# Cisco Gatekeeper External Interface Reference, Version 3.1

Cisco IOS Releases 12.2(8)T
February 2002

Corporate 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: 408 526-4100

*Cisco Gatekeeper External Interface Reference, Version 3.1*


Cisco Gatekeeper External Interface Reference, Version 3.1

# About This Guide

This section describes the objectives, audience, organization, and conventions of the *Cisco Gatekeeper External Interface Reference, Version 3.1*.

## Document History

The versions of this guide are listed in Table 1.

***Table 1 Document History***

| Title | Software Version | Location |
|---|---|---|
| *Cisco Gatekeeper External Interface Reference, Version 1* | 12.2(1)T | http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121rel/gktmp/index.htm |
| *Cisco Gatekeeper External Interface Reference, Version 2* | 12.1(5)XM | http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121rel/gktmpv2/index.htm |
| *Cisco Gatekeeper External Interface Reference, Version 3* | 12.2(2)XA | http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/rel_docs/gktmpv3/index.htm |
| *Cisco Gatekeeper External Interface Reference, Version 3.1* | 12.2(2)XB<br>12.2(4)T<br>12.2(8)T | http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/rel_docs/gktmpv31/index.htm |

## Document Objectives

This guide is designed to help you understand and implement an external interface to the Cisco IOS Gatekeeper using the Cisco Gatekeeper Transaction Message Protocol (GKTMP) and application programming interface (API).

## Audience

This guide is intended for application programmers who want to develop an application that interfaces with the Cisco IOS Gatekeeper.

# Document Organization

This document is divided into the following chapters shown in Table 2:

***Table 2 Organization***

| Chapter | Title | Description |
|---|---|---|
| Chapter 1 | Overview of H.323 | Provides a high-level overview of H.323. |
| Chapter 2 | Overview of the Cisco IOS Gatekeeper | Provides an overview of the features and functions of the Cisco IOS Gatekeeper. |
| Chapter 3 | Implementing an External Interface to the Cisco IOS Gatekeeper | Provides information about and examples of implementing an external interface to the Cisco IOS Gatekeeper using the GKTMP and the Cisco IOS Gatekeeper API. |
| Chapter 4 | GKTMP Messages | Describes the messages used with the GKTMP. |
| Chapter 5 | Gatekeeper API Functions and Structures | Describes the functions provided with the Cisco IOS Gatekeeper API. |
| Chapter 6 | GKTMP Command Reference | Describes the Cisco IOS software commands used to configure triggers for a Cisco IOS Gatekeeper. |

# Related Documentation

For additional information about the commands used in conjunction with the GKTMP, see the following documents:

*Cisco IOS Voice, Video, and Fax Configuration Guide*, Release 12.2

*Cisco IOS Voice, Video, and Fax Command Reference*, Release 12.2

*Cisco High Performance Gatekeeper*

*Cisco H.323 Scalability and Interoperability Enhancements*

# Command Syntax Conventions

Table 3 describes the syntax used with the commands in this document.

***Table 3 Command Syntax Guide***

| Convention | Description |
|---|---|
| **boldface** | Commands and keywords. |
| *italic* | Command input that is supplied by you. |
| [ ] | Keywords or arguments that appear within square brackets are optional. |
| { x \| x \| x } | A choice of keywords (represented by x) appears in braces separated by vertical bars. You must select one. |
| ^ or Ctrl | Represent the key labeled *Control*. For example, when you read *^D* or *Ctrl-D*, you should hold down the Control key while you press the D key. |

*Table 3 Command Syntax Guide*

| Convention | Description |
|---|---|
| `screen font` | Examples of information displayed on the screen. |
| **boldface screen font** | Examples of information that you must enter. |
| < > | Nonprinting characters, such as passwords, appear in angled brackets. |
| [ ] | Default responses to system prompts appear in square brackets. |

# Obtaining Documentation

The following sections provide sources for obtaining documentation from Cisco Systems.

## World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following sites:

- http://www.cisco.com
- http://www-china.cisco.com
- http://www-europe.cisco.com

## Documentation CD-ROM

Cisco documentation and additional literature are available in a CD-ROM package, which ships with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or as an annual subscription.

## Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco Product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, in North America, by calling 800 553-NETS(6387).

## Documentation Feedback

If you are reading Cisco product documentation on the World Wide Web, you can submit technical comments electronically. Click **Feedback** in the toolbar and select **Documentation**. After you complete the form, click **Submit** to send it to Cisco.

You can e-mail your comments to bug-doc@cisco.com.

To submit your comments by mail, use the response card behind the front cover of your document, or write to the following address:

Attn Document Resource Connection
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

# Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain documentation, troubleshooting tips, and sample configurations from online tools. For Cisco.com registered users, additional troubleshooting tools are available from the TAC website.

**Note** If you are an application programmer and need more information or technical assistance with GKTMP/API developer support, see Developer Support Central on CCO at http://www.cisco.com/warp/public/570/ or contact gktmp-sdp@cisco.com.

## Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information and resources at anytime, from anywhere in the world. This highly integrated Internet application is a powerful, easy-to-use tool for doing business with Cisco.

Cisco.com provides a broad range of features and services to help customers and partners streamline business processes and improve productivity. Through Cisco.com, you can find information about Cisco and our networking solutions, services, and programs. In addition, you can resolve technical issues with online technical support, download and test software packages, and order Cisco learning materials and merchandise. Valuable online skill assessment, training, and certification programs are also available.

Customers and partners can self-register on Cisco.com to obtain additional personalized information and services. Registered users can order products, check on the status of an order, access technical support, and view benefits specific to their relationships with Cisco.

To access Cisco.com, go to the following website:

http://www.cisco.com

# Technical Assistance Center

The Cisco TAC website is available to all customers who need technical assistance with a Cisco product or technology that is under warranty or covered by a maintenance contract.

## Contacting TAC by Using the Cisco TAC Website

If you have a priority level 3 (P3) or priority level 4 (P4) problem, contact TAC by going to the TAC website:

http://www.cisco.com/tac

P3 and P4 level problems are defined as follows:

- P3—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- P4—You need information or assistance on Cisco product capabilities, product installation, or basic product configuration.

In each of the above cases, use the Cisco TAC website to quickly find answers to your questions.

To register for Cisco.com, go to the following website:

http://www.cisco.com/register/

If you cannot resolve your technical issue by using the TAC online resources, Cisco.com registered users can open a case online by using the TAC Case Open tool at the following website:

http://www.cisco.com/tac/caseopen

## Contacting TAC by Telephone

If you have a priority level 1 (P1) or priority level 2 (P2) problem, contact TAC by telephone and immediately open a case. To obtain a directory of toll-free numbers for your country, go to the following website:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

P1 and P2 level problems are defined as follows:

- P1—Your production network is down, causing a critical impact to business operations if service is not restored quickly. No workaround is available.
- P2—Your production network is severely degraded, affecting significant aspects of your business operations. No workaround is available.

CHAPTER 1

# Overview of H.323

This chapter provides an overview of H.323 and includes the following sections:

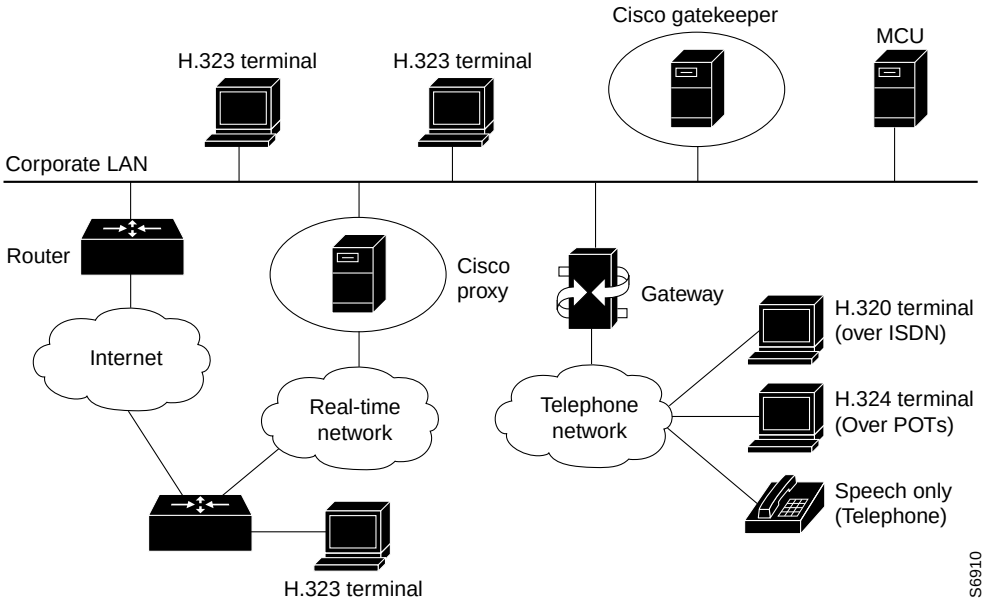


H.323 is an ITU standard for transmitting audio, video, and data conferencing data on an IP-based internetwork. The H.323 standard provides for the following types of *endpoints* in the network:

- H.323 Terminals
- Gatekeepers
- MCUs
- Gateways

Figure 1-1 shows a typical H.323 network:

*Figure 1-1 H.323 Network*




# H.323 Terminals

An H.323 terminal is an endpoint in the LAN that participates in real-time, two-way communications with another H.323 terminal, gateway, or multipoint control unit (MCU). A terminal must support audio communication and can also support audio with video, audio with data, or a combination of all three.

H.323 terminals must support the following standards and protocols:

- H.245—An ITU standard used by the terminal to negotiate its use usage of the channel. The H.245 control channel provides in-band reliable transport for capabilities exchange, mode preference from the receiving end, logical channel signaling, and control and indication. Part of the capabilities exchange includes specifying which coder-decoders (CODECs) are available. Recommended audio CODECs include G.711, G.722, G.723, G.723.1, G.728, and G.729. Recommended video CODECs include H.261 and H.263.
- H.225.0—An ITU standard that uses a variant of Q.931 to set up the connection between two H.323 endpoints.
- RAS—(Registration Admission Status) A protocol used to communicate with the H.323 gatekeeper.
- RTP and RTCP—(Real-Time Transport Protocol and Real-Time Control Protocol) Protocols used to sequence the audio and video packets. The RTP header contains a time stamp and sequence number, allowing the receiving device to buffer as much as necessary to remove jitter and latency by synchronizing the packets to play back a continuous stream of sound. RTCP controls RTP and gathers reliability information and periodically passes this information onto session participants.

# Gatekeepers

Gatekeepers are optional nodes that manage other nodes in an H.323 network. Other nodes communicate with the gatekeeper using the RAS protocol. A gatekeeper is not required in an H.323 network, but it must be used if one is present.

The H.323 nodes attempt to *register* with a gatekeeper on startup. When an H.323 node wants to communicate with another endpoint, it requests *admission* to the call, using a symbolic alias for the endpoint name such as an E.164 (ITU-T recommendation for international telecommunication numbering) address or an e-mail ID. If the gatekeeper decides the call can proceed, it returns a destination IP address to the originating H.323 node. This IP address can be the actual address of the target endpoint or it can be an intermediate address. Finally, a gatekeeper and its registered endpoints exchange *status* information.

## Gatekeeper Zones

H.323 endpoints are grouped together in zones. Each zone has one gatekeeper that manages all the endpoints in the zone. A zone is an administrative convenience similar to a DNS domain. Gatekeeper zones are normally set up to correspond to geographic zones.

# MCUs

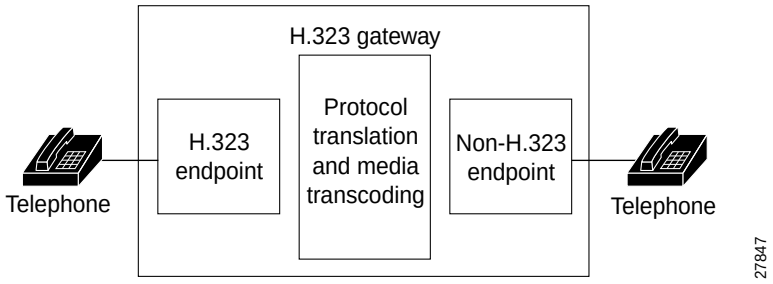
An MCU is an endpoint on the LAN that provides the capability for three or more terminals and gateways to participate in a multipoint conference. It controls and mixes video, audio, and data from terminals to create a robust video conference. An MCU can also connect two terminals in a point-to-point conference that can later develop into a multipoint conference.

**Note** Some terminals have limited multipoint-control built into them. These terminals might not require an MCU with all the functionality mentioned previously.

# Gateways

An H.323 gateway can provide an interface between H.323 and the Public Switched Telephone Network (PSTN), H.320 terminals, V.70 terminals, H.324 terminals, and other speech terminals. It provides standard interfaces to the PSTN, processes the voice and fax signals using CODECs to convert between circuit-switched and packet formats, and works with the gatekeeper through the RAS protocol to route calls through the network. Gateways provide translation between transmission formats, such as H.245 and H.242. Figure 1-2 shows a gateway between an H.323 terminal and a speech-only telephone.

*Figure 1-2 Gateway Between an H.323 Terminal and a Speech-only Telephone*



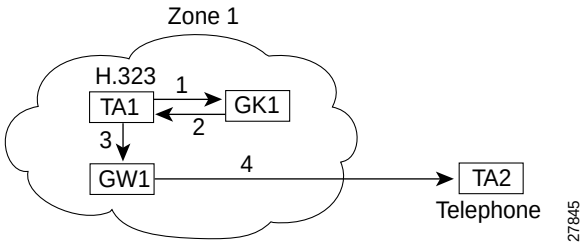

# How Terminals, Gatekeepers, and Gateways Work Together

Gateways provide protocol conversion between terminals running different types of protocols. Gateways communicate with gatekeepers using the RAS protocol. The gatekeeper maintains resource computing information, which it uses to select the appropriate gateway during the admission of a call. In Figure 1-3 and Figure 1-4:

- TA1 is an H.323 terminal registered to GK1.
- GW1 is an H.323-to-H.320 gateway registered to GK1.
- TA2 is a telephone.

Figure 1-3 illustrates the processing of a call that originates with a device in the zone (TA1) and is intended for a device outside the zone (TA2).

*Figure 1-3 Processing of Calls Going Out of the Zone*

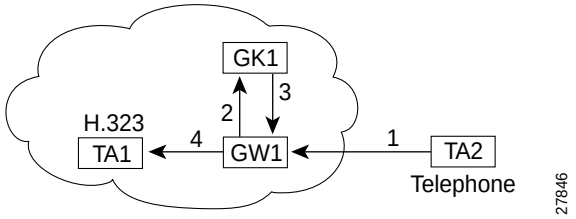



A call from TA1 to TA2 is set up as follows:

1. TA1 asks GK1 for permission to connect to TA2's E.164 address.
2. The gatekeeper looks through its local registrations and does not find any H.323 terminals registered with that E.164 address, so the gatekeeper assumes that it is a telephone outside the scope of H.323. The gatekeeper instructs TA1 to connect to the GW1 IP address.
3. TA1 connects to GW1.
4. GW1 completes the call to TA2.

Figure 1-4 illustrates the processing of a call that originates with a device outside the zone (TA2) and is intended for a device in the zone (TA1).

*Figure 1-4 Processing of Calls Coming Into the Zone*




A call from TA2 to TA1 is set up as follows:

1. TA2 calls GW1 and provides the TA1 E.164 address as the final destination.
2. GW1 sends a message to GK1 asking to connect to that address.
3. GK1 gives GW1 the address of TA1.
4. GW1 completes the call with TA1.

CHAPTER 2

# Overview of the Cisco IOS Gatekeeper

This chapter describes the main functions of a gatekeeper and includes the following sections:



Cisco offers a Voice over IP gatekeeper called the Multimedia Conference Manager, which is an H.323-compliant program implemented as part of the Cisco IOS software. The Multimedia Conference Manager software can run on Cisco 2500 series, Cisco 2600 series, Cisco 3600 series, and Cisco MC3810 Multiservice Access Concentrators.

## Zone and Subnet Configuration

A zone is defined as the set of H.323 nodes controlled by a single gatekeeper. Gatekeepers co-existing on a network can be configured so that they register endpoints from different subnets.

Endpoints attempt to discover a gatekeeper, and consequently what zone they are members of, using the RAS message protocol. The protocol supports a discovery message that can be sent multicast or unicast.

If the message is sent multicast, the endpoint registers nondeterministically with the first gatekeeper to respond. Any endpoint on a subnet that is not enabled for the gatekeeper is not accepted as a member of that gatekeeper's zone. If the gatekeeper receives a discovery message from such an endpoint, it sends an explicit reject message.

## Terminal Name Registration

Gatekeepers recognize one of the following types of terminal aliases, or terminal names:

- H.323 identifiers (IDs), which are arbitrary, case-sensitive text strings.
- E.164 addresses, which are telephone numbers.
- E-mail IDs.

If an H.323 network deploys inter-zone communication, each terminal should at least have a fully-qualified e-mail name as its H.323 ID. For example, *bob@cisco.com*. The domain name of the e-mail ID should be the same as the configured domain name for the gatekeeper of which it is a member. As in the previous example, the domain name is *cisco.com*.

# Inter-Zone Communication

To allow endpoints to communicate between zones, gatekeepers must be able to determine which zone an endpoint is in and locate the gatekeeper responsible for that zone. If DNS is available, you can associate a DNS domain name to each gatekeeper.

# Accounting Using RADIUS/TACACS+

If you enable AAA on the gatekeeper, the gatekeeper emits an accounting record each time an endpoint registers or unregisters, or each time a call is admitted or disconnected.

# Inter-Zone Routing

There are three types of address destinations used in H.323 calls. The destination can be specified using either an H.323-ID address (a character string), an E.164 address (a string containing telephone keypad characters), or an e-mail ID (a character string). The way inter-zone calls are routed by the Cisco IOS Gatekeeper depends on the type of address being used.

- When using H.323-ID addresses, inter-zone routing is handled through the use of domain names. For example, to resolve the domain name *bob@cisco.com*, the source endpoint's gatekeeper finds the gatekeeper for *cisco.com* and sends the location request for target address *bob@cisco.com* to that gatekeeper. The destination gatekeeper looks in its registration database, sees *bob* registered, and returns the appropriate IP address to get to *bob*.

  **Note** Although H.225 does not require the use of a domain name with H.323 IDs, the Cisco IOS Gatekeeper does require a domain name.

- When using E.164 addresses, call routing is handled through means of zone prefixes and gateway type prefixes, also referred to as technology prefixes. The zone prefixes, which are typically area codes, serve the same purpose as domain names in H.323-ID address routing. Unlike domain names, however, more than one zone prefix can be assigned to one gatekeeper, but the same prefix cannot be shared by more than one gatekeeper. With Cisco IOS Release 12.0(3)T and later, you can configure inter-zone routing using E.164 addresses.
- When using e-mail IDs, inter-zone routing is handled through the use of domain names—just as it is with H.323 IDs. Again, the source endpoint's gatekeeper finds the gatekeeper for the specified domain and sends the location request for the target address to that gatekeeper.

CHAPTER 3

# Implementing an External Interface to the Cisco IOS Gatekeeper

This chapter describes how to implement an external interface to the Cisco IOS Gatekeeper and contains the following sections:



Although the Cisco IOS Gatekeeper provides many functions, there might be the occasion when additional function is desired or needed. For example, an organization could require additional authentication functions, need to implement specific policy controls, or want to use Internet call waiting.

The Gatekeeper Transaction Message Protocol (GKTMP) and the gatekeeper application programming interface (API) were developed to allow communication between the Cisco IOS Gatekeeper and an external application.

GKTMP is based on RAS and provides a set of ASCII request/response messages that can be used to exchange information between the Cisco IOS Gatekeeper and the external application over a TCP connection, and through the use of the gatekeeper API.

The gatekeeper API is object code that contains the API functions, which are designed to work with GKTMP. An external application links with the object code and calls the functions as necessary.

Using the GKTMP and the gatekeeper API, organizations can supplement the functions of the Cisco IOS Gatekeeper with their own external application.

## How the External Interface Works

As part of its normal function, a gatekeeper receives certain RAS registration, admission, location, resource, and disengage messages from H.323 endpoints. Typically, the gatekeeper processes these messages and responds to the request. However, with the Cisco IOS Gatekeeper, the GKTMP, and the gatekeeper API, you can supplement or offload the processing of the request to an external application.

In general, the process works as follows:

1. You establish triggers for each external application on the Cisco IOS Gatekeeper. These triggers are based on RAS tags and values.
2. When the Cisco IOS Gatekeeper receives a RAS message from an H.323 endpoint, it compares the message to the triggers.
3. If there is a match, the Cisco IOS Gatekeeper repackages the contents of the RAS message and sends it to the appropriate external application.
4. The gatekeeper API decodes the byte stream and creates a usable C structure.
5. The external application processes the data and sends the results back to the gatekeeper API.
6. The gatekeeper API encodes the response message and sends the data to the Cisco IOS Gatekeeper.
7. The Cisco IOS Gatekeeper performs any additional processing, if necessary, and forwards the results to the requesting H.323 endpoint.

The interaction between the Cisco IOS Gatekeeper and the external application is completely transparent to the H.323 endpoint.

Communication between the Cisco IOS Gatekeeper and the external application is over a TCP connection through the gatekeeper API. Multiple Cisco IOS Gatekeepers can be configured on a single Cisco IOS router as logical gatekeepers. The same TCP connection can be used by all the logical Cisco IOS Gatekeepers. The individual Cisco IOS Gatekeepers are identified by their gatekeeper IDs. This ID is included in all messages that the Cisco IOS Gatekeeper sends to the external application and in all responses that the external application sends back to the Cisco IOS Gatekeeper. If there are different external applications running on the same host, messages to the different external applications can also be multiplexed on the same TCP connection. The external applications are identified by their server IDs.

# How Gatekeeper Triggers Work

By default, the Cisco IOS Gatekeeper does not forward any RAS messages to any external applications. If an application is interested in receiving certain RAS messages, it must register this interest with the Cisco IOS Gatekeeper. To determine which RAS messages the Cisco IOS Gatekeeper forwards to the external application, you can specify trigger parameters. If the Cisco IOS Gatekeeper receives a message that satisfies the specified trigger conditions, the message is forwarded to the external application.

If multiple trigger conditions are specified in a single registration message, the Cisco IOS Gatekeeper treats the trigger conditions as “OR” conditions. In other words, if a RAS message received by the gatekeeper meets any of the trigger conditions the message is sent to the external application.

Trigger conditions are optional. If the Cisco IOS Gatekeeper receives a registration that contains no trigger conditions, it forwards all messages of the specified RAS message type to the external application.

If the Cisco IOS Gatekeeper has a registration for a RAS message type and receives another registration for the same RAS message from the same external application with the same priority, the Cisco IOS Gatekeeper uses the new registration and discards the previous one. The Cisco IOS Gatekeeper allows registrations for the same RAS message type with the same priority from multiple servers.

To indicate that the external application is no longer interested in a message, it must unregister its interest. The contents of the unregistration message must match that of the corresponding registration message before the trigger can be removed.

A Cisco IOS Gatekeeper can be statically (through a command-line interface) or dynamically (through the gatekeeper API) configured with trigger parameters.

**Note** Triggers that are statically configured can be removed only through the command-line interface. Likewise, those triggers that are dynamically configured can be removed or modified only through the gatekeeper API.

# Statically Configured Triggers

Statically configured triggers are established on the router using Cisco IOS commands. To configure triggers using the Cisco IOS command line, do the following:

**Step 1** Access the Cisco IOS Gatekeeper configuration mode. Enter the following command:

```
gatekeeper
```

**Step 2** Enter the trigger configuration mode and specify the RAS message type for the trigger. Enter the following command:

```
server trigger {arq | lcf | lrj | lrq | rrq | urq | rai | drq | brq} gkid priority
server-id server-ip_address server-port
```

**Step 3** If the trigger is to send qualifying messages on a notification-only basis, enter the following command:

```
info-only
```

**Step 4** If you want to limit the qualifying messages based on the destination information, enter the following command:

```
destination-info {e164 | email-id | h323-id} value
```

You can repeat this command to enter multiple destinations. This command cannot be used with an RAI message trigger.

**Step 5** If you want to limit the qualifying messages based on the redirect reason, enter the following command:

```
redirect-reason value
```

You can repeat this command to enter multiple redirect reasons. This command cannot be used with an RAI message trigger.

**Step 6** If you want to limit the qualifying messages based on the remote extension address, enter the following command:

```
remote-ext-address value
```

You can repeat this command to enter multiple remote extension addresses.

**Step 7** If you want to limit the qualifying messages based on the endpoint type, enter the following command:

```
endpoint-type value
```

You can repeat this command to enter multiple endpoint types. This command cannot be used with a DRQ message trigger.

**Step 8** If you want to limit the qualifying messages based on the supported prefix, enter the following command:

```
supported-prefix value
```

You can repeat this command to enter multiple supported prefixes. This command cannot be used with a DRQ message trigger.

**Step 9** When you have specified all the parameters for this trigger, exit trigger submode by entering the following command:

```
exit
```

**Step 10** Repeat steps Step 2 through Step 9 for each trigger that you want to define.

**Step 11** If you want to change the server timeout value for triggers, enter the following command:

```
timer server timeout value
```

To remove a trigger, use the **no server trigger** command. To temporarily suspend a trigger, enter the trigger configuration mode, as described in step Step 2 and enter the **shutdown** subcommand.

For more information about the Cisco IOS commands for configuring triggers, see Chapter 6, "GKTMP Command Reference."

**Note** With statically configured triggers, the gatekeeper initiates the connection to the external application and keeps the connection open for as long as it is running. If the connection is terminated by the external application, the Cisco IOS Gatekeeper periodically attempts to re-establish the connection.

# Dynamically Configured Triggers

Dynamically configured triggers are established using the gatekeeper API and the GKTMP trigger registration messages.

1. The external application creates a trigger and sends it to the Cisco IOS Gatekeeper using the WriteRegisterMessage API function. The triggers are sent in the format for trigger registration messages as prescribed by the GKTMP.
2. In response, the Cisco IOS Gatekeeper sends a message back that indicates whether the registration request has been accepted.

You must send a separate registration message for each message type that you want sent to the external application. If you send a registration message that does not contain any trigger definitions, all messages of the specified type are sent to the external application.

Dynamically configured triggers are removed using the WriteUnregisterMessage API function and GKTMP trigger unregistration messages. Again, the response from the Cisco IOS Gatekeeper indicates whether the unregistration request has been accepted.

## API Functions

You can use the following API functions to dynamically configure triggers:

- WriteRegisterMessage—Sends a registration message to the Cisco IOS Gatekeeper. This function reads the information in the GK_REGISTER_MSG_TYPE structure and sends the contents to the Cisco IOS Gatekeeper using the gkHandle read from the GKAPI_SOCK_INFO_T structure. The header structure, REGISTER_REQUEST_HEADER_TYPE, within each message structure must

contain information for the From, To, and Priority fields. Optionally, if the external application is interested in receiving only notification of a message (not in processing any data for the message), the notificationOnly field should be set to True. Otherwise, it is set to False.

If no filter conditions are to be sent, the parameters within the registration structure should be set to their initialization values or to NULL for pointers. WriteRegisterMessage processes the filters for sending until it reaches the first initialization value for the parameter, or the first null pointer for pointer types.

- WriteUnregisterMessage—Sends an unregistration message to the Cisco IOS Gatekeeper. This function reads the information in the GK_REGISTER_MSG_TYPE structure and sends the contents to the Cisco IOS Gatekeeper using the gkHandle read from the GKAPI_SOCK_INFO_T structure. The header structure, REGISTER_REQUEST_HEADER_TYPE, within each message structure must contain information for the From, To, and Priority fields.

## GKTMP Messages

The format of the GKTMP registration/unregistration request and response messages is as follows:

```
Message line
Message header line 1
Message header line 2
Message header line x

Message body line 1
Message body line 2
Message body line x
```

The request and response messages contain the following fields:

- Message line—A single line indicating whether this message is a REGISTRATION or UNREGISTRATION request from the external application. This line is echoed in the response from the Cisco IOS Gatekeeper. The format is REGISTER *xxx* or UNREGISTER *xxx*.
- Message header—A series of lines indicating the server ID of the external application, the gatekeeper ID of the Cisco IOS Gatekeeper, and the priority of the trigger. The priority indicates the order in which this trigger should be processed with respect to other triggers. The message header also includes a version ID, which indicates the version of the GKTMP. The version ID must be the first header in every GKTMP message.

  For trigger registration requests, if the message contains a body, the header can also contain a line indicating the content length of the body. The message header might also contain a line that indicates whether the external application only wants to be notified of the specified RAS messages that the Cisco IOS Gatekeeper receives. For more information on notification only, see the "Notification-Only Triggers" section on page 3-7.

  For trigger registration and unregistration responses, the header also contains a line that indicates the status of the registration or unregistration request.

  The format of each line is *field*:*value*.
- An empty line.
- Message body (optional)—The body of trigger registration messages contains the RAS tags and values that define the desired triggering parameters. Each triggering parameter occupies a single line. The format of each line is *tag=value*.

  The message body can be included only in trigger registration requests. Trigger registration responses and trigger unregistration requests and responses cannot contain a message body.

For more information about the format of trigger registration and unregistration messages, see Chapter 4, "GKTMP Messages."

With dynamically configured triggers, the external application establishes a TCP connection to the gatekeeper and registers its interest in any of the RAS message types. The external application should then leave the connection open for receiving such messages and for sending its responses. If the external application closes the connection, its registrations are considered cancelled. The Cisco IOS Gatekeeper does not attempt to re-establish the connection.

### Example of a Dynamic Trigger Registration Message

In the following example the trigger registration request indicates that the Cisco IOS Gatekeeper should forward to the external application any RRQ messages from a voice gateway or a gateway with a supported prefix of 1# or 2#:

```
REGISTER RRQ
Version-id: 100
From: server-12
To: gk-dallas1
Priority: 20
Notification-Only:
Content-Length: 29

t=voice-gateway
p=1#
p=2#
```

## Specifying Wildcards in Triggers

Within a trigger, certain wildcard characters are allowed for an alias-address field that contains an E.164 address. Trigger criteria for E.164 alias-addresses can include trailing wildcard characters as follows:

- One or more periods can be used, each denoting a single character
- An asterisk can be used to denote zero or more characters.

Examples of legal E164 address patterns are:

| | |
|---|---|
| 1800..... | The digits 1800 followed by seven characters. |
| 011* | The digits 011 followed by any number of characters. |

Examples of illegal E164 address patterns are:

| | |
|---|---|
| ...4567 | Wildcard characters must be used as trailing characters. They cannot be used at the beginning of a field. |
| 4802*2 | Wildcard characters cannot be used within a field. In this case, the asterisk is interpreted as a literal character. |

## Notification-Only Triggers

If the application needs to be aware of messages but will not be preforming any processing of the message, you can indicate that the messages should be forwarded on a notification-only basis. If notification-only is present in a GKTMP registration message (which means that notification-only is set to true at the API), the Cisco IOS Gatekeeper forwards messages that meet the trigger criteria but does not expect a response. If notification-only is not present (which means that notification-only is set to false at the API), the Cisco IOS Gatekeeper forwards messages that meet the trigger criteria and awaits a corresponding RESPONSE message from the external application.

This header line is typically used for REQUEST RRQ and REQUEST URQ messages, so that the Cisco IOS Gatekeeper can populate an external application's registration database.

# How RAS Messages are Processed

When the Cisco IOS Gatekeeper receives an RAS message that meets the specified trigger conditions, it packages the contents of the fields of the RAS message into the message body of a GKTMP REQUEST message. When the external application receives a request, it must package the response into the message body of a GKTMP RESPONSE message.

The GKTMP specifies formats for exchanging the following types of RAS messages:

- RRQ—Registration request
- RCF—Registration confirm
- RRJ—Registration reject
- URQ—Unregistration request
- ARQ—Admission request
- ACF—Admission confirm
- ARJ—Admission reject
- LRQ—Location request
- LCF—Location confirm
- LRJ—Location reject
- RIP—Request in progress
- RAI—Resource availability indication
- DRQ—Disengage request
- BRQ—Bandwidth request
- BCF—Bandwidth confirm
- BRJ—Bandwidth reject

The URQ, RAI, and DRQ messages are issued as a request from the Cisco IOS Gatekeeper, but do not have a corresponding response. Other messages (RCF, RRJ, ACF, ARJ, BCF, and BRJ) are sent only as responses from the external application.

**Note** The Cisco IOS Gatekeeper does not generate GKTMP Request RRQ messages for lightweight RRQ messages, which are used by H.323 endpoints as a keep-alive mechanism to refresh existing registrations.

The general format of the GKTMP RAS messages is:

```
Message line
Message header line 1
Message header line 2
Message header line x

Message body line 1
Message body line 2
Message body line x
```

These messages include the following fields:

- Message line—A single line indicating whether this message is a REQUEST or RESPONSE. The format is REQUEST *xxx* or RESPONSE *xxx*.
- Message header—A series of lines indicating the server ID of the external application and the gatekeeper ID of the Cisco IOS Gatekeeper. The message header also includes a version ID, which indicates the version of the GKTMP. The version ID must be the first header in every GKTMP message.

  If the message contains a body, the header can also contain a line indicating the content length of the body. The header can also contain a transaction ID, which uniquely identifies the request/response message. The message header might also contain a line that indicates whether the message is being sent on a notification-only basis. For more information on notification only, see the "Notification-Only Triggers" section on page 3-7.

  The format of each line is *field*:*value*.
- An empty line.
- Message body (optional)—The body of trigger registration messages contains the RAS tags and values for the corresponding RAS message. The tags included in the body vary depending on the type of RAS message. Responses from the external application should contain only changed or new body information. The format of each line is *tag=value*.

For more information about the format of GKTMP RAS messages, see Chapter 4, "GKTMP Messages."

How the external application processes requests from the Cisco IOS Gatekeeper depends on the type of RAS message and how the external application has been configured to respond.

**Note** The Cisco IOS Gatekeeper maintains a timeout value for the processing of requests. If a response is not received within the timeout value, the Cisco IOS Gatekeeper assumes the external application is unavailable. Therefore, when the external application receives a message that will take additional time to process, it should send a message back to the Cisco IOS Gatekeeper to request an extension to the timeout. This message is a RESPONSE RIP.

# Processing of *x*RQ Requests

When the external application receives a REQUEST *x*RQ message from the Cisco IOS Gatekeeper it must take one of the following actions:

- Instruct the Cisco IOS Gatekeeper to reject the request. In this case, the external application sends a RESPONSE *x*RJ message to the Cisco IOS Gatekeeper.

- Modify one or more of the fields and return the request to the Cisco IOS Gatekeeper for further processing. In this case, the external application sends a RESPONSE *x*RQ message with the altered information in the body. Only fields that the external application changes can be included in the body. Unchanged fields must not be present in the response message body.
- Indicate no interest in the message and instruct the gatekeeper to continue normal processing. In this case, the external application sends a RESPONSE *x*RQ message with a null message body.
- Complete the processing of the request and send the results to the Cisco IOS Gatekeeper. In this case, the external application sends a RESPONSE *x*CF message. The body of this message must contain all the fields that the Cisco IOS Gatekeeper needs to respond with an *x*CF to the client. This message indicates to the gatekeeper that no further processing is required. If multiple triggers have been configured such that the REQUEST is sent to more than one external application, the RESPONSE *x*CF preempts any other external applications from receiving this message.
- Send no response. This action must be taken only if the request message contains the line Notification-Only: in the header.

## Processing of LCF Requests

An LCF message is sent by a peer gatekeeper to confirm the location of a destination endpoint in its zone. You can configure the Cisco IOS Gatekeeper to forward any LCF messages that it receives to the external application. This gives the application an opportunity to alter any of the fields in the confirmation.

When the external application receives a REQUEST LCF message from the Cisco IOS Gatekeeper, the application must take one of the following actions:

- Confirm the information contained in the request. In this case, the external application sends a RESPONSE LCF with a null message body.
- Alter the information contained in the request. In this case, the external application sends a RESPONSE LCF message with the altered information in the message body. Only fields that the external application changes can be included in the body. Unchanged fields must not be present in the response message body.
- Reject the information contained in the LCF. In this case, the external application sends a RESPONSE LRJ.

## Processing of LRJ Requests

An LRJ message is sent by a peer gatekeeper to reject the location of a destination endpoint, meaning the endpoint does not exist in the peer gatekeeper's zone. You can configure the Cisco IOS Gatekeeper to forward to the external application any LRJ messages that it receives. This gives the external application an opportunity to recommend an alternate destination.

When the external application receives a REQUEST LRJ message from the Cisco IOS Gatekeeper it must take one of the following actions:

- Accept the LRJ. In this case, the external application sends a RESPONSE LRJ with a null message body.
- Suggest an alternate destination. In this case, the external application sends a RESPONSE LCF message with the altered information in the message body. Only fields that the external application changes can be included in the body. Unchanged fields must not be present in the response message body.

# How Security Works

The GKTMP supports the use of CryptoH323Tokens for authentication. The CryptoH323Token is defined in H.225 Version 2 and is used in a "password with hashing" security scheme as described in section 10.3.3 of the H.235 specification.

A cryptoToken can be included in any RAS message and is used to authenticate the sender of the message. The use of cryptoTokens allows you to use a separate database for user ID and password verification.

## CryptoTokens and Cisco Gateways

Cisco gateways support the following levels of authentication:

- Registration—Tokens are generated for RRQ and URQ messages.
- Per-Call—Tokens are generated for ARQ messages.
- All—Tokens are generated for RRQ, URQ, and ARQ messages.

You can configure the level of authentication for the gateway using the Cisco IOS software command–line interface.

CryptoTokens for RRQ, URQ, and the terminating side of ARQ messages contain information about the gateway that generated the token, including the gateway ID (which is the H.323 ID configured on the gateway) and the gateway password. CryptoTokens for the originating side ARQ messages contain information about the user that is placing the call, including the user ID and personal identification number (PIN).

Therefore, if you want to use cryptoTokens for authentication, all clients in your network must include a cryptoToken in every message that they send to the Cisco IOS Gatekeeper.

## Requirements for using CryptoTokens

To participate in this authentication scheme, a GKTMP-based application must have the following:

- Access to a database of user IDs, gateway IDs, and their associated passwords.
- Access to an ASN.1 encoder.

The application should be set up to authenticate the messages that you deem necessary. If you want to authenticate gateways when they register, your application should validate RRQ messages. If you want per-call authentication, your application should validate ARQ messages. Or, you can have your application validate all messages.

## Validating a CryptoToken

To validate a cryptoToken received in a RAS message, the application should:

1. Use the alias in the cryptoToken to look up the associated password.
2. Use the password, the timestamp, and the alias, to ASN.1 encode a ClearToken. The ClearToken is a PwdCertToken. The application should maintain the password and alias as NULL-terminated strings and include the NULL when performing the ASN.1 encoding.
3. Perform an MD5 Hash on the ASN.1 encoded buffer. This results in a 16-byte Hash.
4. Compare the calculated Hash with the one found in the token field of the cryptoEPPwdHash.

If the hash values match, the application should issue a confirmation message (*x*CF) to the gatekeeper, which is transmitted to the gateway. Otherwise, the application should send a rejection message (*x*RJ) with a reject reason of securityDenial.

### CryptoTokens in RAS Messages

The cryptoToken message body line contains a type identifier followed by a colon and a sequence of space separated tag=value parameters that are associated with the particular type of cryptoToken.

For example, a message body line containing a cryptoToken could look like the following:

```
$=E:a=H:gw1-rtp T=940647784 h=FFABCD0067AE12436780167364847343
```

For more information about the parameters included in cryptoTokens, see Chapter 4, "GKTMP Messages."

# GKTMP Message Examples

The following examples show the GKTMP messages that are generated in some uses of the external interface.

## Populating an External Application's Registration Database

An external application might need to maintain a database of active gateways so that it can select gateways for ARQ or LRQ resolution. In this case, triggers can be configured on the Cisco IOS Gatekeepers so that any RRQ or URQ messages will be forwarded to the external application on a notification-only: basis. Example 3-1 shows an RRQ notification for a gateway. Example 3-2 shows a URQ notification for a gateway.

***Example 3-1 RRQ Notification***

```
REQUEST RRQ
Version-id: 100
From: gk1-sj
Notification-only:
Content-Length:90

c=I:171.69.136.205:1720
r=I:171.69.136.205:16523
a=H:gw3-sj
t=voice-gateway
p=2# 99#
```

***Example 3-2 URQ Notification***

```
REQUEST URQ
Version-id: 100
From: gk1-sj
Notification-only:
Content-Length:23

c=I:171.69.136.205:1720
```

# 800 Number Lookup

You might want the Cisco IOS Gatekeeper to forward ARQs to an external application to determine the mapping for an 800 number. Example 3-3 shows an ARQ request from the Cisco IOS Gatekeeper. Example 3-4 shows the corresponding response from the external application.

***Example 3-3 Gatekeeper Request***

```
REQUEST ARQ
Version-id: 100
From: gk1-sj
Transaction-Id: 5de04245
Content-Length: 127

s=E:4085552132
d=E:8005721234
b=560
A=f
m=t
c=f81d4fae-7dec-11d0-a765-00a0c91e6bf6
C=f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

***Example 3-4 External Application Response***

```
RESPONSE ARQ
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04245
Content-Length:14

d=E:4155551212
```

# Internet Call-Waiting

If you have an internet call-waiting (ICW) server in your network, you might want configure the Cisco IOS Gatekeeper to forward all LRQ requests to the ICW server. Example 3-5 shows the LRQ request from the Cisco IOS Gatekeeper.

***Example 3-5 Gatekeeper LRQ Request***

```
REQUEST LRQ
Version-id: 100
From: gk1-sj
Transaction-Id: 5de04246
Content-Length:64

s=H:gk3-la
d=E:4085551111
p=0
c=4085552222
```

If the ICW server determines that the destination (4085551111) is not a subscriber, it sends back a RESPONSE LRQ with a null message body. The Cisco IOS Gatekeeper then proceeds with normal processing. Example 3-6 shows the response.

***Example 3-6 Null Response***

```
RESPONSE LRQ
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04246
```

If the ICW server determines that the destination (4085551111) is a subscriber and is currently logged on, it pings the subscriber to determine how the call should be handled. Because this can take several seconds, the ICW server first sends a RESPONSE RIP to the Cisco IOS Gatekeeper asking for a 60-second extension to the timeout. Example 3-7 shows the response.

***Example 3-7 RIP Response***

```
RESPONSE RIP
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04246
Content-Length:7

d=60000
```

If the subscriber refuses the call, the ICW server sends a rejection to the Cisco IOS Gatekeeper. Example 3-8 shows the response.

***Example 3-8 Rejection***

```
RESPONSE LRJ
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04246
Content-Length:15

R=requestDenied
```

If the subscriber hangs up to accept the call, the ICW server sends a RESPONSE LRQ with a null message body, which instructs the Cisco IOS Gatekeeper to proceed with the call. Example 3-9 shows the response.

***Example 3-9 Null Response***

```
RESPONSE LRQ
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04246
```

If the subscriber chooses to route the call to voicemail (4085553333) and the ICW server knows the IP address of the voicemail gateway (172.45.63.49), the server instructs the Cisco IOS Gatekeeper to route the call to the voicemail system. Example 3-10 shows the response.

***Example 3-10 Response to Reroute***

```
RESPONSE LCF
Version-id: 100
To: gk1-sj
Transaction-Id: 5de04246
Content-Length:94
```

```
d=E:4085553333
D=I:172.45.63.49:1720
r=I:172.45.63.49:13982
t=voice-gateway
X=4085551111
```

# How the API Works

The gatekeeper API is offered as a library that contains the API functions, which are designed to work with GKTMP. An external application must link with the GKAPI object code and call the API functions to communicate with the Cisco IOS Gatekeeper.

The GKAPI includes the following files:

- gk_api.o—The gatekeeper API object code.
- gk_api.h—The gatekeeper API interface header file, which must be included by the application.

The gatekeeper API provides functions and structures that allow an external application to obtain data from and return information to the Cisco IOS Gatekeeper. Using the API functions and structures, as well as some standard functions, the external client application:

1. Establishes a connection with the Cisco IOS Gatekeeper using the GkapiSetupClient function.
2. Monitors the appropriate socket using a standard function.
3. When a connect complete is detected, the application notifies the gatekeeper API using the GkapiClientConnected function.
4. When a read message is detected, it allocates memory for the storage of the message using the GetReadMsgBuffer function.
5. Stores the contents of the message in the appropriate structure using the ReadMsgBuffer function.

**Note** If the message received from the Cisco IOS Gatekeeper is a RAS message that is not supported by the API function, the msgType will be set to MSG_NOT_SUPPORTED. If a response is required, an appropriate response will be constructed by the API function and sent to the Cisco IOS Gatekeeper. The header information in the UNSUPPORTED_MSG_TYPE structure will be filled in by the API function. This situation could occur if the Cisco IOS Gatekeeper has been upgraded to support new messages but the API function has not been correspondingly upgraded.

If the message received from the Cisco IOS Gatekeeper, is not recognized by the API function, the msgType will be set to UNKNOWN_MSG and the STATUS_TYPE will be set to MSG_READ_ERROR. In this case, the external application should close the connection to the Cisco IOS Gatekeeper by calling the CloseGateKeeperConnection function.

If the application has specified the use of non-blocking I/0, the GkapiSetupClient and ReadMsgBuffer functions can return with a CONNECT_IN_PROGRESS or INCOMPLETE_MSG_READ error. These errors indicate that either the connection setup is still in progress or a complete GKTMP message has not been received. If either of these errors is returned, additional socket events will indicate the further processing and completion of these requests. The application should *not* call CloseGateKeeperConnection in these conditions. Instead, the application must monitor the socket using the appropriate handles to detect the additional socket events.

6. Obtains the data from the structure and performs the processing as designed.

7. Frees the memory allocated for the read message using the FreeReadMsgBuffer function.
8. Writes the resulting data to the appropriate structure using the WriteResponseMsg function.

The external application can repeat these steps as often as necessary. If a read error is encountered or if the external application wants to terminate the connection to the Cisco IOS Gatekeeper, the application should use the CloseGateKeeperConnection function.

The API functions and structures are described in Chapter 5, "Gatekeeper API Functions and Structures."

# Linking with the Gatekeeper API

As stated earlier, an external application must be linked with the GKAPI object code and call the API functions in order to communicate with the Cisco IOS Gatekeeper. If you have an external application to use with the gatekeeper API and GKTMP, be sure that you link you it with the gatekeeper API library.

The following is an example makefile for building an application using the GNU "C" compiler and linking with the gatekeeper API library.

This sample makefile is bundled with the GKAPI binary and the header file and can be extracted from the GK software .tar file on the Cisco.com website. The sample file can be modified and used to meet the individual requirements of the end user.

```
rm -f gkapiver.c
echo \#ifndef GKAPI_MAX_VER_STR_LEN >>gkapiver.c
echo \#define GKAPI_MAX_VER_STR_LEN 128 >>gkapiver.c
echo \#endif >>gkapiver.c
echo char version_string\[GKAPI_MAX_VER_STR_LEN\]= \"Compiled `date +"%a
%d-%h-%y %H:%M"` OS target: `uname -sr` \"\; >>gkapiver.c
gcc -g -Wall -c gkapiver.c -o gkapiver
gcc -g gk_api gkapiver gk_application.c
/usr/lib/libintl.a -lsocket -lnsl -ldl -lthread
-lpthread -lposix4 -ogk_application
```

# Guidelines for Using the Gatekeeper API

When you are writing an application that uses the gatekeeper API, keep the following in mind:

- For response messages, the application must send *only* changed or new parameters. Any unchanged fields must not be included in the response message body. If unchanged fields are sent to the Cisco IOS Gatekeeper in a response message, the performance of the Cisco IOS Gatekeeper could be severely impacted.
- For messages received from the Cisco IOS Gatekeeper, the API function removes the tag fields. The type prefix (H:, E:, M: for alias-addresses and I: for transport-addresses) is preserved and is stored in the appropriate structure. The application must interpret the type of address based on the type prefix.
- For responses from the application, the application must insert the type prefix (H:, E:, M: for alias-addresses and I: for transport-addresses). The API function inserts the appropriate tag before constructing the response message.
- For sequence of parameters in messages received from the Cisco IOS Gatekeeper, the API function removes the tag field and stores the parameter in the appropriate structure in the same format as it was read—with the spaces included in the string.

- For sequence of parameters in responses from the application, the application must separate the parameters with spaces. The API function inserts the appropriate tag before constructing the response message.
- For register functions, “sequence of” parameters are not supported. However, the application can have multiple trigger conditions. This is limited by the maximum size of the array in the registration structures.
- The application is responsible for receiving all signals from the operating system. In order for the API function to detect a closed connection with the gatekeeper during a write operation (so that the STATUS_TYPE can be set to TCP_CONNECTION_CLOSED), the application must install a signal handler for SIGPIPE.

# Gatekeeper API Examples

The following examples show how the gatekeeper API functions can be used in an external application. These examples are meant to illustrate how the API functions can be called. They are not examples of actual implementations of the API. Two examples are included in this section; one in which the application is the client and one in which the application is the server.

***Example 3-11 Client Example***

```
#include "gk_api.h"  /* API header file */
#include </usr/include/sys/fcntl.h>
#include </usr/include/sys/socket.h>
#include </usr/include/sys/select.h>
#include <signal.h>

#define APP_VER 1

void sig_int(int sigNo);
STATUS_TYPE BuildRRQRegisterMsg(GKAPI_SOCK_INFO_T *clientConnect);
STATUS_TYPE BuildRRQResponse(GK_READ_MSG_TYPE *ptr,
                             GKAPI_SOCK_INFO_T *connectPtr);


main()
{
    GKAPI_SOCK_INFO_T clientConnect;
    STATUS_TYPE status;
    GK_READ_MSG_TYPE *readMsgPtr;
    struct timeval tval;
    int conn_handle;
    int n;
    fd_set wset, rset;
    BOOLEAN read_pending = FALSE;

    readMsgPtr=NULL;

    /* Install signal handler for SIGPIPE */
    if (signal(SIGPIPE, sig_int) == SIG_ERR) {
        printf("error registering signal \n");
    }

    /* Open Connection to GateKeeper */
    /* Fill in TCP port and IP address of GateKeeper */
    clientConnect.IPAddress = inet_addr("111.222.111.222");
    clientConnect.TCPPort=2000;
```

```
/* Setup the connection for nonblocking I/O */
conn_handle=GkapiSetupClient(&clientConnect, &status, TRUE);

/* Check status for errors */
/* If status == PROCESSING_SUCCESSFUL, no errors were encountered */
/*    status == TCP_CONNECT_ERROR, error in connecting to GateKeeper */
/*    status == TCP_HANDLE_ERROR, error in handle creation */
/* For error conditions, retry connecting to the GateKeeper */

/* Check for following errors:
 * status = MEM_ALLOC_FAIL
 *          INVALID_MSG_SPECIFIED
 *          INVALID_ENDPOINT_SPECIFIED
 *          INVALID_REDIRECT_REASON_SPECIFIED
 *          HEADER_INFO_INCOMPLETE
 *          NULL_POINTER_PASSED
 */

 /* If status is PROCESSING_SUCCESSFUL or CONNECT_IN_PROGRESS,
    wait for connect and read event */
 if (status == CONNECT_IN_PROGRESS) {
    FD_ZERO(&rset);
    FD_SET(conn_handle, &rset);
    wset = rset;
    tval.tv_sec = 1;
    tval.tv_usec = 0;
    if ( (n = select(conn_handle + 1, &rset, &wset, NULL,
                     &tval)) == 0) {
         printf("\nApplication connect timed out");
         CloseGateKeeperConnection(&clientConnect);
         exit(1);
    }

    if (FD_ISSET(conn_handle, &rset) ||
        FD_ISSET(conn_handle, &wset)) {
        status = PROCESSING_SUCCESSFUL;
    } else {
        printf("\nSelect error");
        CloseGateKeeperConnection(&clientConnect);
        exit(1);
    }
 }

 /* If a connect event has occurred tell GKAPI so */
 conn_handle = GkapiClientConnected(&clientConnect, &status, conn_handle);

/* If conn_handle is valid and */
/* If status is PROCESSING_SUCCESSFUL, register triggers if required */
/* Build an RRQ Register message */
status = BuildRRQRegisterMsg(&clientConnect);
/* Check status for errors */
/* If status == PROCESSING_SUCCESSFUL, no errors were encountered */
if ((status == TCP_WRITE_ERROR) ||     /* TCP error encountered */
    (status == TCP_CONNECTION_CLOSED)) { /* TCP connection closed */
     /* Close connection to GateKeeper and free system resources */
     CloseGateKeeperConnection(&clientConnect);
}

for(;;)  {
  FD_ZERO(&rset);
  FD_SET(conn_handle, &rset);
  select(conn_handle + 1, &rset, NULL, NULL, NULL);
  printf("Select event occurred \n");
  if (FD_ISSET(conn_handle, &rset)) {
```

```
/* If a read event has occurred:
 * Allocate a read buffer
 * Call ReadMsgBuffer
 * Process Message
 * Build Response if required
 */
 if (!read_pending)
     readMsgPtr=GetReadMsgBuffer();

 /* Check if readMsgPtr is NULL, if NULL, memory allocation failed. */
 /* if readMsgPtr != NULL, continue    */
     read_pending = FALSE;
     status=ReadMsgBuffer(&clientConnect, readMsgPtr);



     /* Check status for errors */
     /* If status == PROCESSING_SUCCESSFUL, no errors were encountered */
     if ((status == TCP_READ_ERROR) ||      /* TCP error encountered */
         (status == TCP_CONNECTION_CLOSED) || /* TCP connection closed */
         (status == MSG_READ_ERROR)) {     /* Message not understood */
         /* Free the read buffer
            Close connection to GateKeeper and free system resources
          */
         FreeReadMsgBuffer(readMsgPtr);
         CloseGateKeeperConnection(&clientConnect);
         /* Reopen connection to GateKeeper */
     }

     /* Check for other error conditions:
      * status==MEM_ALLOC_FAIL
      * status==NULL_POINTER_PASSED
      */
        FreeReadMsgBuffer(readMsgPtr);

     /* status==INCOMPLETE_MSG_READ */
     /* Call ReadMsgBuffer on the next read event */
     if (status == INCOMPLETE_MSG_READ)
         read_pending = TRUE;

     /* status == PROCESSING_SUCCESSFUL */
     /* Extract message received        */
     switch(readMsgPtr->msgType) {
         case RRQ_REQUEST_MSG:
             status=BuildRRQResponse(readMsgPtr, &clientConnect);
             /* Check status for errors.
              * If TCP_WRITE_ERROR or TCP_CONNECTION_CLOSED
              * call CloseGateKeeperConnection(&clientConnect)
              * Reopen connection to GateKeeper.
              * Check for other errors.
              */
             FreeReadMsgBuffer(readMsgPtr);
             break;

         case ARQ_REQUEST_MSG:
             /* Do processing */
             FreeReadMsgBuffer(readMsgPtr);
             break;

         case MSG_NOT_SUPPORTED:
             FreeReadMsgBuffer(readMsgPtr);
             break;
```

```
                default:
                   FreeReadMsgBuffer(readMsgPtr);
                   break;
             }
          }
       }
}

STATUS_TYPE BuildRRQResponse(GK_READ_MSG_TYPE *ptr,
                         GKAPI_SOCK_INFO_T *connectPtr)
{
    GK_WRITE_MSG_TYPE *writePtr;
    HEADER_INFO_TYPE *headerPtr;
    char buffer1[100];
    char buffer2[100];
    STATUS_TYPE status;

     headerPtr=&ptr->MESSAGE_TYPE.rrqReqMsg.headerInfo;
    /* allocate memory for writePtr, writePtr=malloc(...) */
    /* Fill in msgType and header information */
    writePtr->msgType=RRQ_RESPONSE_MSG;

    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.versionId = APP_VER;
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.from,
                        headerPtr->to);
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.to,
                        headerPtr->from);
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.
                     transactionID, headerPtr->transactionID);

    /* Fill in paramters */
    strcpy(buffer1, "M:joe_smith");
    strcpy(buffer2, "1800");
    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.terminalAlias=buffer1;
    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.supportedPrefix=buffer2;

    /* Send message to GateKeeper */
    status=WriteResponseMsg(connectPtr, writePtr);
    /* If memory was allocated for writePtr, free(writePtr) */
    return(status);
}



STATUS_TYPE BuildRRQRegisterMsg(GKAPI_SOCK_INFO_T *clientConnect)
{
    GK_REGISTER_MSG_TYPE *regPtr;
    STATUS_TYPE status;
    int i=0;
    char buffer1[20];

    /* Allocate memory for regPtr, regPtr=malloc(...) */
    /* After allocating memory:
     * Fill in header info and
     * message parameters if needed
     */

    /* Fill in message type */
    regPtr->msgType = RRQ_REGISTER_MSG;

    /* Fill in header info */
     regPtr->
     REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.versionId =
```

```
        APP_VER;
    strcpy(regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.from,
                              "APPL 1");
    strcpy(regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.to,
                              "GK 1");
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.notificationOnly
                                                                    =FALSE;
    /* Set priority */
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.priority=1;

    /* Specify filters for RRQ message */
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[0]  =
                                                          VOICEGATEWAY;
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[1]  = MCU;

    for (i=2; i<MAX_NUM_ENDPOINT_TYPES; i++) {
        regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[i]  =
                                                 ENDPOINT_INFO_NOT_RCVD;
    }

    strcpy(buffer1, "1#");
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.supportedPrefix[0]=buffer1;
    for (i=1; i< MAX_NUM_SUPPORTED_PREFIX; i++) {
        regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.supportedPrefix[i] = NULL;
    }
    /* Now gatekeeper will only send an RRQ message to the application
     * if filter conditions are satisfied.
     */
    status=WriteRegisterMessage(clientConnect, regPtr);
    /* If memory was allocated for regPtr, free(regPtr) */
    return(status);
}

STATUS_TYPE BuildRRQUnRegisterMsg(GKAPI_SOCK_INFO_T *clientConnect)
{
    GK_UNREGISTER_MSG_TYPE unRegMsg;
    STATUS_TYPE status;

    unRegMsg.versionId = APP_VER;
    strcpy(unRegMsg.from, "APPL 1");
    strcpy(unRegMsg.to, "GK 1");
    unRegMsg.unregisterMsg = RRQ_REGISTER_MSG;
    /* Set priority */
    unRegMsg.priority=1;

    status=WriteUnregisterMessage(clientConnect, &unRegMsg);
    return(status);
}

void sig_int(int sigNo)
{
    switch (sigNo) {
        case SIGPIPE:
            printf("SIGPIPE received\n");
            break;

        /* case ...  */
        default:
    }
}
```

### *Example 3-12 Server Example*

```
#include "gk_api.h"  /* API header file */
#include </usr/include/sys/socket.h>
#include </usr/include/sys/select.h>
#include </usr/include/netinet/in.h>
#include </usr/include/sys/errno.h>
#include <signal.h>

typedef struct client_db_t_ {
        int handle;
        GK_READ_MSG_TYPE *buf;
        GKAPI_SOCK_INFO_T *conn_info;
} client_db_t;

#define MAX_CLIENTS 1024
#define APP_VER 1

void sig_int(int sigNo);
STATUS_TYPE BuildRRQRegisterMsg(GKAPI_SOCK_INFO_T *clientConnect);
STATUS_TYPE BuildRRQResponse(GK_READ_MSG_TYPE *ptr,
                             GKAPI_SOCK_INFO_T *connectPtr);

main()
{
    GKAPI_SOCK_INFO_T ServerInfo;
    GKAPI_TCP_ADDR_INFO_T client_addr;
    STATUS_TYPE status;
    GK_READ_MSG_TYPE *readMsgPtr;
    GKAPI_SOCK_INFO_T *connInfo;
    client_db_t client[MAX_CLIENTS+1];
    int conn_handle, serverHandle, max_fd;
    int i, n;
    fd_set rset;
    BOOLEAN read_pending = FALSE;

    readMsgPtr=NULL;

    for (i=0; i<=MAX_CLIENTS; i++) {
         client[i].handle=0;
         client[i].buf=0;
         client[i].conn_info=0;
    }

    /* Install signal handler for SIGPIPE */
    if (signal(SIGPIPE, sig_int) == SIG_ERR) {
        printf("error registering signal \n");
    }

    /* Open Connection to GateKeeper */
    /* Fill in TCP port and IP address of Application */
    ServerInfo.IPAddress = inet_addr("111.222.111.222");
    ServerInfo.TCPPort=2000;

    /* Setup the connection for nonblocking I/O */
    serverHandle=GkapiSetupServer(&ServerInfo, &status, TRUE);

    /* Check status for errors */
    /* If the serverHandle < 0, there was an error. Check status for
    /* for the error code.
    /* If status == TCP_CONNECT_ERROR, error in connecting to GateKeeper */
    /*    status == TCP_BIND_ERROR, error in connecting to Gatekeeper */
    /*    status == TCP_LISTEN_ERROR, error in connecting to Gatekeeper */
    /*    status == TCP_NONBLOCK_ERROR, error setting up for  */
```

```
/*                                  nonblocking connection */
/*    status == TCP_HANDLE_ERROR, error in handle creation */
/* For error conditions, quit */
if (serverHandle < 0)
    exit(1);

 /* Set up select mask with the server's handle to listen for
  * incoming connections.
  */

 max_fd = serverHandle;

 FD_ZERO(&rset);
 FD_SET(serverHandle, &rset);

 for ( ; ; ) {
    /* If status is PROCESSING_SUCCESSFUL wait for incoming connections
     * and read events */
    n = select(max_fd + 1, &rset, NULL, NULL, NULL);

    /* If the select event has occurred on the server handle,
     * it is a new incoming connection.
     */
    if (FD_ISSET(serverHandle, &rset)) {
        connInfo = (GKAPI_SOCK_INFO_T *)malloc(sizeof(GKAPI_SOCK_INFO_T));
        connInfo->TCPPort = ServerInfo.TCPPort;
        connInfo->IPAddress = ServerInfo.IPAddress;
        conn_handle = GkapiAcceptConnection(connInfo, &status,
                                            serverHandle, &client_addr);
       /* If conn_handle < 0, there is an error. Ignore and continue to
        * to process other select events.
        * If conn_handle is valid, add new connection
        * to select read list
        */
        FD_SET(conn_handle, &rset);

        /* Setup the max file descriptor we need to select on */
        if (conn_handle > max_fd)
            max_fd = conn_handle;

        /* Add this new connection to list of active connections */
        for (i=0; i<MAX_CLIENTS; i++) {
             if (client[i].handle == 0) {
                 client[i].handle = conn_handle;
                 client[i].buf = 0;
                 client[i].conn_info = connInfo;
             }
        }

       /* The application set GK triggers for this connection at
        * this point.
        */
    }

    /*
     * Check to see if the select event is a read occurring
     * on one of the existing connections. If so, have GKAPI process the
     * received buffer.
     */
   for (i=0; n>0,i<=MAX_CLIENTS; i++,n--) {
       if (client[i].handle <= 0)
           continue;

       if (FD_ISSET(client[i].handle, &rset)) {
```

```
 if (client[i].buf == 0) {
     readMsgPtr=GetReadMsgBuffer();
 } else {
     readMsgPtr=client[i].buf;
 }

/* If a read event has occurred:
 * Allocate a read buffer if it isn't a pending read.
 * Call ReadMsgBuffer
 * Process Message
 * Build Response if required
 */
 if(readMsgPtr != NULL) {
    status=ReadMsgBuffer(client[i].conn_info, readMsgPtr);

  /* Check if readMsgPtr is NULL, if NULL,
   * memory allocation failed.
   */
  /* if readMsgPtr != NULL, continue    */
    if(status == PROCESSING_SUCCESSFUL) {
       client[i].buf = 0;
       /* Process the Message */
       /* Extract message received        */
       switch(readMsgPtr->msgType) {
       case RRQ_REQUEST_MSG:
           status=BuildRRQResponse(readMsgPtr, &ServerInfo);
           /* Check status for errors.
            * If TCP_WRITE_ERROR or TCP_CONNECTION_CLOSED
            * call CloseGateKeeperConnection(&ServerInfo)
            * Reopen connection to GateKeeper.
            * Check for other errors.
            */
          FreeReadMsgBuffer(readMsgPtr);
          break;

       case ARQ_REQUEST_MSG:
           /* Do processing */
           FreeReadMsgBuffer(readMsgPtr);
          break;

      case MSG_NOT_SUPPORTED:
          FreeReadMsgBuffer(readMsgPtr);
          break;

      default:
          FreeReadMsgBuffer(readMsgPtr);
          break;
      }

   }  /* End of status == PROCESSING_SUCCESSFUL */

   /* Check status for errors */

   if ((status == TCP_READ_ERROR) ||  /* TCP error encountered */
       (status == TCP_CONNECTION_CLOSED) || /*connection closed*/
       (status == MSG_READ_ERROR)) { /* Message not understood */
        /* Free the read buffer
         * Close connection to GateKeeper and
         * free system resources
         */
       FreeReadMsgBuffer(readMsgPtr);
       CloseGateKeeperConnection(client[i].conn_info);
      /* Reopen connection to GateKeeper */
   }
```

```
                 /* Check for other error conditions:
                  * status==MEM_ALLOC_FAIL
                  * status==NULL_POINTER_PASSED
                  */
                 FreeReadMsgBuffer(readMsgPtr);

                /* status==INCOMPLETE_MSG_READ */
                /* Call ReadMsgBuffer on the next read event */
                if (status == INCOMPLETE_MSG_READ)
                    client[i].buf = readMsgPtr;

          }
        }
      }
    }
}

STATUS_TYPE BuildRRQResponse(GK_READ_MSG_TYPE *ptr,
                         GKAPI_SOCK_INFO_T *connectPtr)
{
    GK_WRITE_MSG_TYPE *writePtr;
    HEADER_INFO_TYPE *headerPtr;
    char buffer1[100];
    char buffer2[100];
    STATUS_TYPE status;

     headerPtr=&ptr->MESSAGE_TYPE.rrqReqMsg.headerInfo;
    /* allocate memory for writePtr, writePtr=malloc(...) */
    /* Fill in msgType and header information */
    writePtr->msgType=RRQ_RESPONSE_MSG;

    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.versionId = APP_VER;
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.from,
                        headerPtr->to);
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.to,
                        headerPtr->from);
    strcpy(writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.headerInfo.
                    transactionID, headerPtr->transactionID);

    /* Fill in parameters */
    strcpy(buffer1, "M:joe_smith");
    strcpy(buffer2, "1800");
    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.terminalAlias=buffer1;
    writePtr->WRITE_MESSAGE_TYPE.rrqRespMsg.supportedPrefix=buffer2;

    /* Send message to GateKeeper */
    status=WriteResponseMsg(connectPtr, writePtr);
    /* If memory was allocated for writePtr, free(writePtr) */
    return(status);
}



STATUS_TYPE BuildRRQRegisterMsg(GKAPI_SOCK_INFO_T *ServerInfo)
{
    GK_REGISTER_MSG_TYPE *regPtr;
    STATUS_TYPE status;
    int i=0;
    char buffer1[20];

    /* Allocate memory for regPtr, regPtr=malloc(...) */
    /* After allocating memory:
     * Fill in header info and
```

```
     * message parameters if needed
     */

    /* Fill in message type */
    regPtr->msgType = RRQ_REGISTER_MSG;

    /* Fill in header info */
     regPtr->
     REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.versionId =
         APP_VER;
    strcpy(regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.from,
                              "APPL 1");
    strcpy(regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.to,
                              "GK 1");
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.notificationOnly
                                                                    =FALSE;
    /* Set priority */
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.headerInfo.priority=1;

    /* Specify filters for RRQ message */
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[0]  =
                                                          VOICEGATEWAY;
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[1]  = MCU;

    for (i=2; i<MAX_NUM_ENDPOINT_TYPES; i++) {
        regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.terminalType[i]  =
                                               ENDPOINT_INFO_NOT_RCVD;
    }

    strcpy(buffer1, "1#");
    regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.supportedPrefix[0]=buffer1;
    for (i=1; i< MAX_NUM_SUPPORTED_PREFIX; i++) {
        regPtr->REGISTRATION_MESSAGE_TYPE.rrqRegMsg.supportedPrefix[i] = NULL;
    }
    /* Now gatekeeper will only send an RRQ message to the application
     * if filter conditions are satisfied.
     */
    status=WriteRegisterMessage(ServerInfo, regPtr);
    /* If memory was allocated for regPtr, free(regPtr) */
    return(status);
}

STATUS_TYPE BuildRRQUnRegisterMsg(GKAPI_SOCK_INFO_T *ServerInfo)
{
    GK_UNREGISTER_MSG_TYPE unRegMsg;
    STATUS_TYPE status;

    unRegMsg.versionId = APP_VER;
    strcpy(unRegMsg.from, "APPL 1");
    strcpy(unRegMsg.to, "GK 1");
    unRegMsg.unregisterMsg = RRQ_REGISTER_MSG;
    /* Set priority */
    unRegMsg.priority=1;

    status=WriteUnregisterMessage(ServerInfo, &unRegMsg);
    return(status);
}

void sig_int(int sigNo)
{
    switch (sigNo) {
        case SIGPIPE:
            printf("SIGPIPE received\n");
            break;
```

```
            /* case ... */
            default:
        }
    }
```

CHAPTER 4

# GKTMP Messages

This chapter describes GKTMP messages and contains the following sections:



The GKTMP messages are used for communication between the Cisco IOS Gatekeeper and the external application. There are two types of GKTMP messages:

- GKTMP RAS Messages—Used to exchange the contents RAS messages between the Cisco IOS Gatekeeper and the external application.
- Trigger Registration Messages—Used by the external application to indicate to the Cisco IOS Gatekeeper which RAS message should be forwarded.

# GKTMP RAS Messages

The general format of all GKTMP RAS messages is as follows:

- Single message line
- One or more message header lines
- Blank line, which separates the message header from the message body
- Zero or more message body lines

## Message Line

Each GKTMP RAS message is either a request or a response. Requests are generated by the Cisco IOS Gatekeeper and responses are generated by the external application.

The first line of each GKTMP RAS message sent by the Cisco IOS Gatekeeper uses the format:

`REQUEST` *`RAS_message_type`*

The first line of each GKTMP RAS message sent by the external application uses the format:

`RESPONSE` *`RAS_message_type`*

Possible RAS message types are as follows:

- RRQ—Registration request
- RCF—Registration confirm

- RRJ—Registration reject
- URQ—Unregistration request
- ARQ—Admission request
- ACF—Admission confirm
- ARJ—Admission reject
- LRQ—Location request
- LCF—Location confirm
- LRJ—Location reject
- RIP—Request in progress
- DRQ—Disengage request
- RAI—Resource availability information
- BRQ—Bandwidth request
- BCF—Bandwidth confirm
- BRJ—Bandwidth reject

**Note** The Cisco IOS Gatekeeper does not generate GKTMP Request RRQ messages for lightweight RRQ messages, which are used by H.323 endpoints as a keep-alive mechanism to refresh existing registrations.

# Message Header

The message line is immediately followed by the message header. Each message header contains a field name and a value, separated by a colon (*field*:*value*). Table 4-1 shows the possible fields:

***Table 4-1 Message Header Fields***

| Field Names | Field Values |
|---|---|
| Version-Id | Version of the protocol that the sender is running. The version ID consists of a major number (gk_major) and a minor number (gk_minor). For example, version 1 is represented as 100. |
| From | String that identifies the originator of the message. For requests from the Cisco IOS Gatekeeper, this field contains the gatekeeper ID. For responses from the external application, this field contains the server ID. |
| To | String that identifies the receiver of the message. For requests from the Cisco IOS Gatekeeper, this field contains the server ID. For responses from the external application, this field contains the ID of the gatekeeper that initiated the request. |
| Content-Length | Number of octets contained in the message body. If the message body is null, this field can be omitted. |

*Table 4-1 Message Header Fields*

| Field Names | Field Values |
|---|---|
| Transaction-Id | String that identifies the transaction. If this field is present in the request from the Cisco IOS Gatekeeper, it must be echoed in the response from the external application. |
| Notification-Only | None. No value is included after the colon. If this field name is present, it indicates to the external application no response should be sent. Request URQ must contain this field. Also, Request RRQ contains this field when that message is used to populate the external application's registration database. |

The message header is followed immediately by a blank line.

# Message Body

The message body follows the blank line. Each line in the message body contains a tag and a value, separated by an equal sign (*tag=value*). The tags are case-sensitive and denote an RAS message field. The possible tags depend on the GKTMP RAS message.

**Note** If the message body is null, the message must terminate with a CRFL after the message header.

In some cases, depending on the field type, the value is preceded a value-type identifier followed by a colon (*tag=type*:*value*).

Possible field types are as follows:

- Alias-Address—This type of field can contain a series of addresses separated by spaces. Each is preceded by a value-type identifier that indicates the type of address. H indicates that the address is an H.323 ID; E indicates that the address is an E.164 address; M indicates that the address is an e-mail ID.
- Transport-Address—This type of field contains an address. Currently, only one value-type identifier is possible for this field type. That is I, which indicates that the address is an IP version 4 address. The address is specified in dotted-decimal notation and can be followed by a colon and a port number.
- Endpoint-Type—This type of field indicates the type of endpoint. Possible values are: gatekeeper, terminal, mcu, proxy, voice-gateway, h320-gateway, and other-gateway.
- Supported-Prefix—This type of field indicates a supported technology prefix. Possible values are the digits 0 through 9 and the pound sign (#).
- Globally-Unique-Identifier (GUID)—This type of field contains the 16-octet conference ID or call ID that uniquely identifies the call or conference. The IDs are specified in hexadecimal format.
- Bandwidth—This type of field contains an unsigned integer from 0 through 4294967295 that indicates the bandwidth in 100 bits per second.
- Boolean—This type of field contains a single character. T or t for true; F or f for false.
- IA5 String—This type of field contains characters from the International Alphabet 5 (IA5), which is a character set defined by the ITU X.400 Message Handling System specification.

- cryptoToken—This type of field contains one of the cryptoToken types defined for the CryptoH323Token field specified in H.225. Currently, the only type of cryptoToken supported is the cryptoEPPwdHash.
- HASHED-EncodedPwdCertToken—This type of field contains a 16 octet IA5String. It represents the RAS Message Digest 5 (MD5) hashed encoded PwdCertToken.
- TimeStamp—This type of field contains a 32-bit integer that represents Universal Time Coordinated (UTC) time.
- OBJECT-IDENTIFIER—This type of field contains a sequence of non-negative integer values separated by dots, which is used to uniquely identify an object.
- UseSpecifiedTransport—This type of field contains a string that indicates the transport layer that is used for the signaling: Annex E/UDP or TCP.
- AlternateGK—This type of field contains a set of fields enclosed in braces ({ }). Each field is identified by a tag and separated from the other fields by SP (ASCII space, 0x20) characters. This field can contain more than one set of fields, each enclosed by braces.
- AlternateEndpoint—This type of field contains a set of fields enclosed in braces. Each field is identified by a tag and separated from the other fields by SP (ASCII space, 0x20) characters. A message body line containing an AlternateEndpoint field must pertain to a single endpoint. Multiple call signal addresses and tokens that pertain to the same endpoint can be provided in a single message body line. If there are multiple AlternateEndpoints, each pertaining to a different H.323 endpoint, the information about the alternate endpoints must be provided in separate message body lines.
- AlternateTransportAddress—This type of field contains a single sub-field enclosed in braces. The fields within the braces pertain to a single instance of a RAS AlternateTransportAddress structure. They are defined as a Transport-Address and are encoded as defined for the Transport-Address field.
- clearToken—This type of field contains a set of fields enclosed in braces. Each field is identified by a tag and separated from the other fields by SP (ASCII space, 0x20) characters. The fields within the braces pertain to a single instance of a RAS ClearToken structure. However, the message line of a clearToken field can contain multiple instances, each enclosed in braces and separated by a space character. The clearToken field can be embedded within an AlternateEndpoint field.
- remoteZone—This type of field contains a set of fields enclosed in braces. Each field is identified by a tag and separated from the other fields by SP (ASCII space, 0x20) characters. The fields within the braces pertain to a single instance of a remoteZone structure. However, the message line of a remoteZone field can contain multiple instances, each enclosed in braces and separated by a space character.

This section describes the possible fields for each message. When the external application sends a response, it includes only the fields that it has altered. Unaltered fields must not be included.

# Registration Messages

Registration messages are used to control which H.323 endpoints are in the gatekeeper's zone.

This section describes the following:

- Request RRQ
- Response RRQ
- Response RCF
- Response RRJ

## Request RRQ

This message is sent from the Cisco IOS Gatekeeper to the external application when an H.323 endpoint wants to join the zone. This message can be used to populate the external application's registration database. In this case, the request is sent as a notification only and no response is expected from the external application.

Table 4-2 shows the possible Request RRQ tags:

***Table 4-2 Request RRQ Tags***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | Transport-Address | Required | RRQ:callSignalAddress |
| r | Transport-Address | Required | RRQ:rasAddress |
| a | Alias-Address | Optional | RRQ:terminalAlias |
| t | Endpoint-Type | Required | RRQ:terminalType |
| P | Supported-Prefix | Optional | RRQ:terminalType:gateway:protocol:*:supportedPrefixes |
| $ | cryptoToken | Optional | RRQ:cyptoTokens |
| T | clearToken | Optional | RRQ:tokens |
| N | AlternateTransportAddr | Optional | RRQ:AlternateTransportAddress |

If the message contains a cryptoToken field with a value of cryptoEPPwdHash, the additional fields shown in Table 4-3 are included:

***Table 4-3 Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| a | Alias-Address | Required | CryptoH323Token:cryptoEPPwdHash:alias |
| t | TimeStamp | Required | CryptoH323Token:cryptoEPPwdHash:timestamp |
| h | HashedToken | Required | CryptoH323Token:cryptoEPPwdHash:token |

If the message contains a clearToken field, the additional fields shown in Table 4-4 are included:

***Table 4-4 Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| O | OBJECT-IDENTIFIER | Required | tokens:objectIdentifier |
| p | IA5string | Optional | tokens:password |
| t | integer | Optional | tokens:timestamp |
| s | IA5string | Optional | tokens:challengeString |
| r | integer | Optional | tokens:random |
| G | IA5string | Optional | tokens:generalID |

*Table 4-4 Additional Fields*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| o | OBJECT-IDENTIFIER | Optional | tokens:nonStandard:objectIdentifier |
| d | IA5string | Optional | tokens:nonStandard:data |

If the message contains an AlternateTransportAddr field, the additional field shown in Table 4-5 is included:

*Table 4-5 Additional Field*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| I | Transport-Address | Required | IP address and port for Annex E |

## Response RRQ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request RRQ message. If the external application has no interest in the Request RRQ message, it returns a Response RRQ with a null body. Otherwise, the external application modifies the fields as appropriate and sends the response with the updated information to the Cisco IOS Gatekeeper for further processing.

For Response RRQ, the possible tags are shown in Table 4-6:

*Table 4-6 Response RRQ Tags*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| a | Alias-Address | Optional | RRQ:terminalAlias |
| p | Supported-Prefix | Optional | RRQ:terminalType:gateway:protocol:<br>*:supportedPrefixes |

## Response RCF

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request RRQ. This message indicates that the external application has completed the processing of the request.

For Response RCF, the possible tags are shown in Table 4-7:

*Table 4-7 Response RCF Tags*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| a | Alias-Address | Optional | RRQ:terminalAlias |
| p | Supported-Prefix | Optional | RRQ:terminalType:gateway:protocol:<br>*:supportedPrefixes |
| G | AlternateGK | Optional | RCF:alternateGatekeeper |

If the message contains an AlternateGK field, the additional fields shown in Table 4-8 are included:

***Table 4-8 Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| r | Transport-Address | Required | AlternateGK:rasAddress |
| g | Alias-Address | Optional | AlternateGK:gatekeeperIdentifier |
| n | Boolean | Required | AlternateGK:needToRegister |
| p | integer | Required | AlternateGK:priority |

## Response RRJ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request RRQ. It indicates that the Cisco IOS Gatekeeper should reject the request for the specified reason.

For Response RRJ, the possible tag is shown in Table 4-9:

***Table 4-9 Response RRJ Tag***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| R | RRJ-Reason | Required | RRJ:rejectReason |

Possible values for the rejectReason are:

- undefinedReason
- securityDenial
- resourceUnavailable

# Unregistration Message

Unregistration messages are used to remove an H.323 endpoint from a gatekeeper zone.

This section describes the following:

- Request URQ

## Request URQ

This message is sent from the Cisco IOS Gatekeeper to the external application when the H.323 endpoint wants to leave the zone or when its registration expires. This request is sent as a notification only. No response is generated by the external application.

For Request URQ, the possible tag is shown in Table 4-10:

*Table 4-10 Request URQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | Transport-Address | Required | URQ:callSignalAddress |

# Admission Messages

Admission messages are used to control which H.323 endpoints can participate in calls.

This section describes the following:

- Request ARQ
- Response ARQ
- Response ACF
- Response ARJ

## Request ARQ

This message is sent from the Cisco IOS Gatekeeper to the external application when an H.323 endpoint wants to initiate a call.

For Request ARQ, the possible tags are shown in Table 4-11:

*Table 4-11 Request ARQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| s | Alias-Address | Required | ARQ:srcInfo |
| S | Transport-Address | Optional | ARQ:srcCallSignalAddress |
| d | Alias-Address | Optional | ARQ:destinationInfo |
| D | Transport-Address | Optional | ARQ:destCallSignalAddress |
| x | Alias-Address | Optional | ARQ:destExtraCallInfo |
| b | Bandwidth | Required | ARQ:bandWidth |
| A | Boolean | Required | ARQ:answerCall |
| c | GUID | Optional | ARQ:callIdentifier |
| C | GUID | Required | ARQ:conferenceID |
| m | Boolean | Optional | ARQ:canMapAlias |
| e | IA5String | Optional | ARQ:nonStandardData:redirectNumber |
| E | integer | Optional | ARQ:nonStandardData:redirectReason[1] |
| p | integer | Optional | ARQ:nonStandardData:callingPartyNumOctet3a[2] |
| w | IA5string | Optional | ARQ:nonStandardData:displayIE |
| i | TransportAddress | Required | arqing-endpoint identifier[3] |
| $ | cryptoToken | Optional | ARQ:cyptoTokens |

*Table 4-11 Request ARQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| T | clearToken | Optional | ARQ:tokens |
| B | IA5string | Optional | ARQ:nonStandardData:interfaceSpecific:BillingInfo |
| I | IA5string | Optional | ARQ:nonStandardData:interfaceDescriptor |

Possible values for the redirectReason are:

- 0—Unknown
- 1—Call forwarding busy or called DTE busy
- 2—Call forwarded, no reply
- 4—Call deflection
- 9—Called DTE out of order
- 10—Call forwarding by the called DTE
- 15—Call forwarding unconditional or systematic call redirection

CallingPartyNumOctet3a is from the Q.931 Setup octet 3a of calling party number.

When an H.323 endpoint sends an ARQ to the Cisco IOS Gatekeeper, it includes its endpointIdentifier. Because this value is local and has meaning to the Cisco IOS Gatekeeper only and not to the external application, the Cisco IOS Gatekeeper substitutes a more meaningful value of CallSignalAddress in its Request ARQ messages.

If the message contains a cryptoToken field with a value of cryptoEPPwdHash, the additional fields shown in Table 4-12 are included:

*Table 4-12 Additional Fields*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| a | Alias-Address | Required | CryptoH323Token:cryptoEPPwdHash:alias |
| t | TimeStamp | Required | CryptoH323Token:cryptoEPPwdHash:timestamp |
| h | HashedToken | Required | CryptoH323Token:cryptoEPPwdHash:token |

If the message contains a clearToken field, the additional fields shown in Table 4-13 are included:

*Table 4-13 Additional Fields*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| O | OBJECT-IDENTIFIER | Required | tokens:objectIdentifier |
| p | IA5string | Optional | tokens:password |
| t | integer | Optional | tokens:timestamp |
| s | IA5string | Optional | tokens:challengeString |
| r | integer | Optional | tokens:random |

*Table 4-13 Additional Fields*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| G | IA5string | Optional | tokens:generalID |
| o | OBJECT-IDENTIFIER | Optional | tokens:nonStandard:objectIdentifier |
| d | IA5string | Optional | tokens:nonStandard:data |

## Response ARQ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request ARQ message. If the external application has no interest in the Request ARQ message, it returns a Response ARQ with a null body. Otherwise, it modifies the fields as appropriate and sends the response with the updated information to the Cisco IOS Gatekeeper for further processing.

For Response ARQ, the possible tags are shown in Table 4-14:

*Table 4-14 Response ARQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | ARQ:destinationInfo |
| D | Transport-Address | Optional | ARQ:destCallSignalAddress |
| x | Alias-Address | Optional | ARQ:destExtraCallInfo |
| b | Bandwidth | Optional | ARQ:bandWidth |
| e | IA5String | Optional | ARQ:nonStandardData:redirectNumber |
| E | integer | Optional | ARQ:nonStandardData:redirectReason |
| w | IA5string | Optional | ARQ:nonStandardData:displayIE |
| z | remoteZone | Optional | None |
| T | clearToken | Optional | ARQ:tokens |
| c | integer | Optional | None[1] |
| p | integer | Optional | None[2] |
| A | alternateEndpoint | Optional | ARQ:alternateEndpoints |

1. Reflects the cost value of the primary endpoint, if any, whose address is returned in the ‘D’ field of this message. It should only be sent if the endpoint is filled in.
2. Reflects the priority value of the primary endpoint, if any whose address is returned in the ‘D’ field of this message. It should only be sent if the endpoint is filled in.

The external application has the option of reducing the bandwidth.

If the message contains a remoteZone field, the additional fields shown in Table 4-15 are included:

*Table 4-15 Additional Fields*

| Tag | Field Type | Required or Optional | Description |
|---|---|---|---|
| r | Transport-Address | Required | RAS address of the zone |
| c | Integer | Optional | Cost value associated with the zone |
| p | Integer | Optional | Priority value associated with the zone |
| T | clearToken | Optional | ARQ:tokens |

If this field is included, the Cisco IOS Gatekeeper sends LRQs to all the listed zones. The zone with the least cost and highest priority that returns and LCF is chosen for inclusion in the ACF that is sent to the endpoint.

If the message contains an alternateEndpoint field, the additional fields shown in Table 4-16 are included:

*Table 4-16 Additional Fields*

| Tag | Field Type | Required or Optional | Description |
|---|---|---|---|
| C | Integer | Optional | Cost value associated with the zone |
| p | Integer | Optional | Priority value associated with the zone |

## Response ACF

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request ARQ. The message indicates that the external application has completed the processing of the request.

For Response ACF, the possible tags are shown in Table 4-17:

*Table 4-17 Response ACF*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | ACF:destinationInfo |
| D | Transport-Address | Required | ACF:destCallSignalAddress |
| x | Alias-Address | Optional | ACF:destExtraCallInfo |
| X | Alias-Address | Optional | ACF:remoteExtensionAddress |
| b | Bandwidth | Optional | ARQ:bandWidth |
| t | Endpoint-type | Optional | ACF:destinationType |
| T | ClearToken | Optional | ACF:tokens |
| A | AlternateEndpoint | Optional | ACF:alternateEndpoints |
| N | AlternateTransportAddr | Optional | ACF:alternateTransportAddress |
| u | useSpecifiedTransport | Optional | ACF:useSpecifiedAddress |

If the message contains an AlternateEndpoint field, the additional fields shown in Table 4-18 are included:

***Table 4-18 Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | Transport-Address | Required | alternateEndpoints:callSignalAddress |
| T | clearToken | Optional | alternateEndpoints:tokens |

If the AlternateEndpoint field contains a clearToken field, the additional fields shown in Table 4-19 are included:

***Table 4-19 Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| O | OBJECT-IDENTIFIER | Required | tokens:objectIdentifier |
| p | IA5string | Optional | tokens:password |
| t | integer | Optional | tokens:timestamp |
| s | IA5string | Optional | tokens:challengeString |
| r | integer | Optional | tokens:random |
| G | IA5string | Optional | tokens:generalID |
| o | OBJECT-IDENTIFIER | Optional | tokens:nonStandard:objectIdentifier |
| d | IA5string | Optional | tokens:nonStandard:data |

If the message contains an AlternateTransportAddr field, the additional field shown in Table 4-20 is included:

***Table 4-20 Additional Field***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| I | Transport-Address | Required | IP address and port for Annex E |

## Response ARJ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request ARQ. The message indicates that the Cisco IOS Gatekeeper should reject the request for the specified reason.

For Response ARJ, the possible tag is shown in Table 4-21:

*Table 4-21 Response ARJ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| R | ARJ-Reason | Required | ARJ:rejectReason |

Possible values for rejectReason are:

- calledPartyNotRegistered
- invalidPermission
- requestDenied
- undefinedReason
- resourceUnavailable
- securityDenial

# Location Messages

Location messages are used by gatekeepers to communicate with each other to process interzone calls.

This section describes the following:

- Request LRQ
- Response LRQ
- Request LCF
- Response LCF
- Request LRJ
- Response LRJ

## Request LRQ

This message is sent from the Cisco IOS Gatekeeper to the external application when the Cisco IOS Gatekeeper has received an interzone location request.

For Request LRQ, the possible tags are shown in Table 4-22:

*Table 4-22 Request LRQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| s | Alias-Address | Optional | LRQ:srcInfo |
| d | Alias-Address | Required | LRQ:destinationInfo |
| e | IA5String | Optional | LRQ:nonStandardData:redirectNumber |
| E | integer | Optional | LRQ:nonStandardData:redirectReason[1] |
| p | integer | Optional | LRQ:nonStandardData:callingPartyNumOctet3a[2] |
| w | IA5String | Optional | LRQ:nonStandardData:displayIE |

*Table 4-22 Request LRQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | IA5String | Optional | LRQ:nonStandardData:callingPartyNum |
| T | clearToken | Optional | LRQ:tokens |

Possible values for the redirectReason are:

- 0—Unknown
- 1—Call forwarding busy or called DTE busy
- 2—Call forwarded, no reply
- 4—Call deflection
- 9—Called DTE out of order
- 10—Call forwarding by the called DTE
- 15—Call forwarding unconditional or systematic call redirection

CallingPartyNumOctet3a is from the Q.931 Setup octet 3a of calling party number.

## Response LRQ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request LRQ message. If the external application has no interest in the Request LRQ message, it returns a Response LRQ with a null body. Otherwise, it modifies the fields as appropriate and sends the response with the updated information to the Cisco IOS Gatekeeper for further processing.

For Response LRQ, the possible tags are shown in Table 4-23:

*Table 4-23 Response LRQ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | LRQ:destinationInfo |
| z | remoteZone | Optional | None |
| T | clearToken | Optional | ARQ:tokens |
| c | integer | Optional | None[1] |
| p | integer | Optional | None[2] |
| A | alternateEndpoint | Optional | ARQ:alternateEndpoints |

1. Reflects the cost value of the primary endpoint, if any, whose address is returned in the ‘D’ field of this message. It should only be sent if the endpoint is filled in.
2. Reflects the priority value of the primary endpoint, if any whose address is returned in the ‘D’ field of this message. It should only be sent if the endpoint is filled in.

If the message contains a remoteZone field, the additional fields shown in Table 4-24 are included:

*Table 4-24 Additional Fields*

| Tag | Field Type | Required or Optional | Description |
|---|---|---|---|
| r | Transport-Address | Required | RAS address of the zone |
| c | Integer | Optional | Cost value associated with the zone |
| p | Integer | Optional | Priority value associated with the zone |
| T | clearToken | Optional | LRQ:tokens |

If this field is included, the Cisco IOS Gatekeeper will send the original LRQs to all the listed zones.

If the message contains an alternateEndpoint field, the additional fields shown in Table 4-25 are included:

*Table 4-25 Additional Fields*

| Tag | Field Type | Required or Optional | Description |
|---|---|---|---|
| C | Integer | Optional | Cost value associated with the zone |
| p | Integer | Optional | Priority value associated with the zone |

## Request LCF

This message is sent from the Cisco IOS Gatekeeper to the external application when the Cisco IOS Gatekeeper has received an LCF from the remote Cisco IOS Gatekeeper. This gives the external application an opportunity to accept (Response LCF), modify (Response LCF), or reject (Response LRJ) the information contained in the LCF.

For Request LCF, the possible tags are shown in Table 4-26:

*Table 4-26 Request LCF*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| s | Alias-Address | Optional | LRQ:srcInfo |
| e | IA5String | Optional | LRQ:nonStandardData:redirectNumber |
| E | integer | Optional | LRQ:nonStandardData:redirectReason |
| p | integer | Optional | LRQ:nonStandardData:callingPartyNumOctet3a |
| w | IA5String | Optional | LRQ:nonStandardData:displayIE |
| c | IA5String | Optional | LRQ:nonStandardData:callingPartyNum |
| d | Alias-Address | Required | LRQ/LCF:destinationInfo |
| D | Transport-Address | Required | LCF:callSignalAddress |
| r | Transport-Address | Required | LCF:rasAddress |
| x | Alias-Address | Optional | LCF:destExtraCallInfo |
| X | Alias-Address | Optional | LCF:remoteExtensionAddress |
| t | Endpoint-Type | Optional | LCF:destinationType |
| N | AlternateTransportAddr | Optional | LCF:AlternateTransportAddress |
| u | useSpecifiedTransport | Optional | ACF:useSpecifiedAddress |
| T | clearToken | Optional | LCF:tokens |

The destinationInfo from the LCF is used if one is available. Otherwise, the destinationInfo from the LRQ is used.

If the message contains an AlternateTransportAddr field, the following additional field shown in Table 4-27 is included:

*Table 4-27 Additional Field*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| I | Transport-Address | Required | IP address and port for Annex E |

## Response LCF

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request LRQ. The message indicates that the external application has completed the processing of the request.

This message can also be sent to the Cisco IOS Gatekeeper from the external application in response to a Request LCF or a Request LRJ. In the case of a Request LCF, the response can contain:

- A null message body, which indicates that the external application accepts the information in the Request LCF.
- Modified fields, which indicates that the external application wants to use different values than those included in the Request LCF.

In the case of a Request LRJ, the response contains an alternate destination.

For Response LCF, the possible tags are shown in Table 4-28:

***Table 4-28 Response LCF***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | LCF:destinationInfo |
| D | Transport-Address | Required | LCF:destCallSignalAddress |
| r | Transport-Address | Required | LCF:rasAddress |
| x | Alias-Address | Optional | LCF:destExtraCallInfo |
| X | Alias-Address | Optional | LCF:remoteExtensionAddress |
| t | Endpoint-Type | Optional | LCF:destinationType |
| A | AlternateEndpoint | Optional | ACF:alternateEndpoints |
| N | AlternateTransportAddr | Optional | LCF:AlternateTransportAddress |
| u | useSpecifiedTransport | Optional | ACF:useSpecifiedAddress |
| T | clearToken | Optional | LCF:tokens |

**Note** The D and r are not required if the Response LCF is being sent in reply to a Request LCF.

If the message contains an AlternateTransportAddr field, the additional field shown in Table 4-29 included:

***Table 4-29 Additional Field***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| I | Transport-Address | Required | IP address and port for Annex E |

## Request LRJ

This message is sent from the Cisco IOS Gatekeeper to the external application when the Cisco IOS Gatekeeper has received an LRJ from a remote Cisco IOS Gatekeeper. This gives the Cisco IOS Gatekeeper the opportunity to accept the rejection (Response LRJ) or propose an alternative destination (Response LCF).

For Request LRJ, the possible tags are shown in Table 4-30:

***Table 4-30 Request LRJ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| s | Alias-Address | Optional | LRQ:srcInfo |
| d | Alias-Address | Required | LRQ:destinationInfo |
| e | IA5String | Optional | LRQ:nonStandardData:redirectNumber |
| E | integer | Optional | LRQ:nonStandardData:redirectReason |

*Table 4-30 Request LRJ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| p | integer | Optional | LRQ:nonStandardData:callingPartyNumOctet3a |
| w | IA5String | Optional | LRQ:nonStandardData:displayIE |
| c | IA5String | Optional | LRQ:nonStandardData:callingPartyNum |
| R | LRJ-reason | Required | LRJ:rejectReason |

## Response LRJ

This message is sent from the external application to the Cisco IOS Gatekeeper in response to a Request LRQ. The message indicates that the Cisco IOS Gatekeeper should reject the request for the specified reason.

This message can also be sent to the Cisco IOS Gatekeeper from the external application in response to a Request LCF or a Request LRJ. In the case of a Request LCF, this response rejects the information provided in the LCF for the specified reason. In the case of a Request LRJ, this response acknowledges the rejection. The reason is optional when the Response LRJ is sent due to a Request LRJ.

For Response LRJ, the possible tag is shown in Table 4-31:

*Table 4-31 Response LRJ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| R | LRJ-Reason | Required (LRQ, LCF)<br>Optional (LRJ) | LRJ:rejectReason |

Possible values for rejectReason are:

- notRegistered
- invalidPermission
- requestDenied
- undefinedReason
- securityDenial

# Disengage Messages

Disengage messages are used to indicate that a party wants to end the call.

This section describes the following:

- Request DRQ

## Request DRQ

This message is sent from the Cisco IOS Gatekeeper to the external application to indicate that an endpoint wants to end the call.

For Request DRQ, the possible tags are shown in Table 4-32:

***Table 4-32 Request DRQ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | GUID | Optional | DRQ:callIdentifier |
| C | GUID | Required | DRQ:conferenceID |
| R | DRQ-reason | Required | DRQ:disengageReason |
| A | Boolean | Required | DRQ:answeredCall |
| S | Transport-Address | Required | ARQ:srcCallSignalAddress |
| T | clearToken | Optional | DRQ:tokens |

Possible values for the DRQ-reason are:

- forcedDrop
- normalDrop
- undefinedReason

**Note** All Request DRQ messages must contain Notification-only in the header. No response to this message is sent.

# Resource Messages

Resource messages are used to indicate the current call capacity of the gateway.

This section describes the following:

- Request RAI

## Request RAI

This message is sent from the Cisco IOS Gatekeeper to the external application to indicate the call capacity and data rate of the gateway for H.323 calls.

For Request RAI, the possible tags are shown in Table 4-33:

***Table 4-33 Request RAI***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | Transport-Address | Required | RRQ:callSignalAddress |
| r | Boolean | Required | RAI:almostOutOfResources |

**Note** All Request RAI messages must contain Notification-only in the header. No response to this message is sent.

# Bandwidth Messages

Bandwidth messages are used to request a change in bandwidth.

This section describes the following:

- Request BRQ
- Response BCF
- Response BRJ

## Request BRQ

This message is sent from the Cisco IOS Gatekeeper to the external application to request that an endpoint be allowed to change (increase or decrease) its bandwidth.

For Request BRQ, the possible tags are shown inTable 4-34:

***Table 4-34 Request BRQ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| i | Transport-Address | Required | See Note |
| b | Bandwidth | Required | BRQ:bandWidth |
| C | GUID | Required | BRQ:conferenceID |
| c | GUID | Required | BRQ:callIdentifier |
| A | Boolean | Required | BRQ:answeredCall |

**Note** When sending a BRQ message, an endpoint identifies itself to the gatekeeper using the endpointIdentifier that it received from the gatekeeper in the RCF. Because this endpointIdentifier has only local significance to the gatekeeper and no significance to the server, the endpoint's CallSignalAddress is used here as an identifier.

## Response BCF

This message is sent from the external application to the Cisco IOS Gatekeeper to confirm the request to allow an endpoint to change (increase or decrease) its bandwidth. This response gives the external application the opportunity to modify the Bandwidth field of a received LCF, but because the Cisco IOS Gatekeeper is not prepared to make changes in its bandwidth, any change in the BCF will automatically generate a BRJ back to the endpoint.

For Response BCF, the possible tag is shown in Table 4-35:

*Table 4-35 Response BCF*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| b | Bandwidth | Required | BCF:bandWidth |

## Response BRJ

This message is sent from the external application the Cisco IOS Gatekeeper to deny the request to allow an endpoint to change (increase or decrease) its bandwidth.

For Response BRJ, the possible tag is shown in Table 4-36:

*Table 4-36 Response BRJ*

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| R | BRJ-Reason | Required | BRJ:rejectReason |

Possible values for rejectReason are:

- notBound
- invalidConferenceID
- invalidPermission
- insufficientResource
- invalidRevision
- undefinedReason
- securityDenial

# Progress Messages

Progress messages provide information about the progress of a request. Progress messages include:

- Response RIP
- Request IRR
- Request ALV
- Response ALV

## Response RIP

This message is sent from the external application to the Cisco IOS Gatekeeper when the external application cannot immediately process the request. This message indicates that the request is in progress (RIP) and that additional time is needed. When the Cisco IOS Gatekeeper receives this message, it forwards a request to the H.323 endpoint indicating that an extension of the timeout is required. The external application can send more that one Response RIP as is needed to process the request.

For Response RIP, the possible tag is shown in Table 4-37:

***Table 4-37 Response RIP***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Integer | Required | RIP:delay |

Possible values of the delay are 1 through 65535 milliseconds.

## Request IRR

This message is sent to the GK and contains details for the call after a successful connect. A Request IRR message is sent at both the originating and terminating side of the call. If both legs reference the same GK, only one Request IRR is sent. The GK sends information for only one call in each Request IRR message.

Table 4-38 shows the new Request IRR tags:

***Table 4-38 New Request IRR Tags***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| S | Transport-Address | Mandatory | IRR:srcCallSignalAddress |
| P | PerCallInfo | Optional | IRR:perCallInfo |

If the message contains a PerCallInfo field, the following additional fields shown in Table 4-39 are included:

***Table 4-39 New Additional Fields***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| c | GUID | Optional | IRR:perCallInfo:callIdentifier |
| C | GUID | Mandatory | IRR:perCallInfo:conferenceID |
| A | Boolean | Optional | IRR:perCallInfo:originator |
| b | Bandwidth | Mandatory | IRR:perCallInfo:bandwidth |
| t | System Time | Optional | IRR:perCallInfo:NonStandard:start_time |

## Request ALV

The REQUEST ALV is sent from the Gatekeeper to a GKTMP server on the detection of slower response or server failure.

This message does not contain any parameters in its body.

## Response ALV

This message is returned in response to a REQUEST ALV message and does not contain any parameters in its body.

# Trigger Registration Messages

Trigger registration messages are used by external applications to inform the Cisco IOS Gatekeeper which RAS messages are interesting to the external application. Interesting RAS messages trip a trigger in the Cisco IOS Gatekeeper and cause the Cisco IOS Gatekeeper to send a GKTMP RAS message to the external application.

As with the GKTMP RAS messages, trigger registration messages have the following format:

- Single message line
- One or more message header lines
- Blank line, which separates the message header from the message body
- Zero or more message body lines

## Message Line

There are two types of trigger registration messages: register and unregister.

The first line of each trigger registration request/response message uses the format:

`REGISTER` *`RAS_message_type`*

The first line of each trigger unregistration request/response message uses the format:

`UNREGISTER` *`RAS_message_type`*

Possible RAS message types are as follows:

- RRQ—Registration request
- URQ—Unregistration request
- ARQ—Admission request
- LRQ—Location request
- LCF—Location confirm
- LRJ—Location reject
- DRQ—Disengage request
- RAI—Resource availability information
- BRQ—Bandwidth request

## Message Header

The message line is immediately followed by the message header. Each message header contains a field name and a value, separated by a colon (*field*:*value*). Possible fields are shown in Table 4-40:

*Table 4-40 Message Header Fields*

| Field Names | Field Values |
|---|---|
| Version-ID | Version of the GKTMP. The version ID consists of a major number (gk_major) and a minor number (gk_minor). For example, Version 1 is represented as 100. |
| From | String that identifies the originator of the message. For trigger registration requests from the external application, this field contains the server ID. For trigger registration responses from the Cisco IOS Gatekeeper, this field contains the gatekeeper ID. This field is required for trigger registration and unregistration requests and responses. |
| To | String that identifies the receiver of the message. For trigger registration requests from the external application, this field contains the gatekeeper ID. For trigger registration responses from the Cisco IOS Gatekeeper, this field contains the ID of the external application that initiated the request. This field is required for trigger registration and unregistration requests and responses. |
| Priority | A number indicating the priority of this trigger in relation to other triggers for the same RAS message type. Possible values are 1 through 20. 1 is the highest priority.<br><br>If the Cisco IOS Gatekeeper has a registration for a RAS message type and receives another registration for the same RAS message from the same external application with the same priority, the Cisco IOS Gatekeeper uses the new registration and discards the previous one. If the Cisco IOS Gatekeeper has a registration for a RAS message type and receives another registration with the same priority from a different external application, the Cisco IOS Gatekeeper discards the new registration. This field is required for trigger registration and unregistration requests and is echoed in trigger registration and unregistration responses. |
| Content-length | The number of octets contained in the message body. If the message body is null, this field is omitted. This field is used only in trigger registration requests. |

*Table 4-40 Message Header Fields*

| Field Names | Field Values |
|---|---|
| Notification-only | None. No value is included after the colon. If this field name is present, it indicates to the Cisco IOS Gatekeeper that it should forward requests for the specified RAS messages as a notification only. This field is used only in trigger registration requests. |
| Status | String that indicates the response code from the Cisco IOS Gatekeeper. This field is used only in trigger registration and unregistration responses.<br>Possible response codes for unregistration requests are:<br>• success—The registration has been accepted.<br>• invalidPriority—The registration has been rejected because the Gatekeeper already has a registration for this RAS message type with the same priority from another application.<br>• invalidFilters—Parsing of the message body failed.<br>• invalidGKID—The gatekeeper ID specified in the "To" field of the request does not match the ID of any gatekeepers on this Cisco router.<br>Possible response codes for unregistration responses are:<br>• success—The unregistration has been accepted.<br>• invalidPriority—The unregistration has been rejected because the Gatekeeper does not have a registration for this RAS message type with the same priority from this application.<br>• invalidGKID—The gatekeeper ID specified in the "To" field of the request does not match the ID of any gatekeepers on this Cisco router. |

The message header is followed immediately by a blank line.

## Message Body

The message body follows the blank line. Only trigger registration requests contain a message body. Trigger registration responses, unregistration requests, and unregistration responses end after the blank line.

The message body in a trigger registration request can be used to narrow the circumstances under which the Cisco IOS Gatekeeper sends a REQUEST *xxx* to the external application. In this case, the external application includes tags and values in the message body that if matched will trigger the Cisco IOS Gatekeeper to generate a REQUEST *xxx*.

The tags that can be included vary depending on the RAS message type, and are a subset of the types that can be included in GKTMP RAS messages.

For the field type of Alias-Address, trailing wildcards can be used with E.164 addresses. An asterisk can be used to indicate a string of characters (for example, 1800*). A period can be used to indicate a single character (for example, 1800.....).

**Note** Wildcards cannot be used at the beginning or in the midst of a value, only at the end. If you include a wildcard at the beginning or in the midst of a value, it will be interpreted as a literal character.

## Register RRQ and RAI

For Register RRQ and RAI, the tags shown in Table 4-41 can be used to filter messages:

***Table 4-41 Register RRQ and RAI***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| t | Endpoint-Type | Optional | RRQ:terminalType |
| p | Supported-Prefix | Optional | RRQ:terminalType:gateway:protocol:*:supportedPrefixes |

## Register URQ

For Register URQ, the tags shown in Table 4-42 can be used to filter messages:

***Table 4-42 Register URQ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| t | Endpoint-Type | Optional | RRQ:terminalType |
| p | Supported-Prefix | Optional | RRQ:terminalType:gateway:protocol:*:supportedPrefixes |

## Register ARQ, DRQ, IRR, and BRQ

For Register ARQ, DRQ, IRR, and BRQ the tags shown in Table 4-43 can be used to filter messages:

***Table 4-43 Register ARQ, DRQ, IRR, and BRQ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | ARQ:destinationInfo |
| E | integer | Optional | ARQ:nonStandardData:redirectReason |

## Register LRQ

For Register LRQ, the tags shown in Table 4-44 can be used to filter messages:

***Table 4-44 Register LRQ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | LRQ:destinationInfo |
| E | integer | Optional | LRQ:nonStandardData:redirectReason |

**Note** A gatekeeper might not be the final destination of the LRQ messages that it receives. If the queried address in an LRQ is in another Gatekeeper's zone, the LRQ is forwarded to that gatekeeper and is not resolved locally. This means that there might not be a local zone that can be associated with the LRQ. To address this situation, the gatekeeper arbitrarily uses the server registrations for the first configured local zone. Because the order in which configured zones appear can change with deletions and additions, servers should send identical LRQ registrations to all zones (all logical gatekeepers) on the same router.

## Register LCF

For Register LCF, the tags shown in Table 4-45 can be used to filter messages:

***Table 4-45 Register LCF***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | LRQ/LCF:destinationInfo |
| X | Alias-Address | Optional | LCF:remoteExtensionAddress |

## Register LRJ

For Register LRJ, the tag shown in Table 4-46 can be used to filter messages:

***Table 4-46 Register LRJ***

| Tag | Field Type | Required or Optional | Corresponding RAS Message Field |
|---|---|---|---|
| d | Alias-Address | Optional | LRQ:destinationInfo |

CHAPTER 5

# Gatekeeper API Functions and Structures

This chapter describes the API functions and structures that an external application must use to exchange messages with the Cisco IOS Gatekeeper, and contains the following sections:



The external application links with the object code, which contains the API functions. The header file contains API prototypes and type definitions.

## Gatekeeper API Functions

This section describes the functions provided with the API. These functions should be used by the external application to gather information from and provide information to the Cisco IOS Gatekeeper. The functions described in this section are:

# GkapiSetupClient

This function sets up the socket for the application to communicate as a client with the Cisco IOS Gatekeeper. In this situation, the application is the client and the Gatekeeper is the server, which means the application must initiate the communication with the Cisco IOS Gatekeeper.

## Input

The input to this function is:

- A pointer to the GKAPI_SOCK_INFO structure. The application must set up the TCPPort and IPAddress fields and must preserve this structure for the duration of the connection.
- A pointer to the STATUS_TYPE enumeration. Possible values for STATUS_TYPE are:
  - PROCESSING_SUCCESSFUL—Successful connection to the Cisco IOS Gatekeeper.
  - CONNECT_IN_PROGRESS—Connection is pending.
  - TCP_HANDLE_ERROR—Error was encountered in handle creation.
  - TCP_CONNECT_ERROR—Error was encountered in connecting to the Cisco IOS Gatekeeper.
  - TCP_NONBLOCK_ERROR—Error was encountered when setting up the socket for nonblocking I/O
- A boolean value that allows the application to specify if the socket I/O should be nonblocking or blocking. If the application specifies blocking, the Gatekeeper API calls to setup the connection and read a message that does not return until the action is complete.

## Return

The return for this function is an integer. If the client socket connection has been set up successfully or is in progress, a connection handle is returned. This connection handle is the socket descriptor that the application uses to wait on a connection completion or read socket event. If an error occurs while setting up the client connection, the value -1 is returned. In this case, the error information is provided in the STATUS_TYPE.

# GkapiSetupServer

This function sets up the socket for the application to communicate as a server with the Cisco IOS Gatekeeper. In this situation, the application is the server and the Gatekeeper is the client, which means that the application will accept incoming connections from Cisco IOS Gatekeeper clients.

## Input

- A pointer to the GKAPI_SOCK_INFO structure. The application must set up the TCPPort and IPAddress fields and must preserve this structure for the duration of the connection.
- A pointer to the STATUS_TYPE enumeration. Possible values for STATUS_TYPE are:
  - PROCESSING_SUCCESSFUL—Successful connection to the Cisco IOS Gatekeeper.
  - TCP_HANDLE_ERROR—Error was encountered in handle creation.
  - TCP_ADDRESS_ALREADY_IN_USE—Specified local IP address is already in use.

  - TCP_ADDRESS_NOT_AVAIL—Specified local IP address is not available on the local machine.
  - TCP_BIND_ERROR—Error was encountered in setting up the server socket.
  - TCP_LISTEN_ERROR—Error was encountered in setting up the server socket.
  - TCP_NONBLOCK_ERROR—Error was encountered when setting up the socket for nonblocking I/O.
- A boolean value that allows the application to specify if the socket I/O should be nonblocking or blocking. If the application specifies blocking, the Gatekeeper API calls to setup the connection and read a message that does not return until the action is complete.

### Return

The return for this function is an integer. If the client socket connection has been set up successfully or is in progress, a connection handle is returned. This connection handle is the socket descriptor that the application uses to wait on a connection completion or read socket event. If an error occurs while setting up the client connection, the value -1 is returned. In this case, the error information is provided in the STATUS_TYPE.

## GkapiClientConnected

This function must be called by the application to indicate that a select event for a connect complete has occurred.

### Input

The input to this function is:

- A pointer to the GKAPI_SOCK_INFO structure.
- A pointer to the STATUS_TYPE enumeration. Possible values for STATUS_TYPE are:
  - PROCESSING_SUCCESSFUL—Successful connection to the Gatekeeper.
  - TCP_CONNECT_ERROR—Error was encountered in connecting to the Gatekeeper.
- An integer that indicates that a connect complete has occurred.

### Return

The return for this function is an integer. If the socket connection has been set up successfully or is in progress, a connection handle is returned. This connection handle is the socket descriptor that the application uses to wait on a connection completion or read socket event. If an error occurs while setting up the client connection, the value -1 is returned. In this case, the error information is provided in the STATUS_TYPE.

# GkapiAcceptConnection

This function must be called by the application (when it is running in the server mode) to indicate that a select event for an incoming connection has occurred.

## Input

The input to this function is

- A pointer to the GKAPI_SOCK_INFO structure.
- A pointer to the STATUS_TYPE enumeration. Possible values for STATUS_TYPE are:
  - PROCESSING_SUCCESSFUL—Successful connection to the Cisco IOS Gatekeeper.
  - TCP_CONNECT_ERROR—Error was encountered in connecting to the Cisco IOS Gatekeeper.
- An integer that indicates that an incoming connection has occurred.
- A pointer to the GKAPI_TCP_ADDR_INFO structure. The Gatekeeper API provides the IP address and TCP port of the client with which this connection is associated.

## Return

The return for this function is an integer. If the socket connection has been set up successfully or is in progress, a connection handle is returned. This connection handle is the socket descriptor that the application uses to wait on a connection completion or read socket event. If an error occurs while setting up the client connection, the value -1 is returned. In this case, the error information is provided in the STATUS_TYPE.

# GkapiGetVersion

Applications can use this function to obtain the GKTMP version used by the API and the gatekeeper. The version number consists of a major number (gk_major) and a minor number (gk_minor). For example, Version 1 is represented as 100.

## Input

The input to this function is

- A pointer to the GKAPI_VERSION_INFO structure.

## Return

The return for this function is an integer. The integer indicates one of the following status codes:

- GKAPI_RET_OK—Gatekeeper TMP version is prior to or the same as the gatekeeper API version.
- GKAPI_RET_NOK—Gatekeeper TMP version is later than the gatekeeper API version.
- GKAPI_RET_OK_NOGKDATA—Version used by the gatekeeper is not known. In this case, the gk_major and gk_minor members of GKAPI_VERSION_INFO_T are invalid and set to -1.

# CloseGateKeeperConnection

This function closes the TCP connection between the external application and the Cisco IOS Gatekeeper. This function is called under error circumstances and when the external application no longer wants to maintain a relationship with the Cisco IOS Gatekeeper.

## Input

The input for this function is a pointer to the GKAPI_SOCK_INFO structure.

## Return

There is no return for this function.

# GetReadMsgBuffer

This function allocates memory for the size of GK_READ_MSG structure. This structure is used to store messages received from the Cisco IOS Gatekeeper. This function contains an enumeration of the messages that can be received (REQUEST messages from the Cisco IOS Gatekeeper for RRQ, ARQ, LRQ, LCF, LRJ, as well as registration and unregistration responses from the Cisco IOS Gatekeeper for ARQ, RRQ, URQ, LRQ, LCF, LRJ messages) and a union of structures for the different messages.

**Note** When the external application no longer needs the message buffer, the application must call FreeReadMsgBuffer to release the memory back to the system.

## Input

There is no input to this function.

## Return

The return for this function is a pointer to the GK_READ_MSG structure. If the memory allocation fails, this pointer will be NULL.

# ReadMsgBuffer

This function reads a message from the TCP socket and should be called when the external application has detected a read event on the socket. This function stores the message type into the structure. The parameters received in the message are stored in the structure that corresponds with the message type.

**Note** GetReadMsgBuffer must be called to allocate an empty buffer before this function can be used. FreeReadMsgBuffer must be called after this function has completed, except when the STATUS_TYPE returns INCOMPLETE_MSG_READ.

After reading a message, this function sets the message type and populates the appropriate structure. For example, if an ARQ message has been received from the Cisco IOS Gatekeeper, the msgType parameter is set to ARQ_REQUEST_MSG and the ARQ_REQUEST_MSG structure is populated.

Because some parameters are optional, these parameters might not be received for a particular message. Structure members that are character pointers are initialized to NULL. Integers and enumerations are set to their initialization values. Therefore, the API can assume that if a structure member has a pointer set to NULL or to its initialization value, that particular parameter has not been received.

The following initialization values indicate that the parameter was not received from the Cisco IOS Gatekeeper:

- canMapAlias—INITIALIZE_CAN_MAP_ALIAS_VALUE
- bandWidthPresent—TRUE (indicating that bandWidth has been received and filled in) or FALSE (indicating that bandWidth has not been received)
- answerCall—INITIALIZE_ANSWER_CALL_VALUE
- REDIRECT_REASON_TYPE—REDIRECT_REASON_INFO_NOT_RCVD
- ENDPOINT_TYPE—ENDPOINT_INFO_NOT_RCVD

## Input

The input for this function is:

- A pointer to the GKAPI_SOCK_INFO structure.
- A pointer to the GK_READ_MSG structure that was allocated by the GetReadMsgBuffer function. The GK_READ_MSG structure contains an enumeration of the message types expected from the Cisco IOS Gatekeeper and a union of structures for various messages expected from the Cisco IOS Gatekeeper.

## Return

The return for this function is the STATUS_TYPE. Possible values for STATUS_TYPE are:

- PROCESSING_SUCCESSFUL—No errors were encountered.
- TCP_READ_ERROR—A TCP read error was encountered. The application should call CloseGateKeeperConnection to close the connection to the Cisco IOS Gatekeeper.
- MEM_ALLOC_FAIL—Memory allocation failed. This function, dynamically allocates memory for fields within the GK_READ_MSG structure.
- MSG_READ_ERROR—The message read was not understood by the API function. The application should call CloseGateKeeperConnection to close the connection to the Cisco IOS Gatekeeper.
- INCOMPLETE_MSG_READ—The message was not completely read from the TCP connection because of network conditions. The application should call the function again in order to continue reading the data. In this situation, FreeMsgBuffer should not be called. After all the data has been read, the STATUS_TYPE is set to one of the other possible values, and after processing the message type the FreeMsgBuffer can be called.
- TCP_CONNECTION_CLOSED—The connection to the Cisco IOS Gatekeeper has been closed. The application must call CloseGateKeeperConnection to free resources such as gkHandle in the GKAPI_SOCK_INFO structure.
- NULL_POINTER_PASSED—The pointer to the GK_READ_MSG is null.

# FreeReadMsgBuffer

This function frees memory that was allocated by the call to GetReadMsgBuffer and ReadMsgBuffer. This function **must** be called after processing the information returned by ReadMsgBuffer.

## Input

The input for this function is a pointer to the GK_READ_MSG structure.

## Return

There is no return for this function.

# WriteResponseMsg

This function writes a response message to the Cisco IOS Gatekeeper. This structure contains RESPONSE_MSG_TYPE, which is an enumeration of the response messages that can be sent to the Cisco IOS Gatekeeper.

The calling function must set the message type and populate the appropriate structure within the union. For example, if a response RCF needs to be sent to the Cisco IOS Gatekeeper, the application should set the msgType to RCF_RESPONSE_MSG and populate the RCF_RESPONSE_MSG structure.

The following rules apply to responses sent by the external application to the Cisco IOS Gatekeeper:

- Transport-addresses must be preceded with “I:”, followed by the address.
- Alias-addresses must be preceded with either “H:”, “E:”, or “M:” followed by the alias address.
- Values in a “sequence of values” must be separated by a space.
- HEADER_INFO must include the “from”, “to” and “transactionID” fields. The notification field is not used with the WriteResponseMsg function.

Only changed or new fields should be populated and sent to the Cisco IOS Gatekeeper. Parameters that are not to be sent to the Cisco IOS Gatekeeper must either be set to their initialization value or to NULL (for pointers). The API assumes that if a structure member is set to its initialization value or has a pointer set to NULL, that parameter should not be sent to the Cisco IOS Gatekeeper.

The following initialization values indicate that the parameter should not be sent to the Cisco IOS Gatekeeper:

- bandWidthPresent—TRUE (indicating that the bandWidth should be sent) or FALSE (indicating that the bandWidth should not be sent)
- REDIRECT_REASON_TYPE—REDIRECT_REASON_INFO_NOT_RCVD
- ENDPOINT_TYPE—ENDPOINT_INFO_NOT_RCVD

**Note** If the application requires additional time before responding to a message from the Cisco IOS Gatekeeper, the application can send a “delay” message by setting msgType to RIP_RESPONSE_MSG. The delay value (1 through 65536) must be specified and the transactionID must be the same as the one received from the Cisco IOS Gatekeeper.

## Input

The input for this function is:

- A pointer to the GKAPI_SOCK_INFO structure.
- A pointer to the IRR_REQUEST_MSG structure, which contains an enumeration of message types for which a response might be sent to the Cisco IOS Gatekeeper. The input also contains a union of structures for each message response.

## Return

The return for this function is the STATUS_TYPE. Possible values for STATUS_TYPE are:

- PROCESSING_SUCCESSFUL—No errors were encountered.
- CONNECT_IN_PROGRESS—Connection is pending. The application should retry this API call after some time has passed.
- TCP_WRITE_ERROR—A TCP write error was encountered. The application should call CloseGateKeeperConnection to close the connection to the Cisco IOS Gatekeeper.
- MEM_ALLOC_FAIL—Memory allocation failed.
- TCP_CONNECTION_CLOSED—The connection to the Cisco IOS Gatekeeper has been closed. The application must call CloseGateKeeperConnection to free resources such as gkHandle in the GKAPI_SOCK_INFO structure.
- INVALID_MSG_SPECIFIED—The message type is not within the RESPONSE_MSG_TYPE range.
- INVALID_ENDPOINT_SPECIFIED—The endpoint does not match one of the possible values for ENDPOINT_TYPE.
- INVALID_REDIRECT_REASON_SPECIFIED—The redirect reason does not match one of the possible values for REDIRECT_REASON_TYPE.
- INVALID_REJECT_REASON_SPECIFIED—The rejection reason does not match one of the possible values for REJECT_REASON_TYPE.
- INVALID_DELAY_SPECIFIED—The delay is not within the valid range.
- HEADER_INFO_INCOMPLETE—One of the fields in the header (To, From, TransactionID) is incomplete.
- NULL_POINTER_PASSED—The pointer to GK_WRITE_MSG is null.

# WriteRegisterMessage

This function sends a registration message to the Cisco IOS Gatekeeper and allows triggers to be dynamically registered with the Cisco IOS Gatekeeper. This structure, REGISTER_MSG_TYPE, contains an enumeration of messages that can be registered with the Cisco IOS Gatekeeper.

The REGISTER_REQUEST_HEADER structure must include the “from,” “to,” “priority,” and “notification-only” fields.

## Input

The input for this function is:

- A pointer to the GKAPI_SOCK_INFO structure.
- A pointer to the GK_REGISTER_MSG structure, which contains a union of the structures for the various registration messages that can be sent to the Cisco IOS Gatekeeper. Each structure contains a header, REGISTER_REQUEST_HEADER, that must be filled in by the application. The msgType field must be filled in to indicate which registration message should be sent to the Cisco IOS Gatekeeper.

### Return

The return for this function is the STATUS_TYPE. Possible values for STATUS_TYPE are:

- PROCESSING_SUCCESSFUL—No errors were encountered.
- CONNECT_IN_PROGRESS—Connection is pending. The application should retry this API call after some time has passed.
- TCP_WRITE_ERROR—A TCP write error was encountered. The application should call CloseGateKeeperConnection to close the connection to the Cisco IOS Gatekeeper.
- MEM_ALLOC_FAIL—Memory allocation failed.
- TCP_CONNECTION_CLOSED—The connection to the Cisco IOS Gatekeeper has been closed. The application must call CloseGateKeeperConnection to free resources such as gkHandle in the GKAPI_SOCK_INFO structure.
- INVALID_MSG_SPECIFIED—The message type is not within the RESPONSE_MSG_TYPE range.
- INVALID_ENDPOINT_SPECIFIED—The endpoint does not match one of the possible values for ENDPOINT_TYPE.
- INVALID_REDIRECT_REASON_SPECIFIED—The redirect reason does not match one of the possible values for REDIRECT_REASON_TYPE.
- HEADER_INFO_INCOMPLETE—One of the fields in the header (To, From, TransactionID) is incomplete.
- NULL_POINTER_PASSED—The pointer to the GK_REGISTER_MSG is null.

The following initialization values indicate that the parameter should not be sent to the Cisco IOS Gatekeeper for Registration, and the external application is not interested in these parameters:

- REDIRECT_REASON_TYPE - REDIRECT_REASON_INFO_NOT_RCVD
- ENDPOINT_TYPE - ENDPOINT_INFO_NOT_RCVD

## WriteUnregisterMessage

This function sends an unregister message to the Cisco IOS Gatekeeper when the application no longer wants to receive a particular message. This structure contains REGISTER_MSG_TYPE, which is an enumeration of messages that can be unregistered with the Cisco IOS Gatekeeper.

### Input

The input for this function is:

- A pointer to the GKAPI_SOCK_INFO structure.

- A pointer to the GK_UNREGISTER_MSG structure, which contains the To, From, and Priority fields that must be filled in by the application. The msgType must be filled in to indicate which message needs to be unregistered.

### Return

The return for this function is the STATUS_TYPE. Possible values for STATUS_TYPE are:

- PROCESSING_SUCCESSFUL—No errors were encountered.
- CONNECT_IN_PROGRESS—Connection is pending. The application should retry this API call after some time has passed.
- TCP_WRITE_ERROR—A TCP write error was encountered. The application should call CloseGateKeeperConnection to close the connection to the Cisco IOS Gatekeeper.
- MEM_ALLOC_FAIL—Memory allocation failed.
- TCP_CONNECTION_CLOSED—The connection to the Cisco IOS Gatekeeper has been closed. The application must call CloseGateKeeperConnection to free resources such as gkHandle in the GKAPI_SOCK_INFO structure.
- INVALID_MSG_SPECIFIED—The message type is not within the RESPONSE_MSG_TYPE range.
- HEADER_INFO_INCOMPLETE—One of the fields in the header (To, From, TransactionID) is incomplete.
- NULL_POINTER_PASSED—The pointer to the GK_UNREGISTER_MSG is null.

## GkapiSetupReport

This function allows the application to control the type of debug messages that the Gatekeeper API provides and the location of the debug output.

### Input

The input for this function is:

- An integer that indicates the type of debugging. If the debugging is set to 0, the Gatekeeper API will not output any debug messages.
- A pointer to the REPORT_DEST_T enumeration, which indicates the destination for the debug messages.

### Return

There is no return for this function.

## GkapiQueryReport

This function returns the current debug setting for the Gatekeeper API.

### Input

There is no input for this function.

### Return

The return for this function is an integer that indicates the type of debugging being performed by the Gatekeeper API.

# API Structures

The Gatekeeper API stores all data received from the Cisco IOS Gatekeeper in structures. The structures point to character strings, integers, and often enumerations (which are lists of possible values for a specific field). The structures used by the Gatekeeper API are:

- GKAPI_SOCK_INFO
- GKAPI_TCP_ADDR_INFO
- GKAPI_VERSION_INFO
- GK_REGISTER_MSG
- GK_UNREGISTER_MSG
- REG_UNREG_RESP_MSG
- REGISTER_REQUEST_HEADER
- REGISTER_RESPONSE_HEADER
- ARQ_REGISTER_MSG
- RRQ_REGISTER_MSG
- URQ_REGISTER_MSG
- LRQ_REGISTER_MSG
- LCF_REGISTER_MSG
- LRJ_REGISTER_MSG
- RAI_REGISTER_MSG
- DRQ_REGISTER_MSG
- BRQ_REGISTER_MSG
- GK_READ_MSG
- HEADER_INFO
- ARQ_REQUEST_MSG
- RRQ_REQUEST_MSG
- URQ_REQUEST_MSG

- LRQ_REQUEST_MSG
- LCF_REQUEST_MSG
- LRJ_REQUEST_MSG
- RAI_REQUEST_MSG
- DRQ_REQUEST_MSG
- BRQ_REQUEST_MSG
- IRR_REQUEST_MSG
- ARQ_RESPONSE_MSG
- ACF_RESPONSE_MSG
- ARJ_RESPONSE_MSG
- RRQ_RESPONSE_MSG
- RCF_RESPONSE_MSG
- RRJ_RESPONSE_MSG
- LRQ_RESPONSE_MSG
- LCF_RESPONSE_MSG
- LRJ_RESPONSE_MSG
- BRQ_RESPONSE_MSG
- BCF_RESPONSE_MSG
- BRJ_RESPONSE_MSG
- ALV_RESPONSE_MSG
- CRYPTO_EP_PWD_HASH
- CRYPTO_EP_PWD_ENCR
- CRYPTO_EP_CERT
- CLEAR_TOKEN
- ALTERNATE_GK
- ALTERNATE_ENDPOINT
- ALTERNATE_TRANSPORT_ADDR_TYPE
- RIP_RESPONSE_MSG
- UNSUPPORTED_MSG

# GKAPI_SOCK_INFO

The GKAPI_SOCK_INFO structure is used by several API functions to identify the connection to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-1:

***Table 5-1 GKAPI_SOCK_INFO***

| Field | Field Type | Description |
|---|---|---|
| TCPPort | Integer | The TCP port of the Cisco IOS Gatekeeper that is establishing the incoming connection to the application. |
| IPAddress | Character string | The IP address of the Cisco IOS Gatekeeper that is establishing the incoming connection to the application. |
| gkHandle | Integer | Handle to the Cisco IOS Gatekeeper function. |
| serverHandle | Integer | Handle to the server function. |

TCPPort and IPAddress are provided by the calling function. The API writes the handle into gkHandle and serverHandle when the connection is established. If an error is encountered in the handle creation or in the connection, the gkHandle will be set to -1. The external application is responsible for storing the handle and using it to read, write, and close the connection.

# GKAPI_TCP_ADDR_INFO

The GKAPI_TCP_ADDR_INFO structure is used to store the TCP Port and IP address. This structure contains the fields shown in Table 5-2:

***Table 5-2 GKAPI_TCP_ADDR_INFO***

| Field | Field Type | Description |
|---|---|---|
| TCPPort | Integer | The TCP port that the Cisco IOS Gatekeeper uses for handling GKTMP messages. For GkapiSetupServer, this is the TCP port that the application uses for interacting with the Gatekeeper. |
| IPAddress | Unsigned long | For GkapiSetupClient, this is the IP address that the Cisco IOS Gatekeeper uses for handling GKTMP messages. For GkapiSetupServer, this is the IP address that the application uses for interacting with the Gatekeeper. |

# GKAPI_VERSION_INFO

The GKAPI_VERSION_INFO structure is used to store the major and minor version numbers of the gatekeeper TMP and API. This structure contains the fields shown in Table 5-3:

***Table 5-3 GKAPI_VERSION_INFO***

| Field | Field Type | Description |
|---|---|---|
| gkapi_major | Integer | The major number identifying the version of the API. |
| gkapi_minor | Integer | The minor number identifying the version of the API. |
| gktmp_major | Integer | The major number identifying the version of the TMP. |
| gktmp_minor | Integer | The minor number identifying the version of the TMP. |
| GKAPI_MAX_VER_STR_LEN | Character string | The build date and target operating system of the protocol. |
| gkapi_release_num | Integer | The release number of the gatekeeper API. |

# GK_REGISTER_MSG

The GK_REGISTER_MSG structure is used to send registration messages to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-4:

***Table 5-4 GK_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| msgType | Enumeration | See REGISTER_MSG_TYPE. |
| rrqRegMsg | Structure | See RRQ_REGISTER_MSG. |
| urqRegMsg | Structure | See URQ_REGISTER_MSG. |
| arqRegMsg | Structure | See ARQ_REGISTER_MSG. |
| lrqRegMsg | Structure | See LRQ_REGISTER_MSG. |
| lcfRegMsg | Structure | See LCF_REGISTER_MSG. |
| lrjRegMsg | Structure | See LRJ_REGISTER_MSG. |

# GK_UNREGISTER_MSG

The GK_UNREGISTER_MSG structure is used to send unregistration messages to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-5:

***Table 5-5 GK_UNREGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| unregisterMsg | Enumeration | See REGISTER_MSG_TYPE. |
| versionId | Integer | Identifier of the version of GKTMP being used. For the initial release, the only possible value is 1. |

*Table 5-5 GK_UNREGISTER_MSG*

| Field | Field Type | Description |
|---|---|---|
| from | Character string | Originator of the message. For requests from the Cisco IOS Gatekeeper, this field contains the gatekeeper ID. For responses from the external application, this field contains the server ID. The limit of this field is MAX_ENDPOINT_LENGTH + 1. |
| to | Character string | Receiver of the message. For requests from the Cisco IOS Gatekeeper, this field contains the server ID. For responses from the external application, this field contains the ID of the gatekeeper that initiated the request. The limit of this field is MAX_ENDPOINT_LENGTH + 1. |
| priority | Integer | Priority of the filter. Possible values are 1 through 20. 1 is the highest priority. |

# REG_UNREG_RESP_MSG

The REG_UNREG_RESP_MSG structure is used to process registration and unregistration responses from the Cisco IOS Gatekeeper. This structure contains the field shown in Table 5-6:

*Table 5-6 REG_UNREG_RESP_MSG*

| Field | Field Type | Description |
|---|---|---|
| regHeader | Structure | See REGISTER_RESPONSE_HEADER. |

# REGISTER_REQUEST_HEADER

The REGISTER_REQUEST_HEADER structure is used when a registration request is to be sent to Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-7:

*Table 5-7 REGISTER_REQUEST_HEADER*

| Field | Field Type | Description |
|---|---|---|
| versionId | Integer | Identifier of the version of GKTMP being used. For the initial release, the only possible value is 1. |
| from | Character string | Originator of the message, which for registration requests is the server ID. The limit of this field is MAX_ENDPOINT_LENGTH+1. |
| to | Character string | Receiver of the message, which for registration requests is the gatekeeper ID The limit of this field is MAX_ENDPOINT_LENGTH+1. |
| priority | Integer | Priority of the filter. Possible values are 1 through 20. 1 is the highest priority. |
| notificationOnly | Boolean | Whether the registration request is for notifications only. If this field is set to True, messages that match the specified trigger parameters are sent on a notification-only basis. |

# REGISTER_RESPONSE_HEADER

The REGISTER_RESPONSE_HEADER structure is used when a registration or unregistration response is received from the Cisco IOS Gatekeeper. The registration or unregistration response is received after the application sends a registration or unregistration request to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-8:

***Table 5-8 REGISTER_RESPONSE_HEADER***

| Field | Field Type | Description |
|---|---|---|
| version-id | Integer | Identifier of the version of GKTMP being used. For the initial release, the only possible value is 1. |
| from | Character string | Originator of the message, which for registration responses is the gatekeeper ID. The limit of this field is MAX_ENDPOINT_LENGTH+1. |
| to | Character string | Receiver of the message, which for registration responses is the server ID. The limit of this field is MAX_ENDPOINT_LENGTH+1. |
| priority | Integer | Priority of the filter. Possible values are 1 through 20. 1 is the highest priority. |
| regStatus | Enumeration | See REG_STATUS_TYPE. |

# ARQ_REGISTER_MSG

The ARQ_REGISTER_MSG structure is used to send registrations for ARQ requests to the Cisco IOS Gatekeeper.

This structure contains the fields shown in Table 5-9:

***Table 5-9 ARQ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_ARQ_DEST_INFO. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. The limit of this field is MAX_NUM_ARQ_REDIRECT_REASON. |

# RRQ_REGISTER_MSG

The RRQ_REGISTER_MSG structure is used to send registrations for RRQ requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-10:

***Table 5-10 RRQ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| terminalType | Enumeration | Type of endpoint being registered. See ENDPOINT_TYPE. The limit of this field is MAX_NUM_ENDPOINT_TYPES. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. The limit of this field is MAX_NUM_SUPPORTED_PREFIX. |

# URQ_REGISTER_MSG

The URQ_REGISTER_MSG structure is used to send registrations for URQ requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-11:

***Table 5-11 URQ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| terminalType | Enumeration | Type of endpoint being unregistered. See ENDPOINT_TYPE. The limit of this field is MAX_NUM_ENDPOINT_TYPES. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. The limit of this field is MAX_NUM_SUPPORTED_PREFIX. |

# LRQ_REGISTER_MSG

The LRQ_REGISTER_MSG structure is used to send registrations for LRQ requests to the Cisco IOS Gatekeeper.

This structure contains the fields shown in Table 5-12:

***Table 5-12 LRQ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_LRQ_DEST_INFO. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. The limit of this field is MAX_NUM_LRQ_REDIRECT_REASON. |

# LCF_REGISTER_MSG

The LCF_REGISTER_MSG structure is used to send registrations for LCF requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-13:

***Table 5-13 LCF_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_LCF_DEST_INFO. |
| rmotExtensionAddr | Character String | Alias address of a called endpoint, present in cases where this information is required to traverse multiple gateways. The limit of this field is MAX_NUM_LCF_RMOT_EXTENSION_ADDR. |

# LRJ_REGISTER_MSG

The LRJ_REGISTER_MSG structure is used to send registrations for LRJ requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-14:

***Table 5-14 LRJ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_LRJ_DEST_INFO. |

# RAI_REGISTER_MSG

The RAI_REGISTER_MSG structure is used to send registrations for RAI requests to the Cisco IOS Gatekeeper.

This structure contains the fields shown in Table 5-15:

***Table 5-15 RAI_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| terminalType | Enumeration | Type of endpoint. See ENDPOINT_TYPE. The limit of this field is MAX_NUM_ENDPOINT_TYPES. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. The limit of this field is MAX_NUM_SUPPORTED_PREFIX. |

# DRQ_REGISTER_MSG

The DRQ_REGISTER_MSG structure is used to send registrations for DRQ requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-16:

***Table 5-16 RAI_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_ARQ_DEST_INFO. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. The limit of this field is MAX_NUM_LRQ_REDIRECT_REASON. |

# BRQ_REGISTER_MSG

The BRQ_REGISTER_MSG structure is used to send registrations for BRQ requests to the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-17:

***Table 5-17 BRQ_REGISTER_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. The limit of this field is MAX_NUM_ARQ_DEST_INFO. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. The limit of this field is MAX_NUM_ARQ_REDIRECT_REASON. |

# GK_READ_MSG

The GK_READ_MSG structure is used process REQUEST messages from the Cisco IOS Gatekeeper for the supported RAS messages, as well as registration and unregistration responses from the Cisco IOS Gatekeeper for the supported RAS messages. This structure contains the fields shown in Table 5-18:

***Table 5-18 GK_READ_MSG***

| Field | Field Type | Description |
|---|---|---|
| msgType | Enumeration | See REQUEST_MSG_TYPE. |
| rrqReqMsg | Structure | See RRQ_REQUEST_MSG. |
| urqReqMsg | Structure | See URQ_REQUEST_MSG. |
| arqReqMsg | Structure | See ARQ_REQUEST_MSG. |

*Table 5-18 GK_READ_MSG*

| Field | Field Type | Description |
|---|---|---|
| lrqReqMsg | Structure | See LRQ_REQUEST_MSG. |
| lcfReqMsg | Structure | See LCF_REQUEST_MSG. |
| lrjReqMsg | Structure | See LRJ_REQUEST_MSG. |
| raireqMsg | Structure | See RAI_REQUEST_MSG. |
| drqreqMsg | Structure | See DRQ_REQUEST_MSG. |
| brqreqMsg | Structure | See BRQ_REQUEST_MSG. |
| unsupportedMsg | Structure | See UNSUPPORTED_MSG. |
| regUnregRespMsg | Structure | See REG_UNREG_RESP_MSG. |

If the message received from the Cisco IOS Gatekeeper is a RAS message that is not supported by the API function, the msgType is set to MSG_NOT_SUPPORTED. If a response is required, an appropriate response is constructed by the API function and sent to the Cisco IOS Gatekeeper. The header information in the UNSUPPORTED_MSG structure is filled in by the API function. This situation could occur if the Cisco IOS Gatekeeper has been upgraded to support new messages but the API function has not been correspondingly upgraded.

If the message received from the Cisco IOS Gatekeeper, is not recognized by the API function, the msgType is set to UNKNOWN_MSG and the STATUS_TYPE is set to MSG_READ_ERROR. In this case, the external application should close the connection to the Cisco IOS Gatekeeper by calling the CloseGateKeeperConnection function.

# HEADER_INFO

The HEADER_INFO structure is used to process header information sent from the Cisco IOS Gatekeeper or information that is sent by the application to the Cisco IOS Gatekeeper.

This structure contains the fields shown in Table 5-19:

*Table 5-19 HEADER_INFO*

| Field | Field Type | Description |
|---|---|---|
| versionId | Integer | Identifier of the version of GKTMP being used. For the initial release, the only possible value is 1. |
| from | Character string | Originator of the message. For requests from the Cisco IOS Gatekeeper, this field contains the gatekeeper ID. For responses from the external application, this field contains the server ID. The limit of this field is MAX_ENDPOINT_LENGTH + 1. |
| to | Character string | Receiver of the message. For requests from the Cisco IOS Gatekeeper, this field contains the server ID. For responses from the external application, this field contains the ID of the gatekeeper that initiated the request. The limit of this field is MAX_ENDPOINT_LENGTH+1. |

*Table 5-19 HEADER_INFO*

| Field | Field Type | Description |
|---|---|---|
| transactionID | Character string | Identifier of the transaction. If this field is present in the request from the Cisco IOS Gatekeeper, it must be echoed in the response from the external application. The limit of this field is MAX_TRANSACTION_ID_LENGTH + 1. |
| notification | Boolean | Whether the message is for notification purposes only. This field is used only in REQUEST messages that are received from the Cisco IOS Gatekeeper. |

# ARQ_REQUEST_MSG

The ARQ_REQUEST_MSG structure is used to process ARQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-20:

*Table 5-20 ARQ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| srcInfo | Character string | Sequence of alias addresses for the source endpoint. |
| srcCallSignalAddress | Character string | Transport address used at the source for call signaling. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| destCallSignalAddress | Character string | Transport address used at the destination for call signaling. |
| destExtraCallInfo | Character string | External addresses for multiple calls. |
| bandWidthPresent | Boolean | Whether a specified bandwidth is present in the request. |
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bi-directional call. |
| answerCall | Integer | Indicates to the Cisco IOS Gatekeeper that the call is incoming. |
| callIdentifier | Character string | A unique call identifier (set by the originating endpoint), which can be used to associate RAS signaling with the modified Q.931 signaling used in H.225.0. |
| conferenceID | Character string | A unique conference identifier. |
| canMapAlias | Integer | Whether the endpoint can copy information from the resulting ACF into the destinationAddress, destExtraCallInfo, and remoteExtensionAddress fields of the SETUP message. |
| redirectNumber | Character string | Taken from the Number Digits field of Q.931 Setup Redirecting Number IE. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. |
| callingOctet3a | Character String | Whether the calling number information can be displayed. |

*Table 5-20 ARQ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| displayIE | Character String | Taken from the Q.931 Setup, display IE. |
| endPointCallSignalAddress | Character String | Call signaling transport address of the endpoint sending the ARQ. |
| cryptoToken | Pointer | See ALV_RESPONSE_MSG. |
| clearToken | Pointer | See CLEAR_TOKEN. |

# RRQ_REQUEST_MSG

The RRQ_REQUEST_MSG structure is used to process RRQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-21:

*Table 5-21 RRQ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| callSignalAddress | Character string | Call signaling transport address for this endpoint. |
| rasAddress | Character string | Registration and status transport address for this endpoint. |
| terminalAlias | Character string | List of alias addresses by which other terminals can identify this terminal. |
| terminalType | Enumeration | Type of endpoint being registered. See ENDPOINT_TYPE. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. |
| cryptoToken | Pointer | See ALV_RESPONSE_MSG. |
| clearToken | Pointer | See CLEAR_TOKEN. |
| altTranspAddr | Pointer | See ALTERNATE_TRANSPORT_ADDR_TYPE. |

# URQ_REQUEST_MSG

The URQ_REQUEST_MSG structure is used to process URQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-22:

*Table 5-22 URQ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| callSignalAddress | Character string | Call signaling transport address for this endpoint. |

# LRQ_REQUEST_MSG

The LRQ_REQUEST_MSG structure is used to process LRQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-23:

***Table 5-23 LRQ_REQUEST_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| srcInfo | Character string | Sequence of alias addresses for the source endpoint. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| redirectNumber | Character string | Taken from the Number Digits field of Q.931 Setup Redirecting Number IE. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. |
| callingOctet3a | Character String | Whether the calling number information can be displayed. |
| displayIE | Character String | Taken from the Q.931 Setup, display IE. |
| callingPartyNum | Character String | Taken from the Q.931. |

# LCF_REQUEST_MSG

The LCF_REQUEST_MSG structure is used to process LCF requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-24:

***Table 5-24 LCF_REQUEST_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| srcInfo | Character string | Sequence of alias addresses for the source endpoint. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| callSignalAddress | Character string | Call signaling transport address for this endpoint. |
| destExtraCallInfo | Character string | External addresses for multiple calls. |
| redirectNumber | Character string | Taken from the Number Digits field of Q.931 Setup Redirecting Number IE. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. |
| callingOctet3a | Character String | Whether the calling number information can be displayed. |
| callingPartyNum | Character String | Taken from the Q.931. |
| displayIE | Character String | Taken from the Q.931 Setup, display IE. |

*Table 5-24 LCF_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| rasAddress | Character String | Registration and status transport address for this endpoint. |
| rmotExtensionAddr | Character String | Alias address of a called endpoint, present in cases where this information is required to traverse multiple gateways. |
| destinationType | Enumeration | Type of destination endpoint. See ENDPOINT_TYPE. |
| altTranspAddr | Pointer | See ALTERNATE_TRANSPORT_ADDR_TYPE. |

# LRJ_REQUEST_MSG

The LRJ_REQUEST_MSG structure is used to process LRJ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-25:

*Table 5-25 LRJ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| srcInfo | Character string | Sequence of alias addresses for the source endpoint. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| redirectNumber | Character string | Taken from the Number Digits field of Q.931 Setup Redirecting Number IE. |
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. |
| callingOctet3a | Character String | Whether the calling number information can be displayed. |
| displayIE | Character String | Taken from the Q.931 Setup, display IE. |
| callingPartyNum | Character String | Taken from the Q.931. |
| rejectReason | Enumeration | Reason for the rejection of the request. See LRJ_REJECT_REASON_TYPE. |

# RAI_REQUEST_MSG

The RAI_REQUEST_MSG structure is used to process RAI requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-26:

***Table 5-26 RAI_REQUEST_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| callSignalAddress | Character string | Call signaling transport address for this endpoint. |
| almostOut | Integer | Resource usage indication. The value is TRUE or FALSE. |

# DRQ_REQUEST_MSG

The DRQ_REQUEST_MSG structure is used to process DRQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-27:

***Table 5-27 DRQ_REQUEST_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See REGISTER_REQUEST_HEADER. |
| drqReason | Enumeration | Reason received for a DRQ sent by an endpoint. See DRQ_REASON_TYPE. |
| srcCallSignalAddress | Character string | Transport address used at the source for call signaling. |
| answeredCall | Integer | Indicates that this party was the original destination. The value is TRUE or FALSE. |
| callIdentifier | Character string | A unique call identifier (set by the originating endpoint) which can be used to associate RAS signaling with the modified Q.931 signaling used in H225.0. |
| conferenceID | Character string | A unique identifier. |
| clearToken | Pointer | See CLEAR_TOKEN. |

# BRQ_REQUEST_MSG

The BRQ_REQUEST_MSG structure is used to process BRQ requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-28:

***Table 5-28 BRQ_REQUEST_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| answerCall | Integer | Indicates to the Cisco IOS Gatekeeper that the call is incoming. |

*Table 5-28 BRQ_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bi-directional call. |
| callIdentifier | Character string | A unique call identifier (set by the originating endpoint), which can be used to associate RAS signaling with the modified Q.931 signaling used in H.225.0. |
| conferenceID | Character string | A unique conference identifier. |
| endPointCallSignal Address | Character string | Call signalling transport address for this endpoint. |
| cryptoToken | Pointer | See ALV_RESPONSE_MSG. |
| clearToken | Pointer | See CLEAR_TOKEN. |

# IRR_REQUEST_MSG

The IRR_REQUEST_MSG structure is used to process IRR requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-29.

*Table 5-29 IRR_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| srcCallSignalAddress | Character String | Transport address used at the source for call signaling. |
| perCallInfo | Character String | Contains information about the call, including the call identifier, conference ID, originator, bandwidth, and call start time. |

# ALV_REQUEST_MSG

The ALV_REQUEST_MSG structure is used to process ALV requests from the Cisco IOS Gatekeeper. This structure contains the fields shown in Table 5-30:

*Table 5-30 ALV_REQUEST_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |

# GK_WRITE_MSG

The GK_WRITE_MSG structure is used to process responses from the external application to the Cisco IOS Gatekeeper.

This structure contains the fields shown in Table 5-31:

***Table 5-31 GK_WRITE_MSG***

| Field | Field Type | Description |
|---|---|---|
| msgType | Enumeration | See RESPONSE_MSG_TYPE. |
| arqRespMsg | Structure | See ARQ_RESPONSE_MSG. |
| acfRespMsg | Structure | See ACF_RESPONSE_MSG. |
| arjRespMsg | Structure | See ARJ_RESPONSE_MSG. |
| rrqRespMsg | Structure | See RRQ_RESPONSE_MSG. |
| rrjRespMsg | Structure | See RRJ_RESPONSE_MSG. |
| rcfRespMsg | Structure | See RCF_RESPONSE_MSG. |
| lrqRespMsg | Structure | See LRQ_RESPONSE_MSG. |
| lcfRespMsg | Structure | See LCF_RESPONSE_MSG. |
| lrjRespMsg | Structure | See LRJ_RESPONSE_MSG. |
| ripRespMsg | Structure | See RIP_RESPONSE_MSG. |

# ARQ_RESPONSE_MSG

The ARQ_RESPONSE_MSG structure is used to process ARQ responses from the external application. This structure contains the fields shown in Table 5-32:

***Table 5-32 ARQ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| destCallSignalAddress | Character string | Transport address used at the destination for call signaling. |
| destExtraCallInfo | Character string | External addresses for multiple calls. |
| bandWidthPresent | Boolean | Whether a specified bandwidth is present in the request. |
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bidirectional call. |
| redirectNumber | Character string | Taken from the Number Digits field of Q.931 Setup Redirecting Number IE. |

*Table 5-32 ARQ_RESPONSE_MSG*

| Field | Field Type | Description |
|---|---|---|
| redirectReason | Enumeration | Taken from the Q.931 Setup Redirecting Number IE. See REDIRECT_REASON_TYPE. |
| displayIE | Character String | Taken from the Q.931 Setup, display IE. |

# ACF_RESPONSE_MSG

The ACF_RESPONSE_MSG structure is used to process ACF responses from the external application. This structure contains the fields shown in Table 5-33:

*Table 5-33 ACF_RESPONSE_MSG*

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| destCallSignalAddress | Character string | Transport address used at the destination for call signaling. |
| destExtraCallInfo | Character string | External addresses for multiple calls. |
| rmotExtensionAddr | Character string | Alias address of a called endpoint, present in cases where this information is required to traverse multiple gateways. |
| bandWidthPresent | Boolean | Whether a specified bandwidth is present in the request. |
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bidirectional call. |
| destinationType | Enumeration | Type of destination endpoint. See ENDPOINT_TYPE. |
| altEndpt | Structure | See ALTERNATE_ENDPOINT. |
| clearToken | Pointer | See CLEAR_TOKEN. |
| altTranspAddr | Pointer | See ALTERNATE_TRANSPORT_ADDR_TYPE. |
| use_transport | Enumeration | See USE_SPECIFIED_TRANSPORT_TYPE_T. |

# ARJ_RESPONSE_MSG

The ARJ_RESPONSE_MSG structure is used to process ARJ responses from the external application. This structure contains the fields shown in Table 5-34:

***Table 5-34 ARJ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| rejectReason | Enumeration | Reason the request was rejected. See ARJ_REJECT_REASON_TYPE. |

# RRQ_RESPONSE_MSG

The RRQ_RESPONSE_MSG structure is used to process RRQ responses from the external application. This structure contains the fields shown in Table 5-35:

***Table 5-35 RRQ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| terminalAlias | Character string | List of alias addresses by which other terminals can identify this terminal. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. |

# RCF_RESPONSE_MSG

The RCF_RESPONSE_MSG structure is used to process RCF responses from the external application. This structure contains the fields shown in Table 5-36:

***Table 5-36 RCF_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| terminalAlias | Character string | List of alias addresses by which other terminals can identify this terminal. |
| supportedPrefix | Character string | Prefix associated with the supported protocol. |
| alternateGK | Structure | See ALTERNATE_GK. |

# RRJ_RESPONSE_MSG

The RRJ_RESPONSE_MSG structure is used to process RRJ responses from the external application. This structure contains the fields shown in Table 5-37:

***Table 5-37 RRJ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| rejectReason | Enumeration | Reason the request was rejected. See RRJ_REJECT_REASON_TYPE. |

# LRQ_RESPONSE_MSG

The LRQ_RESPONSE_MSG structure is used to process LRQ responses from the external application. This structure contains the fields shown in Table 5-38:

***Table 5-38 LRQ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |

# LCF_RESPONSE_MSG

The LCF_RESPONSE_MSG structure is used to process LCF responses from the external application. This structure contains the fields shown in Table 5-39:

***Table 5-39 LCF_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| destinationInfo | Character string | Sequence of alias addresses for the destination endpoint. |
| destExtraCallInfo | Character string | External addresses for multiple calls. |
| callSignalAddress | Character string | Call signaling transport address for this endpoint. |
| rasAddress | Character String | Registration and status transport address for this endpoint. |
| rmotExtensionAddr | Character String | Alias address of a called endpoint, present in cases where this information is required to traverse multiple gateways. |
| destinationType | Enumeration | Type of destination endpoint. See ENDPOINT_TYPE. |
| altTranspAddr | Pointer | See ALTERNATE_TRANSPORT_ADDR_TYPE. |

# LRJ_RESPONSE_MSG

The LRJ_RESPONSE_MSG structure is used to process LRJ responses from the external application. This structure contains the fields shown in Table 5-40:

***Table 5-40 LRJ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| rejectReason | Enumeration | Reason the request was rejected. See LRJ_REJECT_REASON_TYPE. |

# BRQ_RESPONSE_MSG

The BRQ_RESPONSE_MSG structure is used to process BRQ responses from the external application. This structure contains the fields shown in Table 5-41:

***Table 5-41 BRQ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bidirectional call. |

# BCF_RESPONSE_MSG

The BCF_RESPONSE_MSG structure is used to process BCF responses from the external application. This structure contains the fields shown in Table 5-42:

***Table 5-42 BCF_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| bandWidth | Unsigned integer | Bandwidth (in 100 kbps) requested for the bidirectional call. |

# BRJ_RESPONSE_MSG

The BRJ_RESPONSE_MSG structure is used to process BRJ responses from the external application. This structure contains the fields shown in Table 5-43:

***Table 5-43 BRJ_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| rejectReason | Enumeration | See BRJ_REJECT_REASON_TYPE. |

# ALV_RESPONSE_MSG

The ALV_RESPONSE_MSG structure is used to process ALV responses from the external application. This structure contains the fields shown in Table 5-44:

***Table 5-44 ALV_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |

# CRYPTO_H323_TOKEN

The CRYPTO_H323_TOKEN structure is used to process cryptoTokens. This structure contains the fields shown in Table 5-45:

***Table 5-45 CRYPTO_H323_TOKEN***

| Field | Field Type | Description |
|---|---|---|
| token_type | Enumeration | See CRYPTO_H323_TOKEN_TYPE_S. |
| cryptoEPPwdHash | Structure | See CRYPTO_EP_PWD_HASH. |
| cryptoEPPwdEncr | Structure | See CRYPTO_EP_PWD_ENCR. |
| cryptoEPCert | Structure | See CRYPTO_EP_CERT. |

# CRYPTO_EP_PWD_HASH

The CRYPTO_EP_PWD_HASH structure is used to process cryptoTokens. This structure contains the sections shown in Table 5-46:

***Table 5-46 CRYPTO_EP_PWD_HASH***

| Field | Field Type | Description |
|---|---|---|
| alias | Character string | Registration and status transport address for this endpoint. |
| timestamp | Character string | 32-bit integer that represents UTC time. |
| token | Character string | 16 octet IA5String that represents the MD5 hashed encoded PwdCertToken. |

# CRYPTO_EP_PWD_ENCR

The CRYPTO_EP_PWD_ENCR structure is used to process the encrypted data of a cryptoToken. This structure contains the fields shown in Table 5-47:

***Table 5-47 CRYPTO_EP_PWD_ENCR***

| Field | Field Type | Description |
|---|---|---|
| paramS | Character string | Any runtime parameters. |
| encryptedData | Character string | Encrypted data from the cryptoToken. |

# CRYPTO_EP_CERT

The CRYPTO_EP_CERT structure is used to process the authentication certificate of a cryptoToken. This structure contains the fields shown in Table 5-48:

***Table 5-48 CRYPTO_EP_CERT***

| Field | Field Type | Description |
|---|---|---|
| toBeSigned | Character string | Whether the certificate requires a signature. |
| signature | Character string | Digital signature assigned to the authentication certificate. |

# CLEAR_TOKEN

The CLEAR_TOKEN structure is used to process the clear tokens field. This structure contains the fields shown in Table 5-49:

***Table 5-49 CLEAR_TOKEN***

| Field | Field Type | Description |
|---|---|---|
| objectIdentifier | Character string | Object identifier. |
| password | Character string | Secret character string that is used to authenticate a user or H.323 endpoint. |
| timestamp | Character string | 32-bit integer that represents UTC time. |
| challengeString | Character string | Challenge string used for authentication. |
| random | Character string | Integer value, for example a monotonically increasing sequence number. |
| generalID | Character string | Character string that uniquely identifies either the sender or receiver. |
| nonstd_objectID | Character string | Object identifier that is used to indicate the type and format of the nonstandard data being sent in the clear token. |
| nonstd_data | Character string | Nonstandard data in the clear tokens field. |

# ALTERNATE_GK

The ALTERNATE_GK structure is used to process information about an alternate gatekeeper. This structure contains the fields shown in Table 5-50:

***Table 5-50 ALTERNATE GK***

| Field | Field Type | Description |
|---|---|---|
| rasAddress | Character string | Registration and status transport address for this endpoint. |
| gkIdentifier | Character string | Identifier of the gatekeeper. |
| needToRegister | Boolean | Whether there is a need to register with this gatekeeper. |
| priority | Integer | Priority of this gatekeeper. Possible values are 1 through 127. |

# ALTERNATE_ENDPOINT

The ALTERNATE_ENDPOINT structure is used to process information about an alternate H.323 endpoint.

This structure contains the fields shown in Table 5-51:

***Table 5-51 ALTERNATE_ENDPOINT***

| Field | Field Type | Description |
|---|---|---|
| callSignalAddress | Character string | Registration and status transport address for this endpoint. |
| tokenP | Structure | See CLEAR_TOKEN. |

# ALTERNATE_TRANSPORT_ADDR_TYPE

The ALTERNATE_TRANSPORT_ADDR_TYPE structure is used to convey information about an Annex E transport address of the destination H.323 endpoint. This structure contains the fields shown in Table 5-52:

***Table 5-52 ALTERNATE_TRANSPORT_ADDR_TYPE***

| Field | Field Type | Description |
|---|---|---|
| annexE | Character string | Annex E transport address of the destination endpoint. |
| nextP | Pointer | Pointer to the next node in the linked list. |

# RIP_RESPONSE_MSG

The RIP_RESPONSE_MSG structure is used to process requests from the external application for additional time. This structure contains the fields shown in Table 5-53:

***Table 5-53 RIP_RESPONSE_MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |
| delay | Integer | Amount of time, in milliseconds (1 through 65,536), that the endpoint should wait before retrying the request. |

# UNSUPPORTED_MSG

The UNSUPPORTED_MSG structure is used to process requests from the Cisco IOS Gatekeeper that contain a RAS message type that is not supported by the API. This structure contains the field shown in Table 5-54:

***Table 5-54 UNSUPPORTED MSG***

| Field | Field Type | Description |
|---|---|---|
| headerInfo | Structure | See HEADER_INFO. |

# Enumerations

Some of the API structures contain enumerations. An enumeration is simply a list of possible values. This section lists the enumerations used by the structures and includes the following sections:

## STATUS_TYPE

The STATUS_TYPE enumeration lists the possible return values from calls to read, write, register and unregister functions. The possible values are:

- PROCESSING_SUCCESSFUL
- CONNECT_IN_PROGRESS
- NULL_POINTER_PASSED
- TCP_HANDLE_ERROR
- TCP_CONNECT_ERROR
- TCP_READ_ERROR
- TCP_BIND_ERROR
- TCP_LISTEN_ERROR
- TCP_ADDRESS_ALREADY_IN_USE
- TCP_ADDRESS_NOT_AVAIL
- TCP_NONBLOCK_ERROR
- MEM_ALLOC_FAIL
- MSG_READ_ERROR
- TCP_WRITE_ERROR
- TCP_CONNECTION_CLOSED
- INCOMPLETE_MSG_READ
- INVALID_MSG_SPECIFIED
- INVALID_ENDPOINT_SPECIFIED
- INVALID_REDIRECT_REASON_SPECIFIED
- INVALID_REJECT_REASON_SPECIFIED
- INVALID_DELAY_SPECIFIED
- HEADER_INFO_INCOMPLETE

## REG_STATUS_TYPE

The REG_STATUS_TYPE enumeration lists the possible status values for registration and unregistration responses received from the Cisco IOS Gatekeeper. The possible values are:

- SUCCESSFUL
- INVALID_PRIORITY
- INVALID_FILTERS
- INVALID_GKID

## ENDPOINT_TYPE

The ENDPOINT_TYPE enumeration lists the possible types of endpoints. The possible values are:

- GATEKEEPER
- TERMINAL
- MCU
- PROXY
- VOICEGATEWAY
- H320GATEWAY
- OTHERGATEWAY
- ENDPOINT_INFO_NOT_RCVD

## REDIRECT_REASON_TYPE

The REDIRECT_REASON_TYPE enumeration lists the possible reasons that a call might be redirected. The possible values are:

- REDIRECT_REASON_UNKNOWN = 0
- REDIRECT_REASON_CALL_FWD_BUSY = 1
- REDIRECT_REASON_CALL_FWD_NO_REPLY = 2
- REDIRECT_REASON_CALL_DEFLECTION = 4
- REDIRECT_REASON_CLED_DTE_OUT_OF_ORDER = 9
- REDIRECT_REASON_CALL_FWDING_BY_CLED_DTE = 10
- REDIRECT_REASON_CALL_FWDING_UNCONDL = 15
- REDIRECT_REASON_INFO_NOT_RCVD= 99

## DRQ_REASON_TYPE

The DRQ_REASON_TYPE enumeration lists the reasons received for a DRQ sent by an endpoint. The possible values are:

- DRQ_REASON_FORCED_DROP = 1
- DRQ_REASON_NORMAL_DROP = 2
- DRQ_REASON_UNDEF_REASON = 3

## LRJ_REJECT_REASON_TYPE

The LRJ_REJECT_REASON_TYPE enumeration lists the possible reasons that an LRQ request might be rejected. The possible values are:

- LRJ_NOT_REGISTERED
- LRJ_INVALID_PERMISSION
- LRJ_REQUEST_DENIED
- LRJ_UNDEFINED_REASON
- LRJ_SECURITY_DENIAL

## REQUEST_MSG_TYPE

The REQUEST_MSG_TYPE enumeration lists the possible messages that can be received from the Cisco IOS Gatekeeper. The possible values are:

- UNKNOWN_MSG
- MSG_NOT_SUPPORTED
- RRQ_REQUEST_MSG
- URQ_REQUEST_MSG
- ARQ_REQUEST_MSG
- LRQ_REQUEST_MSG
- LRJ_REQUEST_MSG
- LCF_REQUEST_MSG
- BRQ_REQUEST_MSG
- RAI_REQUEST_MSG
- DRQ_REQUEST_MSG
- RRQ_REGISTER_RESPONSE_MSG
- URQ_REGISTER_RESPONSE_MSG
- ARQ_REGISTER_RESPONSE_MSG
- LRQ_REGISTER_RESPONSE_MSG
- LCF_REGISTER_RESPONSE_MSG
- LRJ_REGISTER_RESPONSE_MSG
- BRQ_REGISTER_RESPONSE_MSG
- RAI_REGISTER_RESPONSE_MSG
- DRQ_REGISTER_RESPONSE_MSG
- RRQ_UNREGISTER_RESPONSE_MSG
- URQ_UNREGISTER_RESPONSE_MSG
- ARQ_UNREGISTER_RESPONSE_MSG
- LRQ_UNREGISTER_RESPONSE_MSG
- LCF_UNREGISTER_RESPONSE_MSG
- LRJ_UNREGISTER_RESPONSE_MSG
- BRQ_UNREGISTER_RESPONSE_MSG
- RAI_UNREGISTER_RESPONSE_MSG
- DRQ_UNREGISTER_RESPONSE_MSG

## RRJ_REJECT_REASON_TYPE

The RRJ_REJECT_REASON_TYPE enumeration lists the possible reasons that an RRQ request might be rejected. The possible values are:

- RRJ_UNDEFINED_REASON
- RRJ_SECURITY_DENIAL
- RRJ_RESOURCE_UNAVAIL

## ARJ_REJECT_REASON_TYPE

The ARJ_REJECT_REASON_TYPE enumeration lists the possible reasons that an ARQ request might be rejected. The possible values are:

- CALLED_PARTY_NOT_REGISTERED
- INVALID_PERMISSION
- REQUEST_DENIED
- UNDEFINED_REASON
- ARJ_RESOURCE_UNAVAIL
- ARJ_SECURITY_DENIAL

## BRJ_REJECT_REASON_TYPE

The BRJ_REJECT_REASON_TYPE enumeration lists the possible reasons that a BRQ request might be rejected. The possible values are:

- BRJ_NOT_BOUND
- BRJ_INVALID_CONF_ID
- BRJ_INVALID_PERMISSION
- BRJ_INSUFFICIENT_RSC
- BRJ_INVALID_REVISION
- BRJ_UNDEFINED_REASON
- BRJ_SECURITY_DENIAL

## RESPONSE_MSG_TYPE

The RESPONSE_MSG_TYPE enumeration lists the possible messages that the external application can send to the Cisco IOS Gatekeeper. The possible values are:

- RRQ_RESPONSE_MSG
- RCF_RESPONSE_MSG
- RRJ_RESPONSE_MSG
- ARQ_RESPONSE_MSG
- ACF_RESPONSE_MSG
- ARJ_RESPONSE_MSG
- LRQ_RESPONSE_MSG

- LCF_RESPONSE_MSG
- LRJ_RESPONSE_MSG
- RIP_RESPONSE_MSG
- BRQ_RESPONSE_MSG
- BCF_RESPONSE_MSG
- BRJ_RESPONSE_MSG

## REGISTER_MSG_TYPE

The REGISTER_MSG_TYPE enumeration lists the possible registration messages that the external application can send to the Cisco IOS Gatekeeper. The possible values are:

- RRQ_REGISTER_MSG
- URQ_REGISTER_MSG
- ARQ_REGISTER_MSG
- LRQ_REGISTER_MSG
- LCF_REGISTER_MSG
- LRJ_REGISTER_MSG
- BRQ_REGISTER_MSG
- RAI_REGISTER_MSG
- DRQ_REGISTER_MSG

## REPORT_DEST_T

The REPORT_DEST_T enumeration lists the possible destinations for the Gatekeeper API debug output. The possible values are:

- REPORT_CONSOLE
- REPORT_SYSLOG

## CRYPTO_H323_TOKEN_TYPE_S

The CRYPTO_H323_TOKEN_TYPE_S enumeration lists the possible types of cryptoTokens. The possible values are:

- NO_CRYPTO_TOKEN
- CRYPTO_EP_PWD_HASH
- CRYPTO_EP_PWD_ENCR
- CRYPTO_EP_CERT

**Note** In the first release of the GKTMP and API, the CRYPTO_EP_PWD_HASH is the only type of cryptoToken supported.

## USE_SPECIFIED_TRANSPORT_TYPE_T

The USE_SPECIFIED_TRANSPORT_TYPE_T enumeration lists the possible transport types that an endpoint can select for H.225 signalling. The possible values are:

- TRANSPORT_NONE
- ANNEX_E
- TCP

# Limits

Some of the fields are limited in size. The limits are set using variables in the header file. The limits as set in the default header file are shown in Table 5-55:

***Table 5-55 Field Size LImits***

| Variable | Initial Value |
|---|---|
| MAX_IP_ADDR_LENGTH | 15 |
| MAX_VERSION_ID_LENGTH | 4 |
| MAX_ENDPOINT_LENGTH | 128 |
| MAX_TRANSACTION_ID_LENGTH | 24 |
| MAX_NUM_ENDPOINT_TYPES | 7 |
| MAX_NUM_SUPPORTED_PREFIX | 10 |
| MAX_NUM_ARQ_DEST_INFO | 20 |
| MAX_NUM_ARQ_REDIRECT_REASON | 7 |
| MAX_NUM_LRQ_DEST_INFO | 20 |
| MAX_NUM_LRQ_REDIRECT_REASON | 7 |
| MAX_NUM_LCF_DEST_INFO | 20 |
| MAX_NUM_LCF_RMOT_EXTENSION_ ADDR | 20 |
| MAX_NUM_LRJ_DEST_INFO | 20 |
| MAX_CRYPTO_TOKEN_FIELDS | 5 |

CHAPTER 6

# GKTMP Command Reference

This chapter describes commands that support the new Cisco IOS Gatekeeper functions and includes the following commands:

- **server trigger**
- **timer server timeout**
- **server registration-port**
- **server flow-control**
- **show gatekeeper servers**
- **show gatekeeper status**
- **debug gatekeeper servers**

**Note** As with all Cisco IOS commands, you can abbreviate the Cisco IOS Gatekeeper trigger registration commands. To abbreviate a command, simply enter the first few characters of the command and press tab. To obtain online help for a command, enter the first few characters of the command followed by a question mark.

For additional Cisco IOS commands, see the following documents:

*Cisco High Performance Gatekeeper*

*Cisco IOS Voice, Video, and Fax Configuration Guide*, Release 12.2

*Cisco IOS Voice, Video, and Fax Command Reference*, Release 12.2

## server trigger

To configure a static server trigger for external applications, enter the **server trigger** command from Gatekeeper mode. Enter the **no** form of this command to remove a single statically configured trigger entry. Enter the "all" form of the command to remove every static trigger you configured if you want to delete them all.

**server trigger {arq | lcf | lrj | lrq | rrq | urq | drq | rai | brq}** ***gkid priority server-id server-ip_address server-port***

**no server trigger { arq | lcf | lrj | lrq | rrq | urq | drq | rai | brq}** ***gkid priority***

**no server trigger all**

**Syntax Description**

| | |
|---|---|
| **arq** \| **lcf** \| **lrj** \| **lrq** \| **rrq** \| urq \| **drq** \| **rai** \| **brq** | The RAS messages for which you can create triggers on the Cisco IOS Gatekeeper. You can specify only one message type per server trigger command. There is a different trigger submode for each message type. Each trigger submode has its own set of applicable commands. |
| *gkid* | The identifier of the Cisco IOS Gatekeeper. |
| *priority* | The priority for this particular trigger. Possible values are 1 through 20. 1 is the highest. |
| *server-id* | The identifier of the external application. |
| *server-ip_address* | The IP address of the server on which the external application is running. |
| *server-port* | The port on which the server listens for messages from the Cisco IOS Gatekeeper. |

**Command Modes** Gatekeeper configuration

**Related Commands** The following subcommands can be used in any of the trigger submodes:

- **Trigger Submodes**
- **shutdown**

The following subcommands can be used in specific trigger submodes to configure certain types of trigger conditions:

- **destination-info**
- **redirect-reason**
- **remote-ext-address**
- **endpoint-type**
- **supported-prefix**

# Trigger Submodes

## info-only

To indicate to the Cisco IOS Gatekeeper that messages that meet the specified trigger parameters should be sent as notifications only and that the Cisco IOS Gatekeeper should not wait for a response from the external application, use the **info-only** subcommand.

**info-only**

**Syntax Description**

| | |
|---|---|
| **info-only** | Informational only. No need to wait for acknowledgment. |

**Command Modes** Any of the Cisco IOS Gatekeeper trigger submodes

## shutdown

To temporarily disable a trigger, use the **shutdown** subcommand. Cisco IOS Gatekeepers do not consult triggers in shutdown state when determining whether a message should be forwarded to an external application.

**shutdown**

**Syntax Description**

| | |
|---|---|
| **shutdown** | Changes the administrative state of a trigger to shutdown. |

**Command Modes** Any of the Cisco IOS Gatekeeper trigger submodes

## destination-info

To configure a trigger that is based on a particular destination, use the **destination-info** subcommand.

**destination-info {e164 | email-id | h323-id}** ***value***

**Syntax Description**

| | |
|---|---|
| **e164** | Indicates that the destination address is an E.164 address. |
| **email-id** | Indicates that the destination address is an e-mail ID. |
| **h323-id** | Indicates that the destination address is an H.323 ID. |
| *value* | Specifies the value against which to compare the destination address in the RAS messages. For E.164 addresses, the following wildcards can be used:<br>• A trailing series of periods, each of which represents a single character.<br>• A trailing asterisk, which represents one or more characters. |

**Command Modes** Cisco IOS Gatekeeper ARQ, LRQ, LCF, LRJ, and DRQ trigger submodes

## redirect-reason

To configure a trigger that is based on a specific redirect reason, use the **redirect-reason** subcommand.

**redirect-reason** *value*

**Syntax Description**

| | |
|---|---|
| value | Specifies the value against which to compare the redirect reason in the RAS messages. Possible values are 0-65535. Currently–used redirect reasons are:<br>• 0—Unknown reason<br>• 1—Call forwarding busy or called DTE busy<br>• 2—Call forwarded no reply<br>• 4—Call deflection<br>• 9—Called DTE out of order<br>• 10—Call forwarding by the call DTE<br>• 15—Call forwarding unconditionally |

**Command Modes** Cisco IOS Gatekeeper ARQ, LRQ, DRQ, and BRQ trigger submodes

## remote-ext-address

To configure a trigger that is based on a specific remote extension address, use the **remote-ext-address** subcommand.

**remote-ext-address** ***e164 value***

**Syntax Description**

| | |
|---|---|
| e164 | Indicates that the remote extension address is an E.164 address. |
| value | Specifies the value against which to compare the destination address in the RAS messages. The following wildcards can be used:<br>• A trailing series of periods, each of which represents a single character.<br>• A trailing asterisk, which represents one or more characters. |

**Command Modes** Cisco IOS Gatekeeper LCF trigger submode

## endpoint-type

To configure a trigger that is based on a specific endpoint, use the **endpoint-type** subcommand.

**endpoint-type** *value*

| Syntax Description | | |
|---|---|---|
| | *value* | Specifies the value against which to compare the endpoint-type in the RAS messages. The possible values are:<br>• gatekeeper—The endpoint is an H.323 gatekeeper.<br>• h320-gateway—The endpoint is an H.320 gateway.<br>• mcu—The endpoint is an MCU.<br>• other-gateway—The endpoint is a type of gateway not specified on this list.<br>• proxy—The endpoint is an H.323 proxy.<br>• terminal—The endpoint is an H.323 terminal.<br>• voice-gateway—The endpoint is a voice type gateway. |

**Command Modes** Cisco IOS Gatekeeper RRQ, URQ, and RAI trigger submodes

### supported-prefix

To configure a trigger that is based on a specific supported prefix, use the **supported-prefix** subcommand.

**supported-prefix** *value*

| Syntax Description | | |
|---|---|---|
| | value | Specifies the value against which to compare the supported prefix in the RAS messages. The possible values are any E.164 pattern used as a gateway technology prefix. The value string can contain any of the following: 0123456789#*, |

**Command Modes** Cisco IOS Gatekeeper RRQ, URQ, and RAI trigger submodes

## timer server timeout

To define the server timeout for GKTMP messages, use the **timer server timeout** command.

**timer server timeout** *value*

| Syntax Description | | |
|---|---|---|
| | *value* | The timeout in seconds. Possible values are 1 through 5. The default value is 3. |

**Command Modes** Gatekeeper configuration

# server registration-port

To define a listener port to be used by the external applications to establish connections to the gatekeeper on this router, use the **server registration-port** gatekeeper configuration command.

**server registration-port** *port_number*

**no server registration-port**

The **no** form of this command forces the gatekeeper on this router to close the listener port so that it cannot receive any additional registrations. However, existing connections between the gatekeeper and external application are left open.

**Syntax Description**

| | |
|---|---|
| port_number | The port on which the Cisco IOS Gatekeeper should listen for registration messages from external applications. |

**Command Modes** Gatekeeper configuration

# server flow-control

To enable flow control on the Cisco IOS Gatekeeper (GK) and reset all thresholds to default, use the **server flow-control** command in gatekeeper configuration mode. To disable GK flow control, use the **no** form of this command.

**server flow-control [onset *value*] [abatement *value*] [qcount *value*]**

**no server flow-control**

## Syntax Description

| | |
|---|---|
| **onset** *value* | A percentage of the server timeout value that is used to mark the server as usable or unusable. The range of valid values is 1 through 100; the default value is 80.<br><br>For example, if the server time out value is 3 seconds, the onset value is 50, and the abatement value is 40, when the average response time from the server to the GKTMP reaches 1.5 seconds (the onset percentage of the server timeout value), the server is marked as unusable. During the period that the server is marked as unusable, REQUEST ALV messages are still sent to the unusable server. When the response time is lowered to 1.2 seconds (the abatement percentage of the timeout value), the server is marked usable again and the GKTMP resumes sending messages to the server. |
| **abatement** *value* | A percentage of the server timeout value that is used to mark the server as unusable or usable. The range of valid values is 1 through 100; the default value is 50.<br><br>For example, if the server time out value is 3 seconds, the onset value is 50, and the abatement value is 40, when the average response time from the server to the GKTMP reaches 1.5 seconds (the onset percentage of the server timeout value), the server is marked as unusable. During the period that the server is marked as unusable, REQUEST ALV messages are still sent to the unusable server. When the response time is lowered to 1.2 seconds (the abatement percentage of the timeout value), the server is marked usable again and the GKTMP resumes sending messages to the server.<br><br>**Note** Note The abatement value cannot be greater than or equal to the onset value. |
| **qcount** *value* | Identifies the threshold for the length of the outbound queue on the GK. The queue contains messages waiting to be transmitted to the server. The TCP socket between the GK and GKTMP server queues messages if it has too many to transmit. If the count of outbound queue length on the server reaches the qcount value, the server is marked unusable. The range of valid values is 1 through 2000; the default value is 50. |

## Command Modes

Gatekeeper configuration

### Examples

The following example shows using the command with the default values:

```
Router# server flow-control
```

In the following example, the GKTMP Interface Resiliency Enhancement feature is enabled with an onset level of 50:

```
Router# server flow-control onset 50
*Mar  8 20:05:34.081: gk_srv_handle_flowcontrol: Flow control enabled
Router# show running configuration

Building configuration...

Current configuration : 1065 bytes
!
version 12.2
no service single-slot-reload-enable
service timestamps debug datetime msec
service timestamps log uptime
no service password-encryption
!
hostname snet-3660-3
!
.
.
.
gatekeeper
 zone local snet-3660-3 cisco.com
 zone remote snet-3660-2 cisco.com 209.165.200.225 1719
 zone prefix snet-3660-2 408*
 lrq forward-queries
 no use-proxy snet-3660-3 default inbound-to terminal
 no use-proxy snet-3660-3 default outbound-from terminal
 no shutdown
 server registration-port 8000
 server flow-control onset 50
!
!
.
.
.
end
```

# show gatekeeper servers

To display a list of the triggers (whether dynamically registered from the external applications or statically configured from the command-line interface), use the **show gatekeeper servers** EXEC command.

**show gatekeeper servers [*gkid*]**

## Syntax Description

| | |
|---|---|
| *gkid* | Specifies the ID of the gatekeeper. If you specify a gatekeeper ID, only the information about the external applications that are registered with the specified gatekeeper is displayed. If you do not specify a gatekeeper ID, information about all the external applications that are registered with any of the Cisco IOS Gatekeepers on this router is displayed. |

## Command Modes

EXEC mode

## Examples

The following example shows the **show gatekeeper servers** command output:

***Example 6-1 show gatekeeper servers Output (version 2.0)***

```
router# show gatekeeper servers gk102

               GATEKEEPER SERVERS STATUS
               =========================

     Gatekeeper Server listening port: 20000
        Gatekeeper GKTMP version:2.0

     Gatekeeper-ID: gk102
     --------------------
     RRQ  Priority: 1
       Server-ID: sj-server
       Server IP address: 1.14.93.28:42387
       Server type: dynamically registered
       Connection Status: active
         Server GKAPI version:2.0
       Trigger Information:
         Supported Prefix: 10#
         Supported Prefix: 3#
     RRQ  Priority: 2
       Server-ID: sf-server
       Server IP address: 1.14.93.43:3820
       Server type: CLI-configured
       Connection Status: inactive
         Server GKAPI version:2.0
       Trigger Information:
         Endpoint-type: MCU
         Endpoint-type: VOIP-GW
         Supported Prefix: 99#
     ARQ  Priority: 1
       Server-ID: sj-server
       Server IP address: 1.14.93.28:42387
       Server type: dynamically registered
       Connection Status: active
         Server GKAPI version:2.0
       Trigger Information:
         Destination Info: M:nilkant@zone14.com
         Destination Info: E:1800.......
         Redirect Reason: Call forwarded no reply
         Redirect Reason: Call deflection
```

***Example 6-2 show gatekeeper servers Output (version 3.1)***

```
Router# show gatekeeper server

            GATEKEEPER SERVERS STATUS
            =========================

Gatekeeper Server listening port: 8250
Gatekeeper Server timeout value: 30 (100ms)
GateKeeper GKTMP version: 3.1

Gatekeeper-ID: Gatekeeper1
------------------------
  RRQ  Priority: 5
    Server-ID: Server43
    Server IP address: 209.165.200.254:40118
    Server type: dynamically registered
    Connection Status: active
    Trigger Information:
      Trigger unconditionally

    Server Statistics:
    REQUEST RRQ Sent=0
    RESPONSE RRQ Received = 0
    RESPONSE RCF Received = 0
    RESPONSE RRJ Received = 0
    Timeout encountered=0
    Average response time(ms)=0
    Server Usable=TRUE
```

# show gatekeeper status

To display statistics about the gatekeeper, including authorization and authentication status and if load balancing and flow control are enabled, use the show gatekeeper status command in EXEC mode.

**show gatekeeper status**

## Syntax Description

This command has no arguments or keywords.

## Command Modes

EXEC mode

## Examples

The following example shows output from the show gatekeeper status command:

```
Router# show gatekeeper status

Gatekeeper State: UP
    Load Balancing:   DISABLED
    Flow Control:     ENABLED
    Zone Name:        snet-3660-3
    Accounting:       DISABLED
    Endpoint Throttling:        DISABLED
    Security:         DISABLED
    Maximum Remote Bandwidth:              unlimited
    Current Remote Bandwidth:              0 kbps
    Current Remote Bandwidth (w/ Alt GKs): 0 kbps
```

Table 6-1 describes the significant fields shown in the display.

***Table 6-1 show gatekeeper status Field Descriptions***

| Field | Description |
|---|---|
| Gatekeeper State | Gatekeeper state has the following values:<br>• UP is operational.<br>• DOWN is administratively shut down.<br>• INACTIVE is administratively enabled, that is, the no shutdown command has been issued, but no local zones have been configured.<br>• HSRP STANDBY indicates that the gatekeeper is on hot standby and will take over when the currently active gatekeeper fails. |
| Load Balancing | Shows if load balancing is enabled. |
| Flow Control | Shows if server flow control is enabled. |
| Zone Name | Displays the zone name to which the gatekeeper belongs. |
| Accounting | Shows if authorization and accounting features are enabled. |
| Endpoint Throttling | Shows if endpoint throttling is enabled. |
| Security | Shows if security features are enabled. |
| Bandwidth | Shows the maximum remote bandwidth, current remote bandwidth, and current remote bandwidth with alternate gatekeepers. |

# debug gatekeeper servers

To turn debugging on, use the **debug gatekeeper servers** EXEC command. This command traces all the message exchanges between the Cisco IOS Gatekeeper and the external application. This command also displays any errors that occur when sending messages to the external application or when parsing messages from the external application. The **no** form of this command turns debugging off.

**debug gatekeeper servers**

**no debug gatekeeper servers**

## Syntax Description

This command has no keywords or arguments.

## Command Modes

EXEC mode

## Examples

The following example shows the debug gatekeeper servers output:

***Example 6-3 debug gatekeeper servers Output***

```
router#debug gatekeeper servers
############ begin screen trace
00:08:47:GK:processing server msg:
REGISTER RRQ
From:server1
To:gk617
Priority:1


00:08:47:GK TMSG encoded to write buffer:
"REGISTER RRQ
From:gk617
To:server1
Priority:1
Status:success

"

00:11:16:GK TMSG encoded to write buffer:
"REQUEST RRQ
From:gk617
To:server1
Transaction-Id:6121529400000001
Content-Length:62

c=I:1.14.93.92:1720
r=I:1.14.93.92:24999
t=proxy
a=H:px14
"

00:11:16:GK:processing server msg:
RESPONSE RRQ
From:server1
To:gk617
```

```
Transaction-Id:6121529400000001
Content-Length:35

a=M:jsmith
p=1# 2 # 3# 1800...

00:11:45:GK TMSG encoded to write buffer:
"REQUEST RRQ
From:gk617
To:server1
Transaction-Id:6121529400000002
Content-Length:72

c=I:1.14.93.130:1720
r=I:1.14.93.130:4307
t=voice-gateway
a=H:gw130
"

00:11:45:GK:processing server msg:
RESPONSE RRJ
From:server1
To:gk617
Transaction-Id:6121529400000002
Content-Length:18

R=securityDenial
############ end screen trace
```