# Cisco Unified Contact Center Enterprise Application Gateway Interface, Release 12.6(1)

**First Published:** 2021-05-14

**Americas Headquarters**
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

# CONTENTS

# Preface



## Change History

This table lists changes made to this guide. Most recent changes appear at the top:

| Change | See | Date |
|---|---|---|
| **Initial Release of Document for Release 12.6(1)** | | |

## About This Guide

The Enterprise Application Gateway protocol provides a mechanism for Cisco Unified Contact Center Enterprise (CCE) to invoke and communiate with an external application from within the call routing script by using a simple request/response mechanism. Application Gateway requests are managed from within the CCE routing script. It allows data to be passed to and from the application. The call router, under control of the script editor, can make routing decisions based on the results obtained from the application.

## Communications, Services, and Additional Information

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.

- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

### Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

# Field Notice

Cisco publishes Field Notices to notify customers and partners about significant issues in Cisco products that typically require an upgrade, workaround, or other user action. For more information, see *Product Field Notice Summary* at https://www.cisco.com/c/en/us/support/web/tsd-products-field-notice-summary.html.

You can create custom subscriptions for Cisco products, series, or software to receive email alerts or consume RSS feeds when new announcements are released for the following notices:

- Cisco Security Advisories
- Field Notices
- End-of-Sale or Support Announcements
- Software Updates
- Updates to Known Bugs

For more information on creating custom subscriptions, see *My Notifications* at https://cway.cisco.com/mynotifications.

# Documentation Feedback

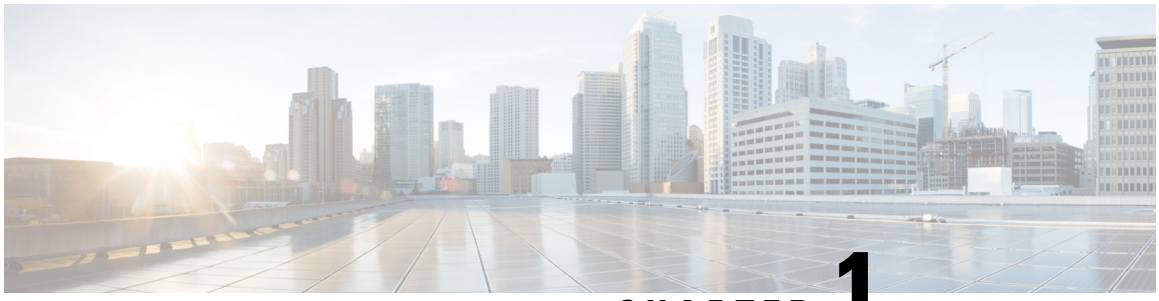
To provide comments about this document, send an email message to the following address: contactcenterproducts_docfeedback@cisco.com

We appreciate your comments.

# Conventions

This document uses the following conventions:

| Convention | Description |
|---|---|
| **boldface** font | Boldface font is used to indicate commands, such as user entries, keys, buttons, folder names, and submenu names.<br>For example:<br>• Choose **Edit** > **Find**.<br>• Click **Finish**. |
| *italic* font | Italic font is used to indicate the following:<br>• To introduce a new term. Example: A *skill group* is a collection of agents who share similar skills.<br>• A syntax value that the user must replace. Example: IF (*condition, true-value, false-value*)<br>• A book title. Example: See the . |
| `window font` | Window font, such as Courier, is used for the following:<br>• Text as it appears in code or that the window displays. Example:<br>`<html><title>Cisco Systems, Inc. </title></html>` |
| < > | Angle brackets are used to indicate the following:<br>• For arguments where the context does not allow italic, such as ASCII output.<br>• A character string that the user enters but that does not appear on the window such as a password. |



CHAPTER 1

# Introduction



# Introduction

This document describes the Cisco Unified Contact Center Enterprise (UCCE) Application Gateway interface. The application gateway is a mechanism for interfacing the UCCE Call Router to an arbitrary customer application. The call router, under control of the script editor, will be able to make queries of a host, and base subsequent routing decisions on the results obtained. This could include:

- Controlling where and how the call is routed
- Passing arbitrary data to the site receiving the call (via translation routing)

The Enterprise Application Gateway interface is a TCP/IP socket based protocol. There is no SDK for this interface and therefore a developer is required to write directly to the socket layer.

### Benefits

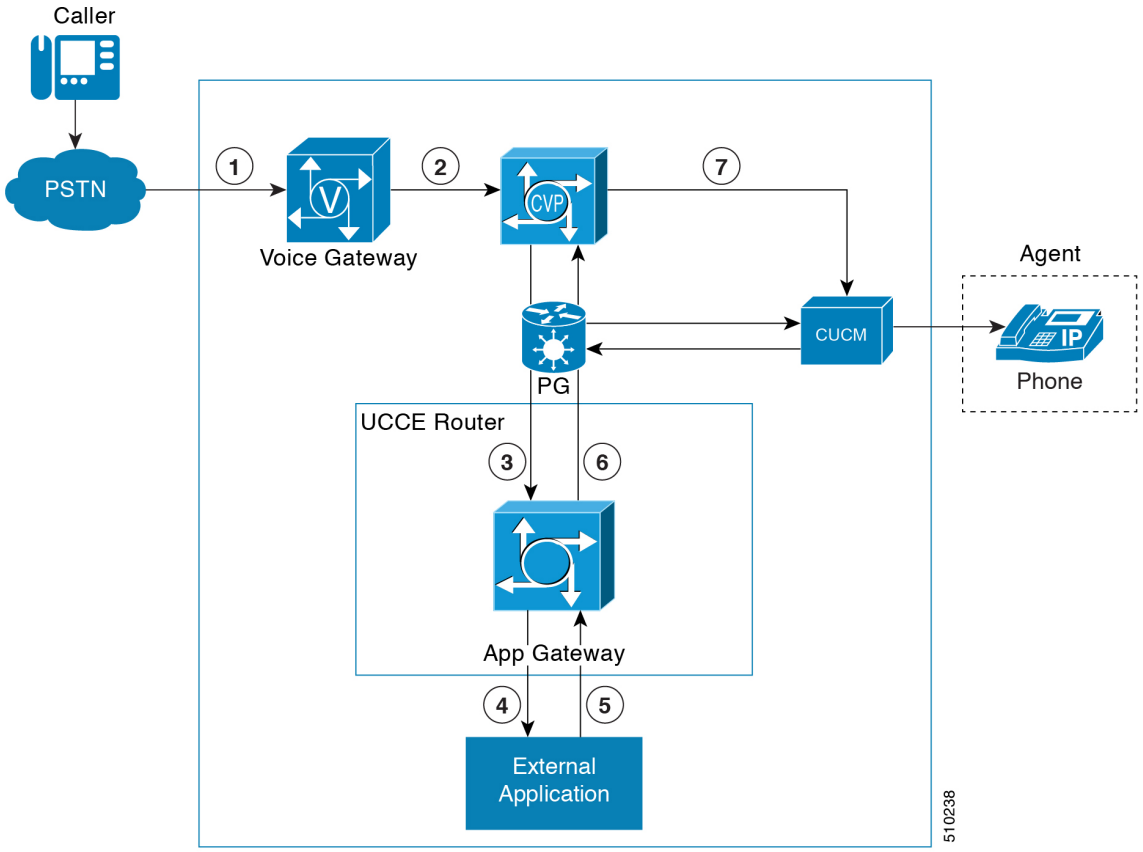
The benefits of using the Enterprise Applicaiton Gateway protocol are:

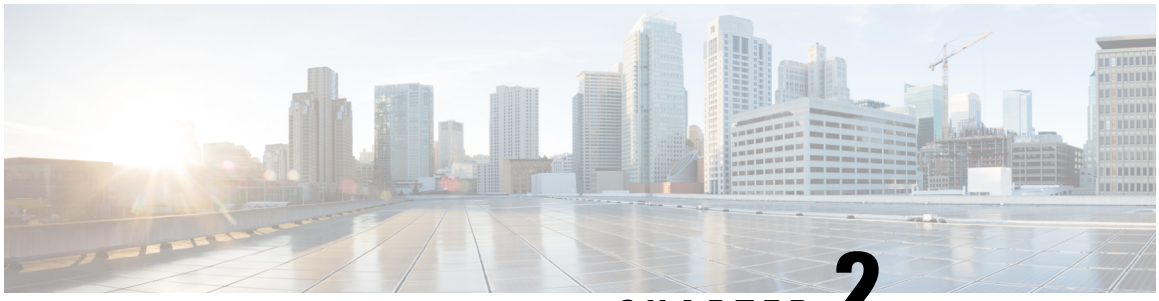
- Control where and how the call is routed
- Pass arbitrary data to the site receiving the call (via translation routing) when it is not possible through CTI
- Retrieve additional call (caller) context data to determine further call processing
- Execute transactions on a back-end system, for example during a script controlled IVR self-service application

### Enterprise Application Gateway Call Flow

The following diagram illustrates the basic Enterprise Application Gateway Contact Sharing call flow.

Introduction





CHAPTER 2

# Connections



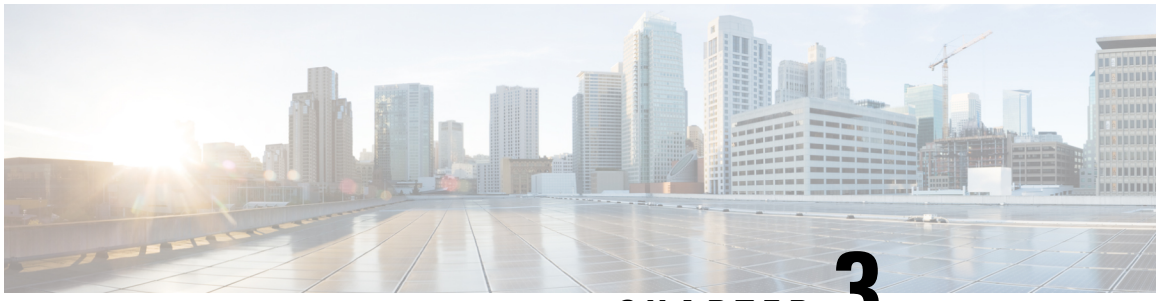
# Connections

Connections are initiated by the router. The customer must have an application listening to a socket on the target machine, and configure the name and port number of the application to connect it to the ICM database. Once the connection is made, the host will receive QUERY_REQ requests whenever requested by a script, and periodic heartbeat requests to verify that the connection is still alive.

Disconnects can be initiated from either the router or the host machine. If the host machine initiates a disconnect, the router will immediately attempt to reconnect. That request, however, can be refused.

Requests from the router are issued asynchronously. This means that the router will not wait for a response to a request before issuing the next one. No limit is placed on the number of requests that can be in the pipeline. The host is not required to answer the requests in the same order that they were issued.

Connections



CHAPTER 3

# Fault Tolerance Options



## Fault Tolerance Options

To achieve fault tolerance, multiple hosts can be used, or (less desirable) multiple connections to a single host. Several fault tolerance methods could be used, although only the duplicate request method will be implemented in the first release.

### Duplicate Request Method

Each router will manage a connection to a different host. Each time a script initiates a request, both routers will ask their corresponding host. Both routers will believe the response from whichever host responds first.

This method is the most reliable, but has the added expense of requiring two hosts to interface to. Even if a host (or a connection) fails, all requests will be satisfied.

### Alternating Request Method

Each router will manage a connection to a different host. The routers will take turns, sending half the requests to the host connected to side A, and the other half to the host connected to side B. If either host fails, the entire load will be directed to the surviving host.

When a host (or connection) fails, some requests may be lost. This is because by the time the router can figure out that a host is not going to respond, it is too late to ask the other host and still route the call within the deadline imposed by the network.

### Hot Standby Method

Each router will manage a connection to a different host. All requests will be directed to the designated primary host. If the host (or connection) fails, all requests will be directed to the backup host.

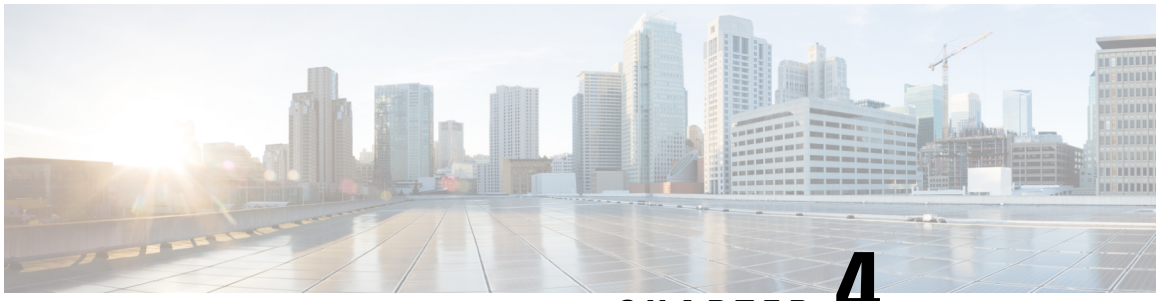
This option may also lose some requests on failures.

# Single Host Methods

Any of the above methods can also be configured to access a single host. This method provides no protection against host failures, but will protect against connection failures.

Nothing special needs be done to configure this method, other than to configure that same host for side A and side B (see database changes, below).

Note that in the single host variant of the duplicate request method, the host will have to be capable of handling two requests for each actual request from a script.



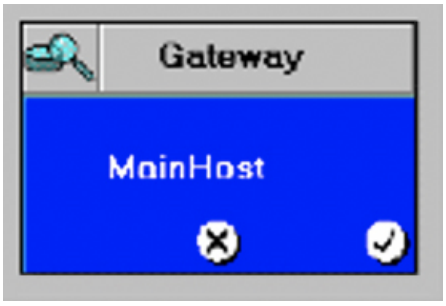
CHAPTER 4

# Script Editor Interface



## Application Gateway Script Node

Application gateway requests are initiated by call routing scripts. A script node to perform the operation looks like this:




The gateway node, like other scripting nodes, has a success and a failure exit. The success path indicates that the gateway responded normally, while the failure path indicates that some error occurred during the access.

## Application Gateway Status Variables

Each gateway has some associated variables that can also help in the scripting. Variables are named with the normal convention of Gateway.Name.Variable.

| Variable Name | Meaning |
|---|---|
| Available | If 1, at least one connection is available to the gateway. If 0, no connection is available. |

| Variable Name | Meaning |
|---|---|
| Status | The status of the previous request to the gateway. It can be one of the following: 0=success, 1=unavailable, 2=rejected, 3=timeout. |

Using the "MainHost" example above, a script could test "Gateway.MainHost.Available" to determine if using the gateway is meaningful. After the request, "Gateway.MainHost.Status" could be used if it is necessary to do processing based on the kind of failure.

# Application Gateway Request Variables

The actual requests to the gateway use existing call variables and user defined Expanded Call Context Variables. The following variables can be used. Those marked as "sent" are sent with the request. Those marked as "received" are returned from the host, and can be used subsequently in the script.

| Variable | Sent | Received | Notes |
|---|---|---|---|
| Subtype | yes | no | Optional string that can be set in the application gateway script node. |
| Call.CallerEnteredDigits | yes | no | |
| Call.CallingLineID | yes | no | ANI |
| Call.CustomerProvidedDigits | yes | yes | Only sent back to routing clients that support this (currently none?) |
| Call.RoutingClient | yes | no | Name of the routing client making the call |
| Call.DialedNumberString | yes | no | The original dialed number string passed by the routing client making the call. |
| Call.PeripheralVariable1 | yes | yes | |
| Call.PeripheralVariable2 | yes | yes | |
| Call.PeripheralVariable3 | yes | yes | |
| Call.PeripheralVariable4 | yes | yes | |
| Call.PeripheralVariable5 | yes | yes | |
| Call.PeripheralVariable6 | yes | yes | |
| Call.PeripheralVariable7 | yes | yes | |
| Call.PeripheralVariable8 | yes | yes | |
| Call.PeripheralVariable9 | yes | yes | |
| Call.PeripheralVariable10 | yes | yes | |
| Call.user.xxxxx | yes | yes | Each Expanded Call Variables that has been referenced by this call is sent. NOTE: The host may reply with any valid Expanded Call Variable, whether the variable has been referenced yet or not. |

Each request node can be configured to control exactly which variables can be sent and received. The default is that all variables are sent, and all variables that are returned by the host will be set when the request completes. If the node is configured to only accept certain variables, then any other extra variables returned by the host are ignored.

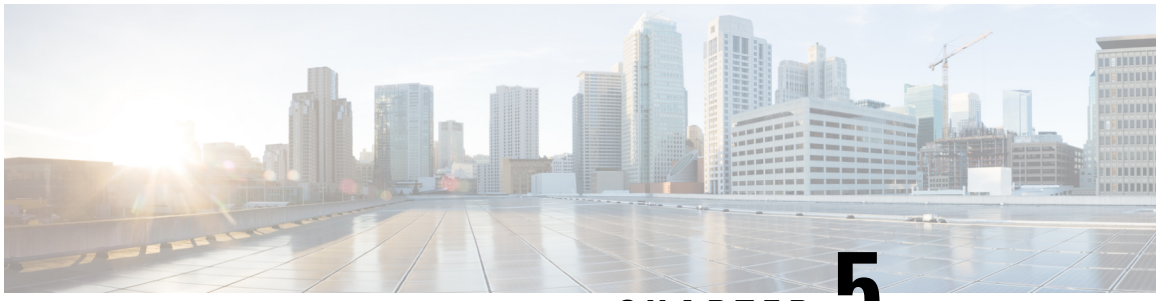
The script can also specify that no reply is required from the host. In this case, the message will be sent to the host, and the script will immediately continue processing without waiting. The processing of the script will fail only if the host is unavailable.

# Label Node Change

Although not absolutely required for the application gateway interface, an additional enhancement has been added to the "Label" node in the scripting language. This change allows the script to use an arbitrary expression as a label, rather than just being able to choose a single label from a list.

This would allow, for example, an application gateway to return a label to the router in one of the peripheral variables. This value could be then used in a subsequent label node to direct the routing of the call. This gives the gateway complete control of the routing, if that is desired.

If the expression does not yield a valid, configured label, the call will be default routed. A new function, ValidLabel, is available in expressions if it is necessary to perform this test in a script.



CHAPTER 5

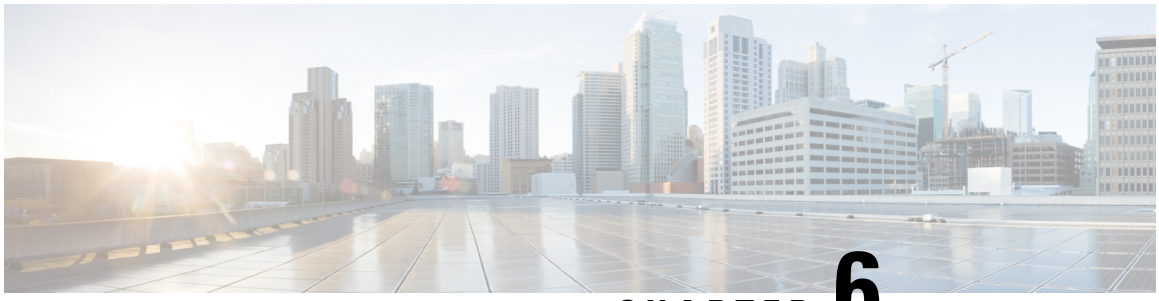
# Database Changes



## Database Changes

For information on database changes and the timeout and registry configuration, refer to Database Schema Handbook for Cisco Unified ICM/Contact Center Enterprise.

Database Changes



CHAPTER 6

# Message Protocol



# Message Protocol

The router connects to the host using TCP/IP, based on the IP address and port number configured in the ICM database. The message formats and protocols described below are similar to those described in the ICM/VRU interface specification.

All message will be sent using network byte ordering.

## Message Header

All messages will be preceded with an 8-byte header, in the following format:

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageLength | Length of the message, excluding the size of this header (8 bytes). | UINT | 4 |
| MessageType | Type of the message | UINT | 4 |

## Message Types

Each type of message is uniquely identified with a type on the message header. The following types are defined.

| Type | Name | Purpose |
|---|---|---|
| 1 | FAILURE_CONF | Sent by the host to report an error. Used in response to any message. |
| 2 | FAILURE_EVENT | Sent by the host at any time to report a failure. Not in response to a specific message. |
| 3 | OPEN_REQ | Sent by the router to initiate a connection. |

| Type | Name | Purpose |
|---|---|---|
| 4 | OPEN_CONF | Sent by the host to confirm initiation of a connection. |
| 5 | CLOSE_REQ | Sent by the router to close a connection. |
| 6 | CLOSE_CONF | Sent by the host to confirm a close. |
| 7 | HEARTBEAT_REQ | Sent periodically by the router to keep the connection alive. |
| 8 | HEARTBEAT_RESP | Sent by the host to confirm the heartbeat. |
| 9 | QUERY_REQ | Sent by the router when a script requests data from the host. |
| 10 | QUERY_RESP | Sent by the host in response to a query. |
| 11 | PARAM_REQ | Sent by the router to change a parameter. |
| 12 | PARAM_CONF | Sent by the host to confirm the parameter change. |

# Tagged fields

Many of the messages contain optional and variable-length character fields. All such fields are encoded using the tagged field format. Tagged fields always occur at the end of the message. To determine if any tagged fields are present, a calculation must be done by subtracting the length of the fixed part of the message from the total message length in the header. If this is greater than zero, tagged fields appear. This calculation must be repeated as each tagged field is processed to see if more tagged fields remain.

| Subfield | Value | Data Type | Bytes |
|---|---|---|---|
| Tag | Encoded type of the field. See the following table. | UCHAR | 1 |
| FieldLength | Number of bytes of data (*n*) which follow. | UCHAR | 1 |
| Data | The data. | CHAR | *n* |

# Tagged Field Codes

The following table shows the possible value for the Tag field, and the messages in which they are used.

| Value | Name | Messages |
|---|---|---|
| 1 | ERROR_TAG | FAILURE_CONF, FAILURE_EVENT |
| 2 | CONFIG_TAG | OPEN_REQ |
| 3 | SUBTYPE_TAG | QUERY_REQ, CONTACT_SHARE_REQ |

| Value | Name | Messages |
|---|---|---|
| 4 | CED_TAG | QUERY_REQ, CONTACT_SHARE_REQ |
| 5 | CLID_TAG | QUERY_REQ, CONTACT_SHARE_REQ |
| 6 | CDPD_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 7 | ROUTING_CLIENT_TAG | QUERY_REQ, CONTACT_SHARE_REQ |
| 8 | DN_STRING_TAG | QUERY_REQ, CONTACT_SHARE_REQ |
| 9 | VAR1_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 10 | VAR2_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 11 | VAR3_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 12 | VAR4_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 13 | VAR5_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 14 | VAR6_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 15 | VAR7_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 16 | VAR8_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 17 | VAR9_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 18 | VAR10_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 19 | SESSION_KEY_TAG | OPEN_REQ |
| 20 | EXPANDED_CALL_VARIABLE_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 21 | EXPANDED_CALL_ARRAY_TAG | QUERY_REQ, QUERY_RESP, CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 22 | CONTACT_SHARE_TAG | CONTACT_SHARE_REQ, CONTACT_SHARE_RESP |
| 23 | PAIRING_TAG | CONTACT_SHARE_SERVICE_CONFIGURATION |
| 24 | HOSTNAME_TAG | CONTACT_SHARE_SERVICE_CONFIGURATION |
| 25 | ADDRESS_TAG | CONTACT_SHARE_SERVICE_CONFIGURATION |
| 26 | USERNAME_TAG | CONTACT_SHARE_SERVICE_CONFIGURATION |
| 27 | PASSWORD_TAG | CONTACT_SHARE_SERVICE_CONFIGURATION |

| Value | Name | Messages |
|---|---|---|
| 28 | ERROR_MESSAGE_TAG | CONTACT_SHARE_EMS_REPORT_EVENT |

# Expanded Call Variable Tags

Expanded Call Variable tags contain multiple packed elements in the data segment of the tag. The following tables define the data segment format for the two Expanded Call Variable tags.

## EXPANDED_CALL_VARIABLE_TAG Data

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| Name | The null-terminated name of the call variable | UCHAR[] | 1-31 |
| Value | The null-terminated value of the call variable | UCHAR[] | 1-220 |

## EXPANDED_CALL_ARRAY_TAG Data

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| Index | Index value of the array value to be set or read | UCHAR | 1 |
| Name | The null-terminated name of the call variable | UCHAR[] | 1-31 |
| Value | The null-terminated value of the call variable | UCHAR[] | 1-220 |

# InvokeID Fields

Most messages include an InvokeID field. This is simply a sequence number which is changed by the router for each message sent. InvokeID must be returned in the response messages. This mechanism allows messages to be accurately matched up with their responses.

# Failure Messages

The host can report failures with two different messages. FAILURE_CONF is used by the host in response to a specific message from the router. FAILURE_EVENT is used by the host to report an error at any time.

If the router receives either of these messages, it will close the connection, initiate the appropriate fault tolerant strategy, and attempt to restart the connection.

Both messages allow for the host to pass an error description which will be logged.

## FAILURE_CONF Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 1 | Header | 8 |
| InvokeID | InvokeID of the message that provoked this response | UINT | 4 |
| Status | Status code indicating the cause of the failure | UINT | 4 |

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| ErrorText | Optional text string describing the reason (ERROR_TAG) | TAGGED | *n* |

## FAILURE_EVENT Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 2 | Header | 8 |
| Status | Status code indicating the cause of the failure | UINT | 4 |
| ErrorText | Optional text string describing the reason (ERROR_TAG) | TAGGED | *n* |

## Status Codes

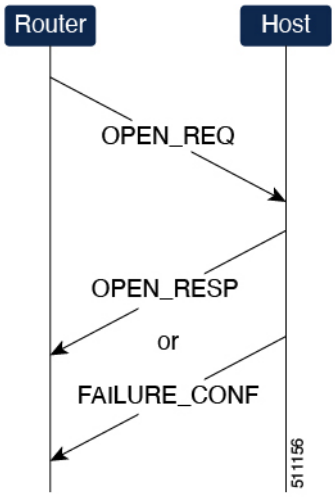
The host may use the following codes to report an error back to the router. For all errors other than E_AG_INVALID_VERSION, the router will log the error, close the connection (if open), and attempt to reconnect.

If E_AG_INVALID_VERSION is received in response to a connect, the router may, in the future, attempt to reconnect with a lower version.

| Value | Name | Meaning |
|---|---|---|
| 0 | E_AG_NO_ERROR | No error occured. |
| 1 | E_AG_INVALID_VERSION | The host does not support the requested version. The router will try to connect with an older version. |
| 2 | E_AG_SESSION_ACTIVE | The host has rejected the connection because one is already active. |
| 3 | E_AG_HOST_OFFLINE | The host cannot accept a connection at this time. |
| 4 | E_AG_INVALID_MESSAGE | The host didn’t recognize the message because of some formatting error. |
| 5 | E_AG_SESSION_INACTIVE | The host rejected a request because it could not connect to the router. |
| 6 | E_AG_HOST_ERROR1 | Available for host to report some application-specific error. The router will log this error. |
| 7 | E_AG_HOST_ERROR2 | See E_AG_HOST_ERROR1. |
| 8 | E_AG_INVALID_KEY | The host could not decrypt the session key. |

# Initiating Connections

The OPEN_REQ and OPEN_RESP messages are used by the router to initiate connections.




## OPEN_REQ Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 3 | Header | 8 |
| InvokeID | An ID for this message, which should be included in the response. | UINT | 4 |
| VersionNumber | Version number of the interface. Currently 2 | UINT | 4 |
| IdleTimeout | Session idle timer value, in milliseconds. If the host does not hear from the router in this time, it should close the connection. This value is set to 4 times the heartbeat interval. | UINT | 4 |
| Encryption | Indicates the kind of encryption being done. 0 = none, 1 = private key known to both ends of the connection. | UINT | 4 |
| ConnectString | This is an optional connection string, configured in the ICM database. It is sent only if it specified in the ICM Application_Gateway configuration (CONFIG_TAG). | TAGGED | *n* |
| SessionKey | DES encryption key to use for this session. Only sent if encryption is enabled. | TAGGED | *n* |

## OPEN_RESP Message

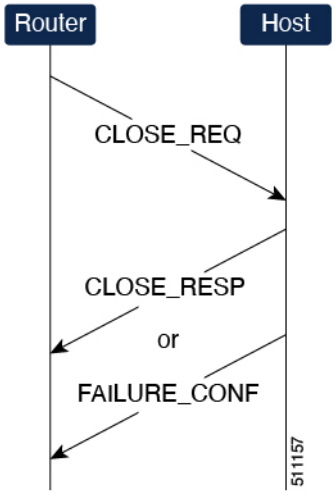
| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 4 | Header | 8 |
| InvokeID | The InvokeID from the OPEN_REQ message. | UINT | 4 |

If the OPEN_REQ message is rejected, the FAILURE_CONF message should be used. If the router receives the FAILURE_CONF message, it will close the connection, wait for a configurable delay, and retry.

If the router receives no response within 15 seconds (configurable), it will close the connection and retry.

# Closing Connections

The CLOSE_REQ and CLOSE_RESP messages are used by the router to terminate connections.




## CLOSE_REQ Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 5 | Header | 8 |
| InvokeID | An ID for this message, which should be included in the response. | UINT | 4 |

## CLOSE_RESP Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 6 | Header | 8 |
| InvokeID | The InvokeID from the CLOSE_REQ message. | UINT | 4 |

If the close command is to be rejected (whatever that may mean), the FAILURE_RESP message may be used to indicate that. The connection will still be considered closed by the router, and disconnected.

If the router receives no response in 15 seconds, the connection will be closed anyway.

The router will attempt an orderly close of the host connection whenever it goes out of service, but this can never be guaranteed. The host may simply see the TCP/IP connection fail.

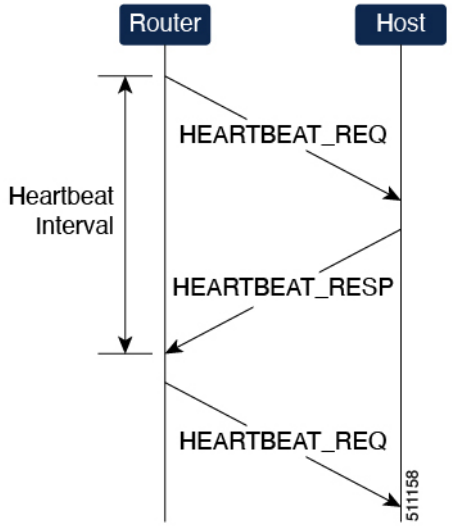
The router will also close and reestablish the connection when certain configuration changes occur, such as changing the IP address.

# Heartbeats

The heartbeat mechanism is used to periodically check the state of the host server. They are sent, by default, every 15 seconds. Heartbeats are only sent when the link has been idle for 15 seconds. If other requests have been sent and acknowledged, heartbeats won't be sent.

If the router doesn't see the reply in time, it will retry heartbeats (at a smaller interval) until either they succeed, or the connection is declared dead.

The router will treat a query timeout also as a failed heartbeat and begin the heartbeat retry operation.




## HEARTBEAT_REQ Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 7 | Header | 8 |
| InvokeID | An ID for this message, which should be included in the response. | UINT | 4 |

## HEARTBEAT_RESP Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 8 | Header | 8 |
| InvokeID | The InvokeID from the HEARTBEAT_REQ message. | UINT | 4 |

# Queries

The router sends a QUERY_REQ message each time it encounters a script that requests a host interaction. This could, it theory, happen multiple times per call, provided that the aggregate time to complete these requests does not exceed the time limits imposed on the router by the routing clients.

There may be multiple queries outstanding at any given time, as the router will continue to process calls while waiting for a response from an earlier one. The host does not have to answer the queries in order, as long as all are answered within the time limit.

# QUERY_REQ Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 9 | Header | 8 |
| InvokeID | An ID for this message, which should be included in the response. | UINT | 4 |
| RouterCallDay | The RouterCallDay and RouterCallKey fields together form a 64 bit unique ID to identify any Cisco ICM call. A host might save this value to relate its processing back to a Route_Call_Detail record in the database. | UINT | 4 |
| RouterCallKey | See RouterCallDay | UINT | 4 |
| Flags | Various flags:<br>0x01: Duplicate Request<br>0x02: NoScriptReply | UBYTE | 1 |
| Various | Any of the tagged fields valid for QUERY_REQ. May vary from request to request, based on script requirements. They may appear in any order. | TAGGED | *n* |

### Duplicate Request Flag

The setting of the "Duplicate" flag depends on the fault tolerance method chosen. If the alternate request or the hot standby option is chosen, the Duplicate field will always be set to false. If the duplicate request method is chosen, the flag will be set depending on the state of the connections. If both the side A and side B connections are OK, then requests will be sent with Duplicate set to true on one side, and set to false on the other. If one side fails, all requests will have duplicate set to false on the remaining good side. When the failed side returns to service, requests will be again marked as duplicate on one of the sides.

This flag is useful for an application that may be retaining some state on the requests. An example might be a password validation, where it is necessary to invalidate the password after a certain number of failures. The application could be written to only count the errors for requests where Duplicate is false. In that way, a user would not be charged for two errors for one failure.

### No Script Reply Flag

This flag indicates that the Cisco ICM script is not waiting for a reply. This flag is informational only; the host is still required to send a QUERY_RESP for this message, in order to maintain the communications protocol. It is not necessary to send any tagged fields in this message, as they will not be passed back to any script.

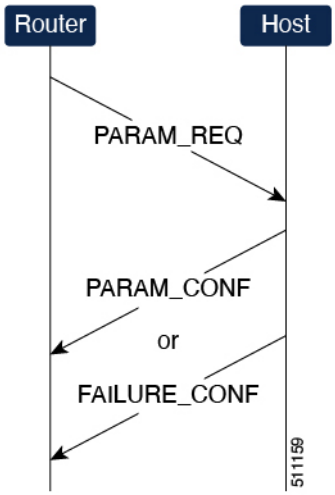
# QUERY_RESP Message

| Field Names | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 10 | Header | 8 |
| InvokeID | The InvokeID from the QUERY_REQ | UINT | 4 |

| Field Names | Value | Data Type | Bytes |
|---|---|---|---|
| Reject | A flag indicating that the host rejected the query, for whatever reason. The script will take the failure path from the Application Node, with the status code indicating a reject. Any tagged variables passed back with the reject will be set. | UINT | 4 |
| Various | Any of the tagged fields valid for QUERY_RESP. They may appear in any order. | TAGGED | *n* |

# Parameter Changes

If there is a change in some configured parameter, the router will inform the host immediately of such a change.

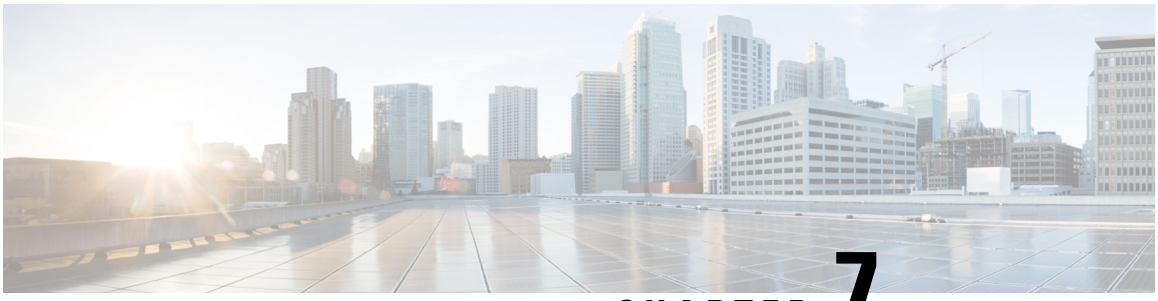



# PARAM_REQ Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 11 | Header | 8 |
| InvokeID | An ID for this message, which should be included in the response. | UINT | 4 |
| ParamID | An ID of the parameter being changed. Only one ParamID is defined:<br>1 = IdleTimeout, set originally in the OPEN_REQ | UINT | 4 |
| ParamValue | Value for the parameter | UINT | 4 |

# PARAM_CONF Message

| Field Name | Value | Data Type | Bytes |
|---|---|---|---|
| MessageHeader | Standard header. MessageType = 12 | Header | 8 |
| InvokeID | The InvokeID from the HEARTBEAT_REQ message. | UINT | 4 |



CHAPTER 7

# Encryption



# Encryption

Any host may be configured to use the encrypted messages. When this is selected, a private key must be selected and configured in the router, and at the host. When the connection is made, the router will generate a random, 8 byte session key, encrypt the session key using DES and the private key, and send the encrypted session key. To allow for checking, the session key is sent doubled, as a 16 byte string. For example, if the key was ABCDEFGH, ABCDEFGHABCDEFGH would be sent as a 16 byte encrypted string to the host. The host should decrypt this using the session key, verify that the first 8 bytes are the same as the last 8 bytes. If this check fails, the connection should be rejected.

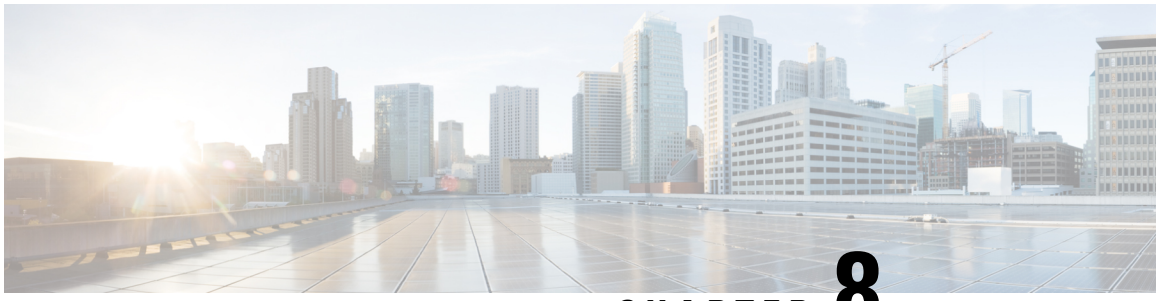
All tagged fields in subsequent QUERY_REQ and QUERY_RESP messages will be encrypted and decrypted using this 8 byte session key. Each tagged field will be padded with nulls until it is a multiple of 8 bytes long, and encrypted. The tagged field sent will always be a multiple of 8 bytes. After decryption, the trailing nulls must be counted and removed in order to compute the true length of the string, if necessary.

Any tagged field that does not decrypt to a valid ASCII string will be treated as an error, and ignored.

### Transport Layer Security (TLS) Security

The application gateway interface supports TLS based security mechanism. For more information, refer to Security Guide for Cisco Unified ICM/Contact Center Enterprise.

Encryption



CHAPTER 8

# Internal Implementation Details



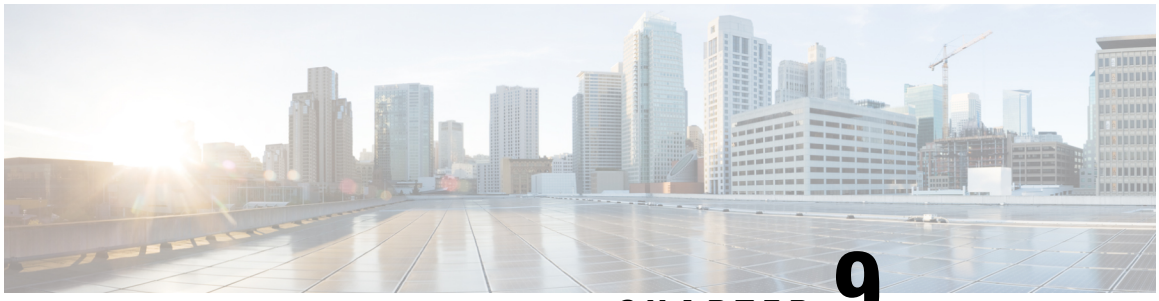
## Internal Implementation Details

Although this specification often references interactions between the router and the host machine, the application gateway interface is not implemented directly in the router process. The diagram below shows the actual connections.

The actual connections to the hosts are managed by an application gateway interface process. One of these processes runs on each side of the central controller. When they send a reply back to the router, it is via the MDS mechanism, which allows both routers to see the reply, even though only one of the gateway processes may have sent it.

In the case of duplicate request fault tolerance, the router simply discards the second reply that it sees. All timeouts are managed by the gateway process.

Internal Implementation Details



CHAPTER 9

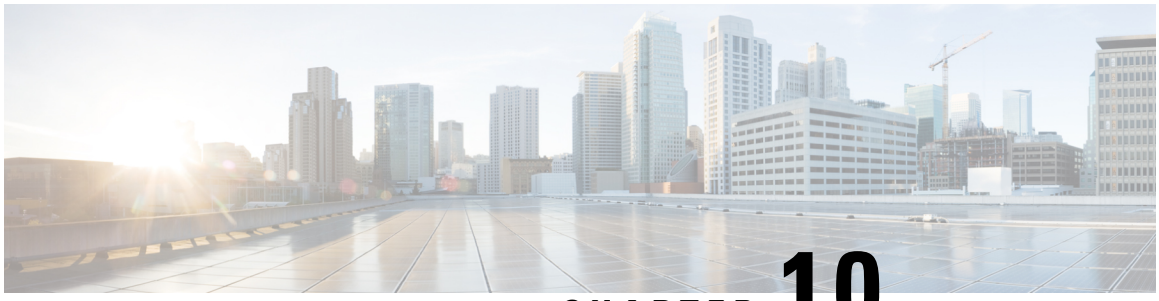
# Administrative Control



## Administrative Control

In addition to the normal configuration control from the AW, an administrative interface will also be provided which will allow an operator to disable a connection, reestablish it, restart after failed connections, and so on.

Administrative Control



CHAPTER 10

# Alarms



## Alarms

Alarms will be reported using existing EMS mechanisms. Alarms can be logged either by the router or the application gateway interface process.

Alarms