.

Figure 4: Vocera Apps screen

2. Press the Navigation keys—up, down, left, or right ( )—to highlight the Messaging app.

You can also press Menu > Messaging to select the Messaging app from a menu.

3. Press the center key

to select the Messaging app.

4. On the New Messages tab, use the navigation key to scroll through the list of messages until you see the message you want to read. Newest messages are listed first. Press the center key to open it.
5. The smartphone displays the body of the message, the sender's name or email address, and the date and time the message was received by the Vocera Voice Server.
6. To respond to the message, press Menu, and then choose one of the response choices.

Responding to VMI Messages

You can use voice commands, button clicks, and menu commands to respond to messages. Urgent VMI messages are automatically played aloud.

Responding to Played Messages Using Voice Commands

Text messages that are played aloud (such as urgent messages) play a beep when the message is finished. After the beep, you have approximately 1.5 seconds to say one of the valid responses before the Genie begins to prompt for a response. This allows you to respond quickly to the message.

After the 1.5-second interval, the Genie announces the responses you can say. You can also call back to the sender (by saying “Call back”) or skip the message (by saying “Skip”).

To respond to an unacknowledged VMI message that is played aloud:

1. Listen to the message, and wait for the beep to indicate that the message is finished.
2. Within 1.5 seconds after the beep, say one of the valid responses.
3. If you don't know what the valid responses are, do one of the following:
   - Wait for the Genie to announce them, and then say your response.
   - Press the Select button, and then use the Up and Down buttons to choose a response. Press Select again to send the selected response.

Responding to Played Messages Using Buttons

You Vocera system can be configured to allow button responses to urgent VMI messages. This feature allows you to respond to an urgent VMI message using the Call or DND buttons on the device. Check with your Vocera system administrator to see if this feature has been enabled.

Note: You cannot use the Call or DND buttons to respond to VMI messages that you play aloud later using the "Play Text Messages" command.

Responding to Read Messages Using Menu Commands

After you read the text of a VMI message and press Select, the badge displays a list of responses for you to choose from. The list usually includes the following menu choices, along with responses (if any) sent with the message.

Table 9: Badge Message menu commands

Command	Description
PLAY	Converts a text message to a spoken message and plays it for you.
CALL	Available only when a callback number is sent with the message. Initiates a call to the specified number.
TO NEXT MSG	Skips to the next message in the list.

Command	Description
DELETE MSG	Erases the message from the badge memory and from the Vocera Voice Server.
SAVE MSG	Saves the message and prevents it from being automatically deleted. You are limited to 20 text messages at a time, and you can save up to 10 of these messages. Saved messages are maintained on the server until you delete them. They are not affected by scheduled sweeps of the server.
BACK TO LIST	Returns to the list of text messages, where you can select another message.
EXIT MENU	Returns to the main screen.

Saving and Deleting VMI Messages

You typically do not have to save or delete VMI messages. Vocera automatically stores up to 20 text messages for you, regardless of whether you explicitly save them.

Note: Undelivered messages are not stored.

If you exceed your message limit, Vocera automatically deletes the oldest message and stores the most recently received one. In addition, Vocera also routinely sweeps and deletes old messages, regardless of whether they have been read, after storing them for a number of days determined by the system administrator. The default sweep age for text messages is 2 weeks.

Because VMI messages are typically time-critical, this automated message management is all that most users need. However, you can also explicitly save and delete text messages. See the *Vocera Badge User Guide* for complete information.

Managing VMI Messages

You can issue voice commands to save, delete, and navigate while playing VMI messages, as described in **Playing VMI Messages**. These commands do not let you respond to VMI messages, they help you manage them.

The following table summarizes the voice commands you can use while playing messages. To use these commands, press the Call button while playing the message, then say the command. To stop playing a message, press the Hold/DND button, or press the Call button twice.

Table 10: Voice commands for working with messages

Action	Recommended Voice Commands	Alternative Forms
Delete the message you just played or are in the process of playing.	Delete.	Erase.
Save the message you just played or are in the process of playing.	Save.	Archive.
Play the next message.	Next.	Skip.
Replay the current message.	Repeat.	
Get the time the message was received.	Time stamp.	Time.
Get the date the message was received.	Date stamp.	Date.
Cancel message play.	Cancel.	Goodbye.

Frequently Asked Questions

This section lists common question and answers related to how to use VMI.

What is the Vocera Messaging Interface?

The Vocera Messaging Interface (VMI) is an application programming interface (API) that enables text messaging between external systems and Vocera badges and smartphones via the Vocera Voice Server. VMI allows a **client** (for example, a nurse call or patient monitoring system) to send a text message to a badge or smartphone, and to receive acknowledgements that describe the delivery status of the message, along with optional responses from a message **recipient**.

What happened to the Nurse Call Interface (NCI)?

VMI replaces the nurse call interface (NCI) offered with previous versions of Vocera. In particular, the NCI `Call` method has been replaced by the `Message` method, which has a different signature. Also, the VMI `Open` method has a different signature and behaves differently from the NCI `Open` method.

Which healthcare, hospitality, and middleware solutions are supported with VMI?

For a list of vendors of nurse call, patient monitoring, patient flow, medical telemetry, hospitality, and middleware systems, and other solutions that Vocera has integrated with using VMI, contact your Vocera representative.

How do I use VMI to connect to a Vocera cluster?

When you call the VMI `Open` method to establish a connection, specify multiple comma-separated IP addresses for the `sVoceraIPAddr` parameter. A cluster ensures continued interoperability with the Vocera Voice Server after a failover occurs. If a VMI connection fails (as it will in the event of a failover), your code should repeatedly try to open a new connection until it succeeds.

Can you use VMI to connect to localhost or 127.0.0.1?

No. You cannot connect to localhost or 127.0.0.1. When you call the VMI `Open` method to establish a connection, you must specify the real IP address(es) of the Vocera Voice Server. The value you specify should be equal to the value of the VOCERA_LOCAL_HOST_ADDRESS environment variable on the Vocera Voice Server.

How does a badge user read a VMI message?

When a VMI message arrives on the Vocera badge, the message text is displayed on the badge immediately. A badge user can read the message on the badge LCD or press the Select button twice to play the message aloud. Urgent VMI messages are automatically played aloud.

When a VMI message is played aloud, a beep tone will sound when the message is finished playing. At that point, the user can say one of the valid responses, such as “Reject” or “Accept.” After 1.5 seconds, the user is placed into an interactive voice-driven menu where the valid responses to the message are announced to assist the user. The user can say one of the valid responses, call back to the sender, or skip the call.

How does a Vocera smartphone user read a VMI message?

The experience is very similar to reading a VMI message on a badge. When a VMI message arrives on the Vocera smartphone, the message text is displayed on the phone immediately. The user can read the message onscreen or press Open Message to open it. Urgent VMI messages are automatically played aloud on the smartphone, and users can respond to the message using voice commands.

What kinds of messages can be sent to a badge from VMI?

VMI can send text messages, and a VMI message can include audio files (.WAV files) containing recordings of the message and responses (if any). Vocera users can play text messages aloud using Vocera's text-to-speech feature.

How does VMI distinguish between normal, high priority, and urgent calls?

VMI interprets three levels of call priority – normal, high, and urgent.

- When a Vocera device receives a normal VMI message, it plays a tone (“klunk”) and displays the message text onscreen.
- When a Vocera device receives a high priority message, it plays a different tone (two “klunks”) and displays the message text onscreen.
- When a Vocera device receives an urgent message, it plays a tone (two “klunks”) and then automatically plays the message aloud.

Note: You can customize the alert tones for VMI messages.

Can VMI messages be sent to groups?

Yes. A VMI message can be sent to a group identified by the group name or the Vocera extension stored in a group profile on the Vocera Voice Server.

When I send urgent VMI messages to a large group, why are some of the messages delayed?

The Vocera Voice Server does not allow you to use all available speech ports for an urgent message because that would prevent all other users from making calls. By default, whenever 60% or more of the speech ports are in use, the Vocera Voice Server defers delivery of urgent messages. Normal and high priority messages do not use speech ports, so they are not affected.

If you require the ability to send urgent messages that do not require a response to a large group, you can configure the Vocera Voice Server to use broadcast (rather than unicast) for urgent VMI messages. Only one speech port is used for the broadcast. See **Enabling or Disabling Broadcast of One-Way, Urgent VMI Messages**.

Can VMI determine group and user status?

Yes. The `QueryGroup` method returns a list of group members, and the `QueryUser` method returns user information including iStatus (scNotLoggedIn, scNotOnline, scOnline). Call `QueryGroup`, then loop through the result set and call `QueryUser` to find and enquire about specific users.

The Vocera Voice Server pings each online badge every 30 seconds. As a best practice, do not call `QueryGroup` more than once every 15 seconds. More frequent calls will not yield better data, and may overload the server.

What is the maximum number of characters per message?

The maximum number of characters allowed in a VMI text message is 256 characters.

How are VMI messages managed?

VMI can delete messages from a user's badge by message ID and user ID. The message is removed from the badge regardless of its status (Read, Saved, etc.)

Status and content for text messages are stored on the Vocera Voice Server and redelivered if the user logs in using a different badge.

Up to 20 messages are stored automatically for each user. When the twenty-first message comes in, the first (oldest) stored message is deleted. Saved messages can only be deleted by the user or, if delivered by the VMI interface, by the VMI application.

The Vocera Voice Server Sweep interval and age can also be used to manage text messages. Users can explicitly save messages. These messages are prefixed with "[S]" on the badge display. Up to 10 messages can be saved.

If the saved message count is 10, then a saved message must be "Unsaved" (available from the saved text messages menu on the badge) or deleted prior to saving another message.

Can speech prompts be customized?

Yes. Custom-recorded prompts can be installed and used in conjunction with Vocera's text-to-speech facility.

Some common prompts, such as "BP", "Room", and "Bed", are provided with the Vocera System. Vocera can provide additional such prompts as a professional service.

For more information, see **Custom Alert Tones and Audio Prompts**.

Can VMI run on the same server as the Vocera application?

Yes. However, VMI is only activated when an appropriate license key is entered in the Vocera system, regardless of where the client applications are deployed.

Can multiple VMI clients connect to the Vocera Voice Server?

Yes. VMI supports multiple connections to the Vocera Voice Server. The VMI license determines the maximum number of connections allowed.

For example, the following figure shows two VMI client applications. Suppose that one client connects to a Nurse Call system that supports 100 beds, and the other client connects to a single supply monitor. Each client uses only one VMI connection.

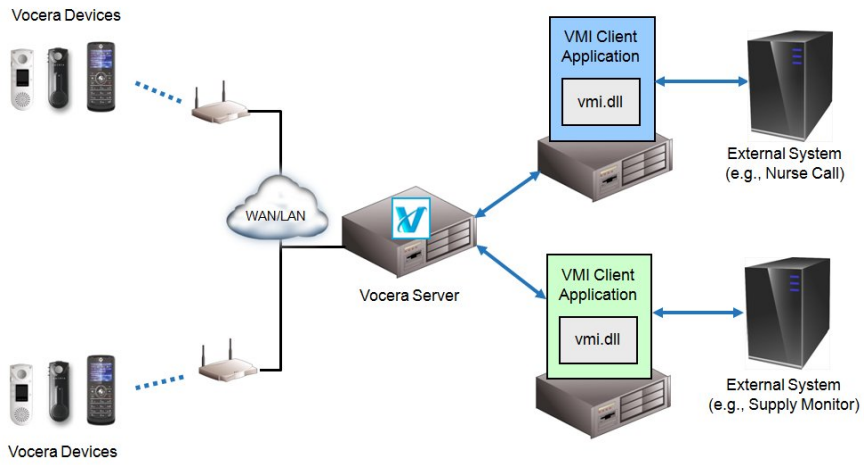

Figure 5: Multiple VMI clients connecting to the Vocera Voice Server

If you have multiple VMI clients, make sure they open connections to the Vocera Voice Server using unique client IDs. If the same client ID is used for two different VMI connections at the same time, the Vocera Voice Server will automatically drop the earlier connection, log a warning to the server log, and send a warning email to the Vocera Voice Server alert recipient(s).

Is VMI thread-safe?

No. VMI calls are not thread-safe, which means they must be synchronized by your application. VMI calls can be made on multiple threads as long as they are made one at a time and not simultaneously. To synchronize VMI calls from a single-process application, use a lock object such as a critical section. For interprocess synchronization, use a mutex object. You can also use a mutex to synchronize VMI calls from a single-process application, but a critical section is faster and more efficient.

VMI API Reference

The VMI API is specified by C++ classes defined in `vmi.h`. The link library `vmi.dll` implements the API, and is either statically or dynamically linked to the external application. The DLL communicates with the Vocera Voice Server through an internal TCP socket.

VMI Class

The `VMI` class contains methods for communicating with the Vocera Voice Server. An external application works by constructing an instance of VMI and calling its methods.

The following table summarizes the `VMI` class methods.

Table 11: VMI class methods

Return type	Signature and description
`int`	`(char* sLoginID, char* sGroupName)` Adds a user to a group.
`void`	`(void)` Closes a connection to the Vocera Voice Server.
`int`	`(long iMessageID, char* sLoginID)` Deletes a message from a badge.
`char*`	`(void)` Returns information about the VMI version.
int	`(long iEventID, char* sEventType, char* sEventInfo)` Logs information about external events—such as those from a middleware system—to Vocera Report Server logs, which can be used to define custom reports.
int	`(long iMessageID, char* sLoginID, char* sText, int iRingTone, int iPriority, char* sPhoneNo, char* sResponses, char* WAVFiles)` Sends a message to a badge.
`int`	`(char* sClientID, char* sVoceraIPAddr, VMIListener* l)` Opens a connection to the Vocera Voice Server.
`int`	`(char* sGroupName, GroupInfo& gi)` Requests information about a Vocera group.
`int`	`(char* sLoginID, UserInfo& ui)` Requests information about a Vocera user.
`int`	`(char* sLoginID, char* sGroupName)` Removes a user from a Vocera group.

AddToGroup

Adds a user to a Vocera group.

Syntax

```
int AddToGroup { char*  | sLoginID } { char*  | sGroupName }
```

Parameters

Parameter	Description
`sLoginID`	The Vocera Voice Server user ID of the user to add.
`sGroupName`	The name of the Vocera group that you are adding the user to. The group name can be further qualified by site. For example, CodeBlue:Cupertino specifies the Code Blue group at the Cupertino site.

Returns

Returns a value defined in VMI Result Codes.

See Also

- QueryGroup
- RemoveFromGroup

Close

Closes a connection to the Vocera Voice Server.

Syntax

```
void Close { void }
```

Parameters

None.

Returns

Void.

See Also

Open

DeleteMessage

Deletes a message from a user's badge.

Syntax

```
int DeleteMessage { long  | iMessageID } { char*  | sLoginID }
```

Parameters

Parameter	Description
`iMessageID`	Identifies the message. Message IDs must be unique for each client that opens a connection to the Vocera Voice Server.
`sLoginID`	Specifies the user ID of the badge user. It cannot be the name of a group or a group's phone extension.

Returns

Returns a value defined in VMI Result Codes.

See Also

- Message
- QueryUser

GetVersion

Returns information about the VMI version. If the VMI version and the Vocera Voice Server version are not compatible, the `Open` method will fail.

Syntax

```
char* GetVersion { void }
```

Parameters

None.

Returns

Returns a string that identifies the VMI version. Example: 4.1

See Also

Open

LogEvent

Logs information about external events—such as those from a middleware system—to Vocera Report Server logs, which can be used to define custom reports.

Syntax

```
int LogEvent { long  | iEventID } { char*  | sEventType } { char*  | sEventInfo }
```

Parameters

Parameter	Description
`iEventID`	A 32-bit integer that uniquely identifies each LogEvent record in the report log per VMI client.
`sEventType`	A string that defines the category of event for Vocera Report Server analysis purposes. The following event types are supported: • `SystemEvent` – records system level event information • `MessageEvent` – links an event to a VMI `MessageID` • `CancelEvent` – cancels an existing event The maximum length of `sEventType` is 25 characters, as specified by the `cMaxEventType` constant. Note: Only `MessageEvent` and `CancelEvent` events will be reported by standard Vocera Report Server Integration report(s). However, you may define your own custom reports to report on `SystemEvent` events.

Parameter	Description
`sEventInfo`	A string consisting of comma-separated items that are specific to the `sEventType`, and provide additional information, such as the `MessageID`, for a given event associated with a particular VMI message. No more than 10 items are allowed in this comma-separated string (as specified by the `cMaxEvents` constant), and each item in the list is a string that cannot exceed 64 characters (as specified by the `cMaxEvent` constant). Leading and trailing spaces in each item of `sEventInfo` will be trimmed. Each supported Event Type has its own reserved items in the `sEventInfo` comma-separated value, and those reserved items must be presented in the beginning of the comma-separated string in the specified order. • `MessageEvent` – There are four reserved items in the following order: "Event_GUID, MessageID, Escalation_Level, Event_Priority" Where • Event_GUID is a global unique identifer (a hexadecimal string) used to link the associated VMI messages • MessageID is an existing VMI message ID • Escalation_Level is the event escalation level defined by middleware vendors • Event_Priority is the event prority defined by middleware vendors • `CancelEvent` or `SystemEvent` – There is only one reserved item: "Event_GUID" Where • Event_GUID is a global unique identifier (a hexadecimal string) that has been used previously in a `MessageEvent` event. Note: Other potential `EventInfo` values corresponding to `SystemEvent` include "Login" or "Logout" as the second item in the comma-separated string.

Returns

Returns a value defined in VMI Result Codes.

See Also

Message

Message

Sends a message to a user's badge.

Syntax

```
int Message { long | iMessageID } { char* | sLoginID } { char* | sText } { int | iRingTone } { int | iPriority } { char* | sPhoneNo } { char* | sResponses } { char* | sWAVFiles }
```

Parameters

Parameter	Description
`iMessageID`	Number that identifies the message. The message ID must be unique for each client that opens a connection to the Vocera Voice Server. The client-ID combination ties asynchronous acknowledgements and responses back to the sender.
`sLoginID`	Specifies the message recipient. VMI checks whether the string contains a user ID of a user, a group's phone extension, or the name of a group, in that order. While checking if the string contains a group's phone extension, alpha characters are excluded.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><code>sText</code></td><td>The message text. Maximum message length (256 characters) is defined in <code>vmi.h</code>. Use the notation <code>[CR]</code> to indicate a line break.
B3000 and B2000 badges can display a 256-character message on the scrollable display. However, on B1000A badges, VMI messages longer than 130 characters will be truncated.
Vocera automatically maps the following text strings to audio prompt files:
<table><tr><th>Character</th><th>Prompt File</th></tr><tr><td>+</td><td>plus.wav</td></tr><tr><td>/</td><td>slash.wav</td></tr><tr><td>@</td><td>at.wav</td></tr><tr><td>. (period)</td><td>point.wav if it precedes digits</td></tr><tr><td>%</td><td>percent.wav</td></tr><tr><td>-</td><td>minus.wav if it precedes digits. dash.wav if it does not.</td></tr><tr><td>0-9</td><td>zero.wav through nine.wav</td></tr></table></td></tr>
<tr><td><code>iRingTone</code></td><td>Reserved for future use. Set to zero.</td></tr>
<tr><td><code>iPriority</code></td><td>Set to 0 for normal priority, 1 for high priority, or 2 for an urgent message.
When a badge receives a normal message, it plays a tone ("klunk") and displays the message text on the LCD. When the badge receives a high priority message, it plays a different tone (two "klunks") and displays the message text on the LCD. When the badge receives an urgent message, it plays a tone (two "klunks") and automatically plays the message aloud.</td></tr>
<tr><td><code>sPhoneNo</code></td><td>(Optional) Phone number for the message recipient to call.</td></tr>
<tr><td><code>sResponses</code></td><td>A comma-separated list of up to five response choices. Each response choice cannot exceed 15 characters.
The choices must consist only of the following characters:

<code>ABCDEFGHIJKLMNOPQRSTUVWXYZ</code>
<code>abcdefghijklmnopqrstuvwxyz</code>
<code>1234567890\'-_</code>

Upper and lower case letters are allowed, but Vocera badges display upper case letters only.
If no response choices are available, specify an empty string (<code>""</code>) as the value of this argument. The <code>NULL</code> value is not acceptable.</td></tr>
<tr><td><code>sWAVFiles</code></td><td>A WAV file to use for the message alert tone. The filename can be up to 64 characters, and it can include spaces. The file must be placed in the <code>\vocera\config\custom\prompts</code> folder on the Vocera Voice Server so that it is available to Vocera devices. If the file is not found, the default alert tone will be used.
For information about the required audio format, sampling rate, and recommended duration of audio files, see Required Format of Audio Prompt Files.</td></tr>
</table>

Returns

Returns a value defined in VMI Result Codes.

See Also

- Custom Alert Tones and Audio Prompts
- DeleteMessage
- Open

Open

Opens a connection to the Vocera Voice Server. If the VMI version and the Vocera Voice Server version are not compatible, the `Open` method fails. This method also fails if you try to open more connections than allowed by the Vocera Voice Server license key. If a VMI connection fails (as it will in the event of a failover), your code should repeatedly try to open a new connection until it succeeds.

Syntax

```
int Open { char*  | sClientID } { char*  | sVoceraIPAddr } { VMIListener*  | l }
```

Parameters

Parameter	Description
`sClientID`	Identifies the sender of the message. Important: Each VMI client must use a unique ID. If the same client ID is used for two different VMI connections at the same time, the Vocera Voice Server will automatically drop the earlier connection, log a warning to the server log, and send a warning email to the Vocera Voice Server alert recipient(s). **Note:** Enter alpha characters only with no numeric values.
`sVoceraIPAddr`	A comma-separated string of Vocera Voice Server IP addresses in numeric format. If you are connecting to a single, standalone Vocera Voice Server, enter only one IP address. If you are connecting to a cluster, specify multiple IP addresses separated by commas. A Vocera cluster can have up to four servers. Note: You cannot specify "localhost" or 127.0.0.1, its equivalent loopback address. You must specify the real IP address on which the Vocera Voice Server is running.
`l`	A VMIListener object to handle callbacks from the server.

Returns

Returns a value defined in VMI Result Codes.

See Also

- Close
- GetVersion

QueryGroup

Requests information about a Vocera group. If the group contains other groups, this method gets user IDs of users in those groups, too, but each user ID is listed only once. This method does not get the names of nested groups.

Syntax

```
int QueryGroup { char*  | sGroupName } { GroupInfo&  | gi }
```

Parameters

Parameter	Description
`sGroupName`	The name of a group on the Vocera Voice Server. The group name can be further qualified by site. For example, CodeBlue:Cupertino specifies the Code Blue group at the Cupertino site.

Parameter	Description

gi	A struct defined in vmi.h:
	`struct GroupInfo {`
	`// # of members in the group.`
	`int cMembers;`
	`// Login names of users in the group.`
	`char saMembers [cMaxMembers][cMaxLoginID];`
	`};`

Returns

Returns a value defined in VMI Result Codes.

See Also

- AddToGroup
- RemoveFromGroup
- QueryUser

QueryUser

Requests information about a Vocera user.

Syntax

```
int QueryUser { char*  | sLoginID } { UserInfo&  | ui }
```

Parameters

Parameter	Description
sLoginID	The user ID of a user on the Vocera Voice Server.

ui	A struct defined in vmi.h:
	`struct UserInfo {`
	`// Status codes defined in vmi.h.`
	`// See enum SC`
	`int iStatus;`
	
	`// The Vocera Voice Server auto-generates`
	`// a serial number for each user.`
	`int iSerialNo;`
	
	`// You could test (bVoiceprint == true)`
	`// before sending a confidential message.`
	`bool bVoiceprint;`
	
	`// Access point location name or MAC address.`
	`char sLocation [cMaxLocationName];`
	
	`char sDeskPhone [cMaxPhoneNo];`
	`char sPagerPhone [cMaxPhoneNo];`
	`};`
	
	Note: Although the Vocera Voice Server automatically generates a serial number for each user, the serial number is not guaranteed to be unique. When you delete users or address book entries, their serial numbers will become available for new users after the administrator restores data from a backup.

Returns

Returns a value defined in VMI Result Codes.

See Also

- QueryGroup

- VMI Status Codes

RemoveFromGroup

Removes a user from a Vocera group.

Syntax

```
int RemoveFromGroup { char*  | sLoginID } { char*  | sGroupName }
```

Parameters

Parameter	Description
`sLoginID`	The Vocera Voice Server user ID of the user to remove.
`sGroupName`	The name of the Vocera group that you are removing the user from. The group name can be further qualified by site. For example, CodeBlue:Cupertino specifies the Code Blue group at the Cupertino site.

Returns

Returns a value defined in VMI Result Codes.

See Also

AddToGroup

VMIListener Class

The `VMIListener` class defines a callback interface that you must implement as a derived class. An object of this derived class is supplied as the second argument to the `Open` method of the `VMI` object. The methods of this class are later called from `vmi.dll` when an acknowledgement or response arrives from the Vocera Voice Server, or if the connection to the Vocera Voice Server fails.

The following table summarizes the `VMIListener` class methods.

Table 12: VMIListener class methods

Return type	Signature and description

`void`	`(long iMessageID,`
	`char* sLoginID,`
	`int iAckCode)`
	Processes an acknowledgement signal from the server.

`void`	`(void)` Processes a TCP connection failure.

`void`	`(long iMessageID,`
	`char* sLoginID,`
	`char* sResponse)`
	Processes a response to a message.

HandleAck

Called by the VMI DLL when it receives an acknowledgement signal from the Vocera Voice Server.

Syntax

```
void HandleAck { long  | iMessageID } { char*  | sLoginID } { int  | iAckCode }
```

Parameters

Parameter	Description
`iMessageID`	Uniquely identifies the message when combined with the name of the sender (client).
`sLoginID`	The Vocera Voice Server login ID of the message recipient. It could be a user ID, a group name, or a group's phone extension.

`iAckCode`	An acknowledgement code defined in `vmi.h`.
	`enum AC`
	`{`
	`acDelivered,`
	`acRead,`
	`acCallStarted,`
	`acCallEnded,`
	`};`

Returns

Void.

See Also

- HandleConnectionFailed
- HandleResponse

HandleConnectionFailed

Called by the VMI DLL when it detects a TCP connection failure.

Syntax

`void HandleConnectionFailed { void }`

Parameters

None.

Returns

Void.

See Also

- HandleAck
- HandleResponse

HandleResponse

Called by the VMI DLL when it receives a response to a message.

Syntax

`void HandleResponse { long | iMessageID } { char* | sLoginID } { char* | sResponse }`

Parameters

Parameter	Description
`iMessageID`	Uniquely identifies the message when combined with the name of the sender (client).
`sLoginID`	The Vocera Voice Server login ID of the message recipient. It could be a user ID, a group name, or a group's phone extension.
`sResponse`	The recipient's response.

Returns

Void.

See Also

- HandleConnectionFailed
- HandleAck

Definitions

This section lists various VMI codes and constants.

VMI Result Codes

When a VMI method returns an integer, you can use the following result codes to interpret the results. The codes are defined in an `enum` named RC in `vmi.h`:

Table 13: VMI result codes

Result code	Description
`rcAccepted`	Operation succeeded.
`rcFailed`	Operation failed.
`rcNotConnected`	Not currently connected - must first Open.
`rcInvalidLoginID`	Invalid LoginID code.
`rcLicenseFailed`	Vocera license violation.
`rcUserNotLoggedIn`	User is not logged in at the moment.
`rcUserNotOnline`	User is not online at the moment.
`rcNoUsersAvailable`	No users available for group message.
`rcMessageNotFound`	Message either does not exist or has been deleted.
`rcInvalidGroupName`	Group with given name does not exist.
`rcInvalidClientID`	Client ID does not satisfy syntactic constraints.

VMI Status Codes

The codes that indicate a badge user's status are defined in an `enum` named SC in `vmi.h`:

Table 14: VMI status codes

Status code	Description
`scNotLoggedIn`	User is not logged in from a badge.
`scNotOnline`	User is logged in, but not currently on the network (possibly in DND mode).
`scOnline`	User is logged in and on the network.

VMI Acknowledgement Codes

The codes that indicate acknowledgement of a VMI message are defined in `enum` named AC in `vmi.h`:

Table 15: VMI acknowledgement codes

Acknowledgement code	Description
`acDelivered`	Message was successfully delivered to recipient.
`acRead`	Message was either opened by the recipient or automatically played aloud.
`acCallStarted`	Callback call started.

Acknowledgement code	Description
`acCallEnded`	Callback call ended.

VMI Priority Codes

The codes that indicate the priority of a VMI message are defined in `enum` named PC in `vmi.h`:

Table 16: VMI priority codes

Priority code	Description
`pcNormal`	0. Normal priority. When a badge receives a normal priority message, it plays a "klunk" tone and then displays the message on the LCD.
`pcHigh`	1. High priority. When a badge receives a high priority message, it plays two "klunk" tones and then displays the message on the LCD.
`pcUrgent`	2. Urgent message. When the badge receives an urgent message, it plays two "klunk" tones and then automatically plays the message aloud.

Maximum Values

The header file `vmi.h` defines the following constants to specify various maximum values.

Table 17: VMI maximum values

Constant	Value	Description
`cMaxClientID`	100	Max ClientID string length.
`cMaxLoginID`	50	Max LoginID string length.
`cMaxText`	256	Max text message size and event log text.
`cMaxPhoneNo`	75	Max phone number length.
`cMaxResponse`	15	Max response choice string length.
`cMaxResponses`	80	Max response choices total string length.
`cMaxWAVFiles`	1000	Max WAV files total string length.
`cMaxLocationName`	101	Max LocationName string length including possible site qualifier.
`cMaxMembers`	100	Max # of members to be returned by QueryGroup.
`cMaxGroupName`	101	Max GroupName string length including optional site qualifier.
`cMaxEventType`	25	Max event type string length, which defines the category of an event.
`cMaxEvent`	64	Max per event item string length in event info for `LogEvent()`.
`cMaxEvents`	10	Max items in event info string for `LogEvent()`.