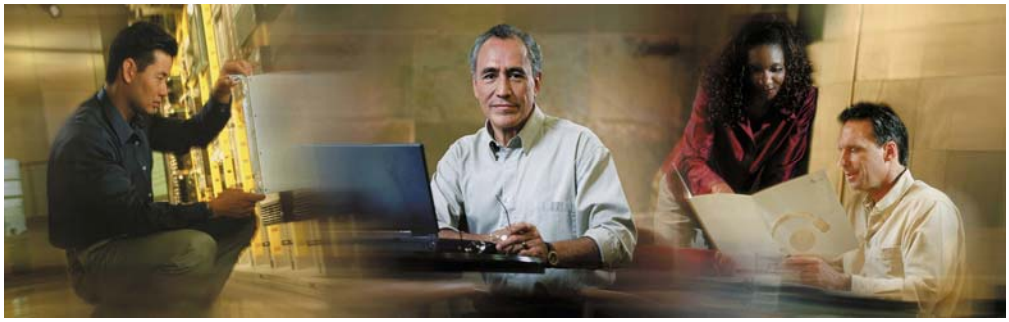


Cisco E-Mail Manager
Application Programming Interface Guide

Cisco E-Mail Manager Release 5.0(0)

Copyright

Table of Contents

Introduction

Introduction

Welcome to the *Cisco E-Mail Manager Application Programming Interface Guide*. This introduction contains the following sections:

Audience

This guide provides instructions, reference information, and examples for programmers and developers creating external application to E-Mail Manager that work with agents and messages.

It is recommended that users of this guide also read the *Cisco E-Mail Manager Overview Guide*.

Cisco E-Mail Manager Documentation

The following documentation is available for E-Mail Manager Release 5.0(0):

Document	Audience	Description	Where to Find
Cisco E-Mail Manager Overview Guide	All E-Mail Manager users.	An overview of E-Mail Manager functionality, capability, and architecture.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager Administration Guide	Administrators and managers working with the Administration Desktop.	Conceptual information about typical administrative and managerial tasks done through the Administration Desktop.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager Agent Guide	Managers who set up agents and determine how they will respond to e-mail messages, as well as those agents.	Conceptual information about typical agent tasks done through the Agent Desktop.	Cisco E-Mail Manager CD Help icon in the Agent Desktop Cisco Web site: www.cisco.com

Cisco E-Mail Manager Installation and Configuration Guide	Those who install and configure E-Mail Manager on the server.	Instructions and requirements for installing and configuring E-Mail Manager on the server.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager Database Guide	Those who maintain the database used by E-Mail Manager, and those who generate reports.	Descriptions of all database tables and columns	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager Implementation and Customization Guide	Those who are implementing and customizing E-Mail Manager to fit particular business needs of the customer.	Guidelines on implementing E-Mail Manager to meet your business needs, instructions on customizing the look of the Web interface, and information about extending the capabilities of rules and templates.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager Application Programming Interface Guide	Programmers and developers creating external application to E-Mail Manager that work with agents and messages.	Reference information, instructions, and examples on building applications that communicates with the E-Mail Manager API Server.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Cisco E-Mail Manager External Data Access Guide	Programmers and developers customizing E-Mail Manager to work with data from an external database.	Information on the External Data Access toolkit, which demonstrates how data from external databases can be used within E-Mail Manager.	Cisco E-Mail Manager CD Help menu in the Administration Desktop Cisco Web site: www.cisco.com
Release Notes for Cisco E-Mail Manager	Those who are administering and managing E-Mail Manager.	Information about the current release, known problems, and documentation updates.	Cisco E-Mail Manager CD Cisco Web site: www.cisco.com

In addition, for all screens in the Agent Desktop and the Administration Desktop, there is online help.

If you are integrating E-Mail Manager with ICM Software, you should also consult the Cisco ICM Software documentation set.

Getting Started with E-Mail Manager

This section contains the following information:

Browser Versions

Browser Settings

Desktop Colors

Logging In to the Agent Desktop

Logging In to the Administration Desktop

For more information about getting started with E-Mail Manager, or if you cannot successfully log in, see your system administrator.

Browser Versions

For complete and current information on browser versions for E-Mail Manager Web interface, see the Cisco Intelligent Contact Management Software Release 5.0(0) Bill of Materials (BOM).

The ICM BOM is available at:

http://www.cisco.com/univercd/cc/td/doc/product/icm/index.htm.

Browser Settings

Before using E-Mail Manager, ensure that your browser is set up to:

Always accept cookies.

Compare the document in cache to the document on the network once per session.

Enable JavaScript

Enable Style Sheets

Caution: Internet Explorer must use the Microsoft Virtual Machine for Java Applets. Ensure that the Sun Virtual Machine is not selected in the Advanced Tab of the Internet Options dialog box. (This option may or may not be available, depending on whether the Sun's Virtual Machine was installed and the options selected.)

For specific instructions, see the documentation for your browser.

Desktop Colors

Your desktop computer must be set to display at least 256 colors in order for you to use E-Mail Manager successfully.

Logging In to the Agent Desktop

To log into the Agent Desktop, you can point your browser to server in one of the following ways:

a. `http://<hostName>/<instanceName>`

b. `http://<hostName>/<instanceName>/default/cem/index.html`

c. `http://<hostname>/<instanceName>/uicommander?req=cem.userMaintenance.cemLoginStart`

Note: The instance name part of the URL is case-sensitive and must be entered exactly as the instance was named on the E-Mail Manager server.

You may want to include your user ID in the bookmarked URL, to automatically populate the Login Name field when the Log In page opens. To do this, you must use the URL in option-b above, and include the user ID as follows:

`http://<hostName>/<instanceName>/default/cem/index.html?user=`*`userID`*

Caution: Do not attempt to run multiple browser sessions on the same desktop with different user IDs or connected to different servers. Unexpected errors may result.

Logging In to the Administration Desktop

To log into the Administration Desktop, point your browser to the name of the server, followed by the port number, as follows:

`http://`*`server-name:port-number`*

The default port number for the Administration Desktop is 8088. If your installation has multiple instances, other instances will have different port numbers.

In the Log In screen, enter your Login Name and Password.

If there is another active session with your Login Name, you are prompted to close that session and login again.

Caution: Do not attempt to run multiple browser sessions on the same desktop with different user IDs or connected to different servers. Unexpected errors may result.

Getting Help

For issues with Cisco E-Mail Manager, your system manager can open a case through the Technical Assistance Center on the Cisco Web site: www.cisco.com. You will need your Contract ID to open a case.

If you have questions, please call us at 800-553-2447, option #3, or send email to tac@cisco.com. From outside of the United States, call 1-408-526-7209.

About the API

Functionality of the API

This topic contains the following sections:

Overview

The E-Mail Manager API supports typical agent functionality. Agents using a third-party application can retrieve, read, and respond to messages using a variety of commands. Agents can also be notified of message-related events.

Commands

This section lists the commands supported by the E-Mail Manager API related to:

Agents

Attachments

Categories

Messages

Message note and history

Events

Queues

Templates

See Also

Overview of API Commands

Agent Commands

The following commands provide functionality related to agents:

```
agentConnect

agentDisconnect

agentDisconnectDuplicateAndConnect

agentGetLogoutReasons

agenttGetNotReadyReasons

agentGetProperties

agentGetState

agentGetStatistics

agentMakeNotReady

agentMakeNotRoutable

agentMakeReady

agentMakeRoutable
```

Attachment Commands

The following commands provide functionality related to attachments:

```
attachGetContent

attachGetList
```

Category Commands

The following commands provide functionality related to categories:

```
catsChangeForMsg

catsGetList

catsGetListForMsg
```

Event Commands

The following commands provide functionality to work with events;

```
eventRegister

eventUnregister
```

Message Commands

The following commands provide functionality to work with messages:

```
msgClaim

msgCreateNewStub

msgDelete

msgEscalate

msgExit

msgGetContent

msgGetExtendedAttributes

msgGetLatestResponseDraft

msgGetResponses

msgGetStatus

msgMarkForArchiving

msgOpenByQueueAndKey

msgOpenFromQueues

msgOpenForRead

msgOpenForResposne

msgOpenRouted

msgReassign

msgSaveAsDraft
```

```
msgSendKeepCurrent

msgSendMarkForArchiving

msgUnarchive
```

Message note and history commands

The following commands provide functionality related to message notes and history:

```
actionHistoryGetForMessage

historyCheckOldForSender

historyCheckOldForTrackingNumber

historyGetForSender

historyGetForTrackingNumber

notesAddForMessage

notesAddForSender

notesAddForTrackingNumber

notesGetForMessage

notesGetForSender

notesGetForTrackingNumber
```

Queue Commands

The following commands provide functionality to get information about queues:

```
queueGetAgentList

queueGetStatistics

queueGetMemberSkillgroupList

queueGetMsgInfo

queueGetPeerList
```

queueGetReassignableList

queueGetSkillgroupList

Template Commands

The following commands provide functionality to get information about templates:

templCreate

templDelete

templGetAllKeywords

templGetAttachments

templGetDefaultDynamicText

templGetDynamicText

templGetDynamicTokenInfo

templGetEditFields

templGetKeywords

templGetList

templGetPublicLibraries

templGetText

templReplace

Events

Following are the events a third-party application can receive:

Event Type 1: Overdue message

Event Type 2: Message received on queue

Event Type 2: Message unarchived on queue

Event Type 2: Message reassigned to queue

Event Type 3: Queue overloaded

Event Type 4: Agent made not routable

See Also

Overview of Events

Limitations

The E-Mail Manager API does not support typical administrative functions.

To perform administrative functions such as creating agents and skill groups, defining rules, and connecting to inbound and outbound e-mail servers, you must use the Administration Desktop. For more information, see the *Cisco E-Mail Manager Administration Guide*.

Architecture of the API

This topic contains the following sections:

Overview

The Cisco E-Mail Manager API is accessible through the UI/API Server. The components of the UI/API Server involved in the API are:

XML Adapter, which accepts commands and returns responses formatted as XML from a client application.

API Command Handlers, which accepts commands and returns responses from the XML Adapter.

Common Classes, which are used by the API Command Handlers to perform operations and communicate with TServer.

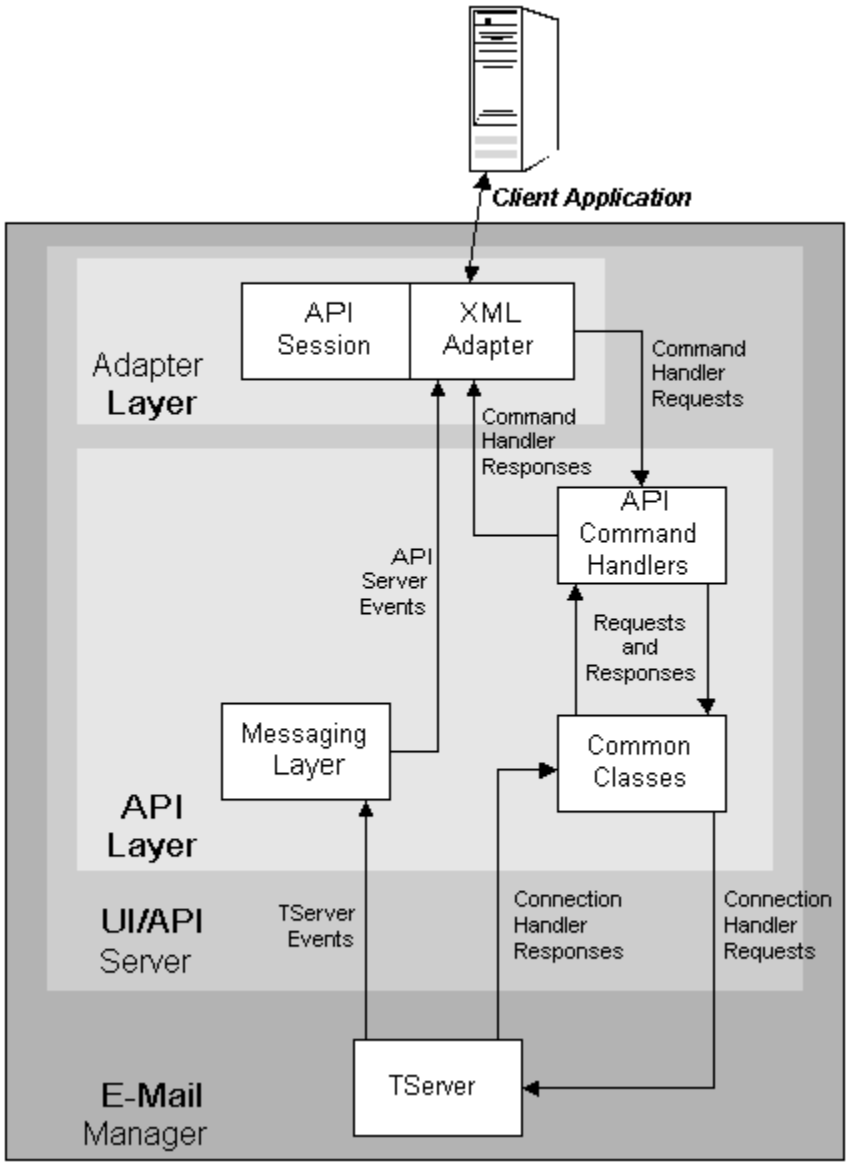
Messaging Layer, which sends events from TServer to the XML Adapter.

For more information about the UI/API Server and general E-Mail Manager architecture, see the *Cisco E-Mail Manager Installation and Configuration Guide*.

For more information about the general E-Mail Manager architecture, see the *Cisco E-Mail Manager Overview Guide*.

Diagram

The following diagram provides an overview of the E-Mail Manager API architecture:



Requirements of the API

This topic contains the following sections:

API Server Installation Requirements

The API Server is installed as part of the UI/API Server. For information about installing the UI/API Server, see the *Cisco E-Mail Manager Installation and Configuration Guide*.

There are no additional installation or configuration requirements for you to be able to use the E-Mail Manager API.

Application Requirements

To build an application that uses the E-Mail Manager API, you do not have to use a specific tool or programming language. The E-Mail Manager API can be used by any application framework that allows you to:

Create and manage connections over sockets or HTTP.

Send XML-formatted commands to the API Server.

Receive and process XML-formatted data from the API Server.

Character Set Requirements

The E-Mail Manager API works with data in the UTF8 character set. The application using the API must send data to the API Server in the UTF8 character set, and must be able to process data received in the UTF8 character set.

Connections to the API Server

Overview of Connections

This topic contains the following sections:

Types of Connections

An application must connect to the server using sockets or HTTP.

Connections and Data Exchange

The connection is used as follows:

The third-party application sends commands over the connection to the API Server.

The API Server sends the following over the connection to the third-party application:

- Responses
- Faults
- Events

Socket Connections

This topic contains the following sections:

Before reading this topic, you should read the topics Overview of Connections and Connection Properties.

Socket Connection Limit

There is a limit of 1000 socket connections to a single UI Server more socket connections can degrade server performance. If an agent is both receiving events and sending commands, that agent is using two socket connections; therefore, if all agents are using commands and receiving events, there is a limit of 500 agents who can connect at one time.

Socket Client Sessions

An application connects to the E-Mail Manager API over a socket client session. A socket client session may consist of one or more command socket connections, and may have one event socket connection.

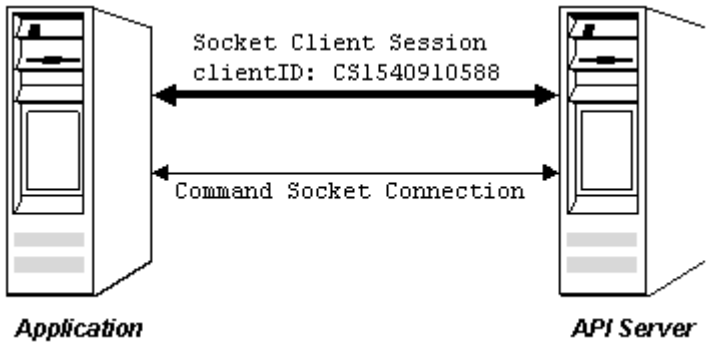
The socket client session is identified by the `clientID` returned to the program at the end of the socket connection protocol.

Command Socket Connections

This section contains the following information:

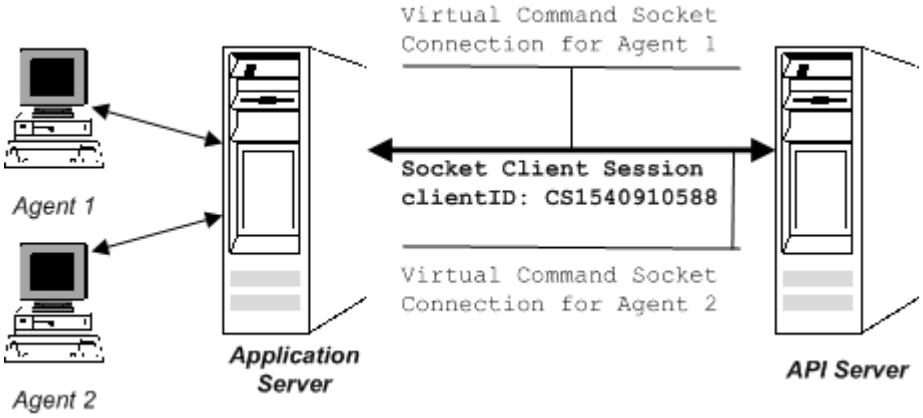
One Command Socket Connection Per Socket Client Session

A command socket connection is the socket connection through which the user sends commands to and receives responses from the API Server. Typically, there is one command socket connection for each socket client session, as shown below:



Multiple Command Socket Connections Per Socket Client Session

It is possible to have one socket client session with multiple virtual command socket connections running inside it. This might be useful in an environment where one server program communicates with the API Server on behalf of multiple users. An example is shown below:



Event Socket Connections

This section contains the following information:

Overview of Event Socket Connections

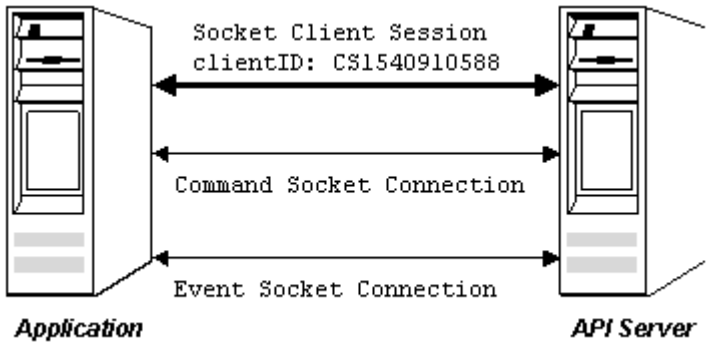
Creating an Event Socket Connection

Closing an Event Socket Connection

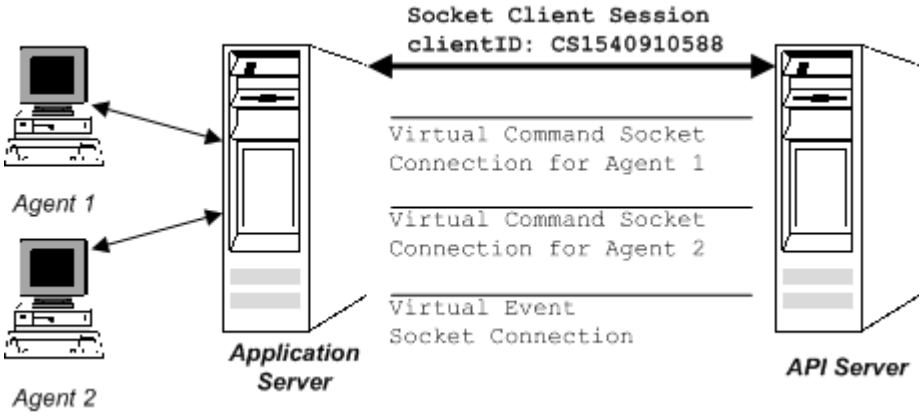
Overview of Event Socket Connections

An event socket connection is the socket connection through which events are sent from the API Server to the application. Each socket client session can have one, and only one, event socket connection.

The following figure shows the event socket connection when only one command socket connection is used:



The following figure shows the event socket connection when multiple command socket connections are used:



Creating an Event Socket Connection

When desired for a socket client session, you should create an event socket connection immediately after you create a command socket connection. Otherwise, any events that occurred before the event socket connection is established are not sent to the application.

Closing an Event Socket Connection

If the event socket connection closes unexpectedly, without the use of the goodbye keyword, the API Server will save events that occur over the next 30 seconds. If a replacement event socket connection is established within 30 seconds, events that were saved are sent over the new event socket. If the replacement event socket is not established within 30 seconds, the events are lost.

Security

The API Server has two mechanisms for controlling security with socket connections:

IP Address Restrictions

PassPhrase validation

IP Address Restrictions

The API Server can restrict socket connections by filtering on the IP address of the computer running the application that tries to connect. The IP address of the computer running the application is automatically sent with the request to create a socket connection. Therefore, you can grant or deny the connection request based on the IP address of client computer.

This section contains the following information:

xmltcpadapater-restrictions File

<RESTRICTIONS> Element

<DENY> and <GRANT> Elements

Using the mask Attribute

Possible Errors in the xmltcpadapter-restrictions.xml File

xmltcpadapter-restrictions.xml File

You define the allowed IP addresses in the xmltcpadapter-restrictions.xml file on the API Server. This file resides in the same directory as the socket adapter property file.

Following is an example of the xmltcpadapter-restrictions.xml file:

```
<RESTRICTIONS defaultAccess="deny">
  <DENY addr="161.45.240.223"/>
  <GRANT addr="161.45.240.0" mask="255.255.255.0"/>
  <GRANT addr="127.0.0.1"/>
</RESTRICTIONS>
```

<RESTRICTIONS> Element

The <RESTRICTIONS> element is the top-level, required element. It has one required attribute, defaultAccess that can have one of the following values:

defaultAccess Value	Description
"deny"	Indicates that all IP addresses are denied a socket connection unless the address is specified in a <GRANT> element.
"grant"	Indicates that all IP addresses are allowed a socket connection unless the address is specified in a <DENY> element.

The <RESTRICTIONS> element can contain multiple <DENY> and <GRANT> elements. It cannot contain other elements.

<DENY> and <GRANT> Elements

The <DENY> and <GRANT> elements specifically identify one or more IP addresses for which to allow socket connections.

The <DENY> and <GRANT> elements have the following attributes:

Attribute	Description
addr	A required attribute that specifies a specific IP address to which to deny or grant a socket connection.
mask	An optional attribute that specifies which bits of the value of the addr attribute are used in a bit-wise AND operation to test the IP addresses trying to create a socket connection. **Note**: The value of the mask attribute is not an IP address.

Note:

When there are multiple <DENY> and <GRANT> element, the first element in which the value of the addr attribute matches the IP address of the computer trying to establish a socket connection is used to grant or deny permission.

If any IP address is specified in an incorrect format, the API Server will refuse to establish socket connections.

Using the mask Attribute

To grant or deny permission to create socket connections to a group of computers, you use the mask attribute. The following examples demonstrate how to use the mask attribute.

In the following example, permission to create socket connections is denied to all IP addresses except those whose first 3 components are 161.45.240. In this example, access is granted to IP address in the range of 161.45.240.0 to 161.45.240.255:

```
<RESTRICTIONS defaultAccess="deny">
  <GRANT addr="161.45.240.0" mask="255.255.255.0"/>
</RESTRICTIONS>
```

In the following example, permission to create socket connections is denied to all IP addresses except those whose first 3 components are 161.45.240 and whose last component is not 233. In this example, the <DENY> element is read first, so the IP address with the last component of 233 is denied access. If the <DENY> and <GRANT> elements were reversed, that IP address would be granted access.

```
<RESTRICTIONS defaultAccess="deny">
  <DENY addr="161.45.240.223"/>
  <GRANT addr="161.45.240.0" mask="255.255.255.0"/>
</RESTRICTIONS>
```

Possible Errors in the xmltcpadapter-restrictions.xml File

The following errors in the xmltcpadapter-restrictions.xml file will prevent the API Server from creating socket connections:

The <RESTRICTIONS> element is missing.

A <DENY> or <GRANT> element is missing the addr attribute.

An addr attribute has an IP address value in an invalid format. For example, if a component of the IP address was greater than 255.

A <DENY> or <GRANT> element has an invalid value for the mask attribute.

The <RESTRICTIONS> element contains an element other than <DENY> or <GRANT>.

PassPhrase Validation

After the API Server checks that the computer trying to establish a socket connection meets the IP address restrictions, it validates the connection with the passphrase. The passphrase is set as a parameter in the xmltcpadapter-props.xml file, and then sent from the client application to the API Server during the socket connection protocol.

If the passphrase sent by the application matches the passphrase defined in the xmltcpadapter-props.xml file, the socket connection is allowed. If the two passphrases do not match, the socket connection is denied.

Socket Client Session Connection Protocol

This section contains the following information:

Overview of the Socket Connection Protocol

Example of Socket Connection Protocol

Keywords and Parameters

Protocol Result Codes

Overview of the Socket Connection Protocol

When you first establish a socket client session connection to the API Server, you must negotiate a simple protocol to indicate the type of socket connection and to supply the passphrase.

Example of the Socket Connection Protocol

The socket connection protocol is a simple dialog between the application and the API Server. In the following example, lines that begin with S are sent from the API Server to the application, and lines that begin with C are sent from the client application to the API Server:

```
S: HELLO SERVER=APIServer__Socket_Adapter VERSION=1.0
C: hello version=1.0 passphrase=the_code clientid=new
S: 100 OK
C: config socket_type=cmd charset=UTF8
S: 100 OK
C: start
S: 100 CLIENTID=CS1540910588
```

Caution: The value of the charset parameter in the config keyword must be UTF8.

Note:

Values of parameters in the dialog are not contained in quotation marks.

The sequence of keywords sent from the application to the API Server must be as shown: (1) hello, (2) config, (3) start.

The following table describes individual lines in this dialog. For further information about the keywords and parameters, see the following section:

Line of Dialog	Explanation	Keyword and Parameters
`S: HELLO SERVER=APIServer__Socket_Adapter VERSION=1.0`	When the API Server first accepts the socket connection, it identifies itself to the application.	Keyword: HELLO Parameters: SERVER VERSION
`C: hello version=1.0 passphrase=the_code clientid=new`	The application specifies the protocol version, the passphrase, and the clientid.	Keyword: hello Parameters: version passphrase clientID
`S: 100 OK`	The API Server responds with the result code.	
`C: config socket_type=cmd charset=UTF8`	The application specifies the type of socket connection to use.	Keyword: config Parameters: socket_type charset
`S: 100 OK`	The API Server responds with the result code.	
`C: start`	The application indicates that it is prepared to begin the session.	Keyword: start
`S: 100 CLIENTID=CS1540910588`	The API Server creates a new Socket Client Session and returns the ID of that session	Parameter: CLIENTID

Keywords and Parameters

Following are explanations of the keywords and parameters used in the socket connection protocol:

Keyword	Description	Parameters
hello	Starts the protocol dialog.	version - specifies the protocol version. In this release of E-Mail Manager, the value must always be *1.0*. Required. passphrase - specifies the passphrase, the code word set on the server to provide security. The passphrase the application sends to the server must match the passphrase set in the xmltcpadapter-props.xml file. Required. clientid - specifies the socket client identifier. To use an existing socket client session, use the value returned by that previous session; to create a new session, use the value *new*. Required. **Note**: The clientid is not the same as the sessID returned by the agentConnect and agentDisconnectDuplicateAndConnect commands.
config	Specifies the kind of socket connection to establish.	socket_type - specifies the type of socket connection. For commands, use the value *cmd*; for events, use the value *event*. Required**.** format - specifies the format of the data exchanged between the API Server and the application. For this release of E-Mail Manager, the only allowed value is *xml*. Optional; the value *xml* is assumed if not specified. charset - specifies the character set used for data exchanged between the API Server and the application. You must specify UTF8 as the character set.
start	Specifies that the application is ready to send commands or receive events.	None
goodbye	Specifies that the socket connection is to be closed. When receiving this keyword, the API Server closes the socket.	None

Protocol Result Codes

During the dialog, the API Server responds to each input with a protocol result code. The following table lists and describes the possible result codes:

Code	Description
100 OK	The input from the application was processed successfully.
600	The format requested is not supported. In this release of E-Mail Manager, the only format supported is *xml*.
601	The value of the clientid parameter is not valid.
602	The value of the socket_type parameter is not valid.
603	Reserved.
604	An event socket already exists. You can only have one active event socket.
605	The protocol version is incompatible. In this release of E-Mail Manager, the value of the version parameter must be *1.0*.
606	The protocol dialog sequence is not in order. The order of keywords sent to the API Server must be: hello, config, start.
607	The command is not valid.
608	The parameter is not valid.
609	The maximum number of allowed socket connections has been reached.
610	The character set requested in the charset parameter is not supported.

HTTP Connections

This topic contains the following information:

Before reading this topic, you should read the topics Overview of Connections and Connection Properties.

Commands over HTTP

This section contains the following information:

apicommander Servlet

You post commands over an HTTP connection to the apicommander servlet running on the API Server. The apicommander servlet accepts incoming commands and returns responses and faults.

The URL

You post commands over HTTP to the apicommander servlet at the following URL:

`http://server-name:port-number/uiroot/apicommander`

Content-type

You post commands over an HTTP connection with the content-type in the HTTP header set to "text/xml". The API Server returns responses and faults with the content-type set to "text/xml" as well.

Time Out

By default, a session connected to the API Server with an HTTP connection automatically times out after being idle for 30 minutes. You can customize the time-out period through the connection properties.

Events over HTTP

This section contains the following information:

- apilongpoll Servlet
- The URL
- Parameters
- Polling for Events Over HTTP
- NOOP Events

You should also read the topic on long poll server properties.

apilongpoll Servlet

You receive events over an HTTP connection by polling the apilongpoll servlet using HTTP GET requests.

The URL

You poll for events over HTTP to the apilongpoll servlet at the following URL:

`http://server-name:port-number/uiroot/apilongpoll`

Parameters

You must include two parameters when polling the apilongpoll servlet:

Parameter	Description
`sessID`	The identifier for the session. A `sessID` is returned by the `agentConnect` and `agentDisconnectDuplicateAndConnect` commands.
`lastEvent`	The value of the `clientID` attribute of the last event that the

application received for the specified `sessID`. The application must save the value of the `cliendID` attribute of the last event, to reuse it as the lastEvent parameter when polling for subsequent events.

Events received are numbered sequentially starting from 0 for each session.

If the application has not received any events for the `sessID`, the value of the `lastEvent` parameter should be `-1`.

Polling for Events Over HTTP

You poll for events over HTTP by sending HTTP requests to the apilongpoll servlet URL with the necessary parameters. For example, the following URL polls for events for the sessID 1234, when the last event received was identified as 7:

```
http://server-name:port-number/uiroot/apilongpoll?sessID=1234&lastEvent=7
```

NOOP Events

The apilongpoll servlet periodically sends NOOP events to the application to keep the connection active, in the following format:

```
<APIEnvelope type = "NOOP">
</APIEnvelope>
```

Connection Properties

This topic contains the following sections:

Overview

You configure several properties on the API Server that determine how socket and HTTP connections are handled. Because most of these properties are common to both types of connections, they are described together in this topic, and not in the topic specific to the type of connection.

HTTP Connection Properties

HTTP connection properties are set in the file xmlhttpadapter-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

A sample of this file is shown below:

```
<PROPERTIES>
    <!-- validate commands using the DTD -->
<PROPERTY name="dtdValidation" value="true" />
<!-- should we produce a stack trace in each fault object -->
<PROPERTY name="stackTraceOnFault" value="false" />
<!-- session timeout (in ms) -->
<PROPERTY name="sessionTimeout" value="1800000" />
    <!-- long poll settings - each longpoll servlet has its own
settings:
         available settings are:
            * [servletname].eventQCheck - how often longpoll servlets
should check
              their queues while connected (milliseconds) (default
500ms)
            * [servletname].noopTimeout - how often noop messages
              should be sent to the client (milliseconds) (default
10000ms)
            * [servletname].maxEvents - the maximum number of events
              that should be written to the client.  This parameter
```

```
              only works when connected to the LongPollServer.  -1
              means no limit.  Default is -1.
            * [servletname].longPollTimeout - how long should
connections remain open
  (milliseconds) (default 90000ms)
* [servletname].connectToLongPollServer - should this longpoll be
available
  from a longpoll server (default true)
            * [servletname].longPollServerConectMethod - either
"java" or "lpsp" - how
  should the longpoll server be contacted (default java)
            * [servletname].longPollServerAddress - if
longPollServerConnectMethod is
              "lpsp", the hostname/port to make the connection on
            * [servletname].longPollServerPropDir - if
longPollServerConnectMethod is
              "java", the property directory to initialize the
LongPoll Server (defaults
              to the same directory as the API Server).
            * [servletname].longPollServerEntryPointPort - the port
the LongPoll Server
              should listen for requests on (default 5831)
            * [servletname].longPollServerEntryPointUseSSL - should
              the LongPoll Server use HTTPS for this entry point
              (default false).
            * [servletname].longPollServerEntryPointPath - the path
the LongPoll Server
              should look for requests for this adapter (default
/longpoll)
    -->
    <PROPERTY name="apiLongPoll.eventQCheck" value="500" />
    <PROPERTY name="apiLongPoll.noopTimeout" value="10000" />
    <PROPERTY name="apiLongPoll.maxEvents" value="-1" />
<PROPERTY name="apiLongPoll.longPollTimeout" value="90000" />
<PROPERTY name="apiLongPoll.connectToLongPollServer" value="true" />
<PROPERTY name="apiLongPoll.longPollServerConnectMethod" value="java"
/>
<PROPERTY name="apiLongPoll.longPollServerAddress" value="" />
<PROPERTY name="apiLongPoll.longPollServerPropDir" value="" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointPort"
value="5831" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointUseSSL"
value="false" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointPath"
value="/apilongpoll" />
</PROPERTIES>
```

Socket Connection Properties

Socket connection properties are set in the file xmltcpadapter-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

A sample of this file is shown below:

```
<PROPERTIES>
<PROPERTY name="loadSocketServer" value="true" />
<PROPERTY name="port" value="1441" />
<PROPERTY name="maxClientSessions" value="-1" />
<PROPERTY name="dtdValidation" value="true" />
<PROPERTY name="passPhrase" value="launch_it" />
<PROPERTY name="sessionTimeout" value="-1" />
<PROPERTY name="hdlrWriterThreadWaitTimeout" value="1800" />
<PROPERTY name="stackTraceOnFault" value="false" />
</PROPERTIES>
```

Property Descriptions

The following table lists and describes the properties that affect connections, and specify which types of connection the properties apply to:

Property	Description	Default Value	Types of Connection Effected	Files Used In
dtdValidation	Specifies whether incoming commands should be validated using the DTD.	true	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
sessionTimeout	Specifies the timeout period for connections to the API Server. The value is in milliseconds. If the connection is not used within the timeout period, either through a command or event polling, the connection is terminated. To specify no timeout period, use -1.	HTTP: 1800000 Socket: -1	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml

stackTraceOnFault	Specifies whether to write a stack trace in the <Details> element in a fault.	false	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
i18n.dfltLocale.language i18n.dfltLocale.country i18n.dfltLocale.variant i18n.dfltTimeZone i18n.dfltCharSet	Specifies the local language, country, variant, time zone, and character set to use for the connection. **Note**: If these properties are not included, the values for these properties in the apiserver_props.xml file are used by default. The i18n.dftlLocale.country property requires a two-letter country codes as a value.	The values in the apiserver_props.xml file. These parameters are not included by default in the files xmlhttpadapter-props.xml and xmltcpadapter-props.xml	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
port	The port number that the socket server should listen on.	1441	Socket	xmltcpadapter-props.xml
maxClientSessions	The maximum number of socket connections allowed. -1 indicates no maximum.	-1	Socket	xmltcpadapter-props.xml
passPhrase	The string that functions as a password for socket connections. The same string must	A default passphrase is included in the file, but this value	Socket	xmltcpadapter-props.xml

	be sent to the API Server during the socket connection protocol dialog.	should not be used. **Caution**: For security purposes, you should change this default following installation.		
hdlrWriterThreadWaitTimeout	The timeout period, in seconds, that the socket should keep the writer thread active. If there are no commands or events sent over the connection in the timeout period, the thread is terminated, to be restarted when next needed.	1800 (30 minutes)	Socket	xmltcpadapter-props.xml

Long Poll Server Properties

This topic contains the following sections:

Location

API Server properties are set in the file longpollserver-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

Sample File

A sample of this file is shown below:

```
<PROPERTIES>
<!-- log manager properties.  where to find the property directory, and what
configuration to use -->
    <PROPERTY name="logManager.configDirectory"
value="common/apiserver/logManager" />
<PROPERTY name="logManager.configName" value="APIServer" />
<!-- Timer information.  This setting controls the granularity of
     the timer thread.  This thread is responsible for scheduling
     noop events and disconnect events.  This value controls how
     granular that timer is.  Lowering it significantly decreases
     performance, especially on multiprocessor machines.  Events
     will be scheduled to run within this time of their real time.
     Time in ms.  Default is 100.  -->
<PROPERTY name="timerGranularity" value="100" />
<!-- Thread information.  This allows the ThreadPool to be tuned.
         Three parameters are supported:
           * minThreads - the minimum number of threads to keep
             running.  The number or running threads will never drop
             below this number.
           * maxThreads - the maximum number of threads to keep
             running.  If a request is ready, and no thread is
             available, one will be started if this threshold has not
             been reached.  Default is 10.
           * idleTime - how long a thread should be allowed to remain
             idle before exiting.  If a thread is idle for this period
             of time, it will exit.  Time in ms.  Default is 5s.
```

```
    -->

<PROPERTY name="minThreads" value="5" />
<PROPERTY name="maxThreads" value="10" />
<PROPERTY name="idleTime" value="5000" />
    <!-- SSL information.  If we're supporting SSL (either for HTTPS
         or LPSP/SSL) these settings must be correct:
           * initializeSSL - should SSL be initialized.  Either "true"
             or "false".  If set to "false" SSL will not be available.
             If set to "true", the LongPoll Server will attempt to
             initialize SSL.  Default is "false".
           * sslInitializer - the name of the class to use to
             initialize SSL.  This class will load all appropriate
             classes and return a ServerSocketFactory and a
             SocketFactory for SSL sockets.  This will allow multiple
             versions of SSL to be used.  The default value of
             "com.cisco.ics.inf.longpoll.JSSEIntializer" will
             initialize SSL using JSSE.
           * keyStoreType - the type of keystore to use.  Should be
             either JKS or PKCS12.  See the "Java(tm) Cryptography
             Architecture API Specification & Reference" (available from
             http://java.sun.com/j2se/1.3/docs/guide/security/CryptoSpec.html)
             for more information about supported types.  Defaults to "JKS".
           * keyStore - the name of the file to use for a keystore.
             NOTE: for security, this is an absolute path.  No default.
           * keyStorePassword - the password to use when loading the
             keystore.  No default.
           * keyPassword - the password to use when loading the keys.
             No default.
    -->
    <PROPERTY name="initializeSSL" value="false" />
    <PROPERTY name="sslInitializer"
value="com.cisco.ics.inf.longpoll.JSSEInitializer" />
    <PROPERTY name="keyStoreType" value="JKS" />
    <PROPERTY name="keyStore" value="" />
    <PROPERTY name="keyStorePassword" value="" />
    <PROPERTY name="keyPassword" value="" />
</PROPERTIES>
```

Property Descriptions

The following table lists and describes the properties used in the longpollserver_props.xml file:

Property	Description	Default Value
logManager.configDirectory	The location of Log Manager configuration files.	common/apiserver/logManager (relative to the installation-folder\uiroot\WEB-INF\properties folder)
logManager.configName	The Log Manager configuration to use.	APIServer
timerGranularity	The granularity of the timer thread, in milliseconds. Lowering this value decreases performance, especially on multiprocessor servers.	100
minThreads	The minimum number of threads to keep running.	5
maxThreads	The maximum number of threads to keep running.	10
idleTime	The time, in milliseconds, that a thread can remain idle before exiting.	5
initializeSSL	Whether or not SSL should be initialized.	false
sslInitializer	The name of the class to use to initialize SSL.	com.cisco.ics.inf.longpoll.JSSEInitializer

keyStoreType	The type of keystore to use. The value should be either 'JKS' or 'PKCS12'. For more information, see "Java(tm) Cryptography Architecture API Specification & Reference".	null ('JKS' is used when no value is specified)
keyStorePassword	The password to use when loading the keystore.	null
keyPassword	The password to use when loading the keys.	null

API Server Properties

API Server Properties

This topic contains the following sections:

Location

API Server properties are set in the file apiserver-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

Sample File

A sample of this file is shown below:

```
<PROPERTIES>
<!-- API Server ID - must be between 0 and 16383 -->
<PROPERTY name="serverID" value="0" />

    <!-- the name of the file to load commands from -->
<PROPERTY name="cmdFile" value="apiserver-commands.xml" />
    <!-- how often to run a cache cleanout (in ms) -->
    <PROPERTY name="cache.cleanout-interval" value="60000" />
    <!-- settings for the default locale, timezone, character sets -
         used to create new UserConnections -->
<PROPERTY name="i18n.dfltLocale.language" value="en" />
<PROPERTY name="i18n.dfltLocale.country" value="US" />
<PROPERTY name="i18n.dfltLocale.variant" value="" />
<PROPERTY name="i18n.dfltTimeZone" value="" />
<PROPERTY name="i18n.dfltCharSet" value="" />
<!-- log manager properties.  where to find the property directory,
and what configuration to use -->
    <PROPERTY name="logManager.configDirectory"
value="common/apiserver/logManager" />
<PROPERTY name="logManager.configName" value="APIServer" />
<!-- debug properties.  these properties should be left at their
default values -->
<!-- unless debugging, changing them may have severe performance and
functionality impacts -->
<!-- turning this on will make resource bundles be reloaded each time
APIServer.getBundle is called -->
```

```
<PROPERTY name="debug.disableResourceBundleCache" value="false" />
<!-- turning this on will wrap resource bundles retrieved by
APIServer.getBundle by DebugResourceBundle objects -->
<PROPERTY name="debug.useDebugResourceBundle" value="false" />

</PROPERTIES>
```

Property Descriptions

The following table lists and describes the properties used in the apiserver_props.xml file:

Property	Description	Default Value
serverID	The unique identifier of the UI/API Server. This must be a value between 0 and 16383.	0
cmdFile	The name of the file in which the API commands are defined.	apiserver-commands.xml
cache.cleanout-interval	The frequency, in milliseconds, to clean out the cache.	60000
i18n.dfltLocale.language	The default language.	en
i18n.dfltLocale.country	The default country. The i18n.dftlLocale.country property requires a two-letter country codes as a value.	US
i18n.dfltLocale.variant	The locale variant.	null
i18n.dfltTimeZone	The default time zone.	null
i18n.dfltCharSet	The default character set.	null

logManager.configDirectory	The location of Log Manager configuration files.	common/apiserver/logManager (relative to the installation-folder\uiroot\WEB-INF\properties folder)
logManager.configName	The Log Manager configuration to use.	APIServer
debug.disableResourceBundleCache	Whether or not to have resource bundles loaded each time APIServer.getBundle is called. **Caution**: You should not modify the default value; making the value 'true' can have severe performance implications.	false
debug.useDebugResourceBundle	Whether or not to wrap resource bundles retrieved by APIServer.getBundle in DebugResourceBundle objects. **Caution**: You should not modify the default value; making the value 'true' can have severe performance implications.	false

Two-Letter Country Codes

The following list shows the standard two-letter country codes to use as values of the i18n.dfltLoacale.country property:

```
aa Afar
ab Abkhazian
af Afrikaans
am Amharic
```

```
ar Arabic
as Assamese
ay Aymara
az Azerbaijani
ba Bashkir
be Byelorussian
bg Bulgarian
bh Bihari
bi Bislama
bn Bengali; Bangla
bo Tibetan
br Breton
ca Catalan
co Corsican
cs Czech
cy Welsh
da Danish
de German
dz Bhutani
el Greek
en English
eo Esperanto
es Spanish
et Estonian
eu Basque
fa Persian
fi Finnish
fj Fiji
fo Faroese
fr French
fy Frisian
ga Irish
gd Scots Gaelic
gl Galician
gn Guarani
gu Gujarati
ha Hausa
he Hebrew (formerly iw)
hi Hindi
hr Croatian
hu Hungarian
hy Armenian
ia Interlingua
id Indonesian (formerly in)
ie Interlingue
ik Inupiak
is Icelandic
it Italian
iu Inuktitut
ja Japanese
jw Javanese
ka Georgian
kk Kazakh
kl Greenlandic
km Cambodian
kn Kannada
ko Korean
```

```
ks Kashmiri
ku Kurdish
ky Kirghiz
la Latin
ln Lingala
lo Laothian
lt Lithuanian
lv Latvian, Lettish
mg Malagasy
mi Maori
mk Macedonian
ml Malayalam
mn Mongolian
mo Moldavian
mr Marathi
ms Malay
mt Maltese
my Burmese
na Nauru
ne Nepali
nl Dutch
no Norwegian
oc Occitan
om (Afan) Oromo
or Oriya
pa Punjabi
pl Polish
ps Pashto, Pushto
pt Portuguese
qu Quechua
rm Rhaeto-Romance
rn Kirundi
ro Romanian
ru Russian
rw Kinyarwanda
sa Sanskrit
sd Sindhi
sg Sangho
sh Serbo-Croatian
si Sinhalese
sk Slovak
sl Slovenian
sm Samoan
sn Shona
so Somali
sq Albanian
sr Serbian
ss Siswati
st Sesotho
su Sundanese
sv Swedish
sw Swahili
ta Tamil
te Telugu
tg Tajik
th Thai
ti Tigrinya
```

```
tk Turkmen
tl Tagalog
tn Setswana
to Tonga
tr Turkish
ts Tsonga
tt Tatar
tw Twi
ug Uighur
uk Ukrainian
ur Urdu
uz Uzbek
vi Vietnamese
vo Volapuk
wo Wolof
xh Xhosa
yi Yiddish (formerly ji)
yo Yoruba
za Zhuang
zh Chinese
zu Zulu
```

Connection Properties

This topic contains the following sections:

Overview

You configure several properties on the API Server that determine how socket and HTTP connections are handled. Because most of these properties are common to both types of connections, they are described together in this topic, and not in the topic specific to the type of connection.

HTTP Connection Properties

HTTP connection properties are set in the file xmlhttpadapter-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

A sample of this file is shown below:

```
<PROPERTIES>
    <!-- validate commands using the DTD -->
<PROPERTY name="dtdValidation" value="true" />
<!-- should we produce a stack trace in each fault object -->
<PROPERTY name="stackTraceOnFault" value="false" />
<!-- session timeout (in ms) -->
<PROPERTY name="sessionTimeout" value="1800000" />
    <!-- long poll settings - each longpoll servlet has its own
settings:
         available settings are:
            * [servletname].eventQCheck - how often longpoll servlets
should check
              their queues while connected (milliseconds) (default
500ms)
            * [servletname].noopTimeout - how often noop messages
              should be sent to the client (milliseconds) (default
10000ms)
            * [servletname].maxEvents - the maximum number of events
              that should be written to the client.  This parameter
```

```
              only works when connected to the LongPollServer.  -1
              means no limit.  Default is -1.
            * [servletname].longPollTimeout - how long should
connections remain open
  (milliseconds) (default 90000ms)
* [servletname].connectToLongPollServer - should this longpoll be
available
  from a longpoll server (default true)
            * [servletname].longPollServerConectMethod - either
"java" or "lpsp" - how
  should the longpoll server be contacted (default java)
            * [servletname].longPollServerAddress - if
longPollServerConnectMethod is
              "lpsp", the hostname/port to make the connection on
            * [servletname].longPollServerPropDir - if
longPollServerConnectMethod is
              "java", the property directory to initialize the
LongPoll Server (defaults
              to the same directory as the API Server).
            * [servletname].longPollServerEntryPointPort - the port
the LongPoll Server
              should listen for requests on (default 5831)
            * [servletname].longPollServerEntryPointUseSSL - should
              the LongPoll Server use HTTPS for this entry point
              (default false).
            * [servletname].longPollServerEntryPointPath - the path
the LongPoll Server
              should look for requests for this adapter (default
/longpoll)
    -->
    <PROPERTY name="apiLongPoll.eventQCheck" value="500" />
    <PROPERTY name="apiLongPoll.noopTimeout" value="10000" />
    <PROPERTY name="apiLongPoll.maxEvents" value="-1" />
<PROPERTY name="apiLongPoll.longPollTimeout" value="90000" />
<PROPERTY name="apiLongPoll.connectToLongPollServer" value="true" />
<PROPERTY name="apiLongPoll.longPollServerConnectMethod" value="java"
/>
<PROPERTY name="apiLongPoll.longPollServerAddress" value="" />
<PROPERTY name="apiLongPoll.longPollServerPropDir" value="" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointPort"
value="5831" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointUseSSL"
value="false" />
<PROPERTY name="apiLongPoll.longPollServerEntryPointPath"
value="/apilongpoll" />
</PROPERTIES>
```

Socket Connection Properties

Socket connection properties are set in the file xmltcpadapter-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

A sample of this file is shown below:

```
<PROPERTIES>
<PROPERTY name="loadSocketServer" value="true" />
<PROPERTY name="port" value="1441" />
<PROPERTY name="maxClientSessions" value="-1" />
<PROPERTY name="dtdValidation" value="true" />
<PROPERTY name="passPhrase" value="launch_it" />
<PROPERTY name="sessionTimeout" value="-1" />
<PROPERTY name="hdlrWriterThreadWaitTimeout" value="1800" />
<PROPERTY name="stackTraceOnFault" value="false" />
</PROPERTIES>
```

Property Descriptions

The following table lists and describes the properties that affect connections, and specify which types of connection the properties apply to:

Property	Description	Default Value	Types of Connection Effected	Files Used In
dtdValidation	Specifies whether incoming commands should be validated using the DTD.	true	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
sessionTimeout	Specifies the timeout period for connections to the API Server. The value is in milliseconds. If the connection is not used within the timeout period, either through a command or event polling, the connection is terminated. To specify no timeout period, use -1.	HTTP: 1800000 Socket: -1	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml

stackTraceOnFault	Specifies whether to write a stack trace in the <Details> element in a fault.	false	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
i18n.dfltLocale.language i18n.dfltLocale.country i18n.dfltLocale.variant i18n.dfltTimeZone i18n.dfltCharSet	Specifies the local language, country, variant, time zone, and character set to use for the connection. **Note**: If these properties are not included, the values for these properties in the apiserver_props.xml file are used by default. The i18n.dftlLocale.country property requires a two-letter country codes as a value.	The values in the apiserver_props.xml file. These parameters are not included by default in the files xmlhttpadapter-props.xml and xmltcpadapter-props.xml	HTTP Socket	xmlhttpadapter-props.xml xmltcpadapter-props.xml
port	The port number that the socket server should listen on.	1441	Socket	xmltcpadapter-props.xml
maxClientSessions	The maximum number of socket connections allowed. -1 indicates no maximum.	-1	Socket	xmltcpadapter-props.xml
passPhrase	The string that functions as a password for socket connections. The same string must	A default passphrase is included in the file, but this value should not be used.	Socket	xmltcpadapter-props.xml

	be sent to the API Server during the socket connection protocol dialog.	**Caution**: For security purposes, you should change this default following installation.		
hdlrWriterThreadWaitTimeout	The timeout period, in seconds, that the socket should keep the writer thread active. If there are no commands or events sent over the connection in the timeout period, the thread is terminated, to be restarted when next needed.	1800 (30 minutes)	Socket	xmltcpadapter-props.xml

Long Poll Server Properties

This topic contains the following sections:

Location

API Server properties are set in the file longpollserver-props.xml located on the API Server at:

installation-folder\uiroot\WEB-INF\properties

Sample File

A sample of this file is shown below:

```
<PROPERTIES>
<!-- log manager properties.  where to find the property directory, and what
configuration to use -->
    <PROPERTY name="logManager.configDirectory"
value="common/apiserver/logManager" />
<PROPERTY name="logManager.configName" value="APIServer" />
<!-- Timer information.  This setting controls the granularity of
     the timer thread.  This thread is responsible for scheduling
     noop events and disconnect events.  This value controls how
     granular that timer is.  Lowering it significantly decreases
     performance, especially on multiprocessor machines.  Events
     will be scheduled to run within this time of their real time.
     Time in ms.  Default is 100.  -->
<PROPERTY name="timerGranularity" value="100" />
<!-- Thread information.  This allows the ThreadPool to be tuned.
         Three parameters are supported:
           * minThreads - the minimum number of threads to keep
             running.  The number or running threads will never drop
             below this number.
           * maxThreads - the maximum number of threads to keep
             running.  If a request is ready, and no thread is
             available, one will be started if this threshold has not
             been reached.  Default is 10.
           * idleTime - how long a thread should be allowed to remain
             idle before exiting.  If a thread is idle for this period
             of time, it will exit.  Time in ms.  Default is 5s.
```

```
    -->

<PROPERTY name="minThreads" value="5" />
<PROPERTY name="maxThreads" value="10" />
<PROPERTY name="idleTime" value="5000" />
    <!-- SSL information.  If we're supporting SSL (either for HTTPS
         or LPSP/SSL) these settings must be correct:
           * initializeSSL - should SSL be initialized.  Either "true"
             or "false".  If set to "false" SSL will not be available.
             If set to "true", the LongPoll Server will attempt to
             initialize SSL.  Default is "false".
           * sslInitializer - the name of the class to use to
             initialize SSL.  This class will load all appropriate
             classes and return a ServerSocketFactory and a
             SocketFactory for SSL sockets.  This will allow multiple
             versions of SSL to be used.  The default value of
             "com.cisco.ics.inf.longpoll.JSSEIntializer" will
             initialize SSL using JSSE.
           * keyStoreType - the type of keystore to use.  Should be
             either JKS or PKCS12.  See the "Java(tm) Cryptography
             Architecture API Specification & Reference" (available from
             http://java.sun.com/j2se/1.3/docs/guide/security/CryptoSpec.html)
             for more information about supported types.  Defaults to "JKS".
           * keyStore - the name of the file to use for a keystore.
             NOTE: for security, this is an absolute path.  No default.
           * keyStorePassword - the password to use when loading the
             keystore.  No default.
           * keyPassword - the password to use when loading the keys.
             No default.
    -->
    <PROPERTY name="initializeSSL" value="false" />
    <PROPERTY name="sslInitializer"
value="com.cisco.ics.inf.longpoll.JSSEInitializer" />
    <PROPERTY name="keyStoreType" value="JKS" />
    <PROPERTY name="keyStore" value="" />
    <PROPERTY name="keyStorePassword" value="" />
    <PROPERTY name="keyPassword" value="" />
</PROPERTIES>
```

Property Descriptions

The following table lists and describes the properties used in the longpollserver_props.xml file:

Property	Description	Default Value
logManager.configDirectory	The location of Log Manager configuration	common/apiserver/logManager (relative to the installation-folder\uiroot\WEB-INF\properties

	files.	folder)
logManager.configName	The Log Manager configuration to use.	APIServer
timerGranularity	The granularity of the timer thread, in milliseconds. Lowering this value decreases performance, especially on multiprocessor servers.	100
minThreads	The minimum number of threads to keep running.	5
maxThreads	The maximum number of threads to keep running.	10
idleTime	The time, in milliseconds, that a thread can remain idle before exiting.	5
initializeSSL	Whether or not SSL should be initialized.	false
sslInitializer	The name of the class to use to initialize SSL.	com.cisco.ics.inf.longpoll.JSSEInitializer
keyStoreType	The type of keystore to use. The value	null ('JKS' is used when no value is

	should be either 'JKS' or 'PKCS12'. For more information, see "Java(tm) Cryptography Architecture API Specification & Reference".	specified)
keyStorePassword	The password to use when loading the keystore.	null
keyPassword	The password to use when loading the keys.	null

XML Standards

API DTD

Following is the DTD used by the API server for a:

Command

Response

Fault

Event

```
<!--
   message.xml - a dtd for UIServer's Command, Response, Fault Event
XML
-->
<!-- CMD -->
<!ELEMENT Cmd  (Param*)>
<!ATTLIST Cmd  name    NMTOKEN  #REQUIRED >
<!ATTLIST Cmd  version NMTOKEN  #REQUIRED >
<!ATTLIST Cmd  serviceDomain NMTOKEN  #IMPLIED >
<!-- CMDRESPONSE -->
<!ELEMENT CmdResponse  (Param*) >
<!ATTLIST CmdResponse  name    NMTOKEN  #REQUIRED >
<!ATTLIST CmdResponse  version NMTOKEN  #REQUIRED >
<!ATTLIST CmdResponse  serviceDomain NMTOKEN  #IMPLIED >
<!-- CMDFAULT -->
<!ELEMENT CmdFault  (ErrString, Details?) >
<!ATTLIST CmdFault  version  NMTOKEN  #REQUIRED >
<!ATTLIST CmdFault  serviceDomain NMTOKEN  #IMPLIED >
<!ATTLIST CmdFault  code     NMTOKEN  #REQUIRED >
<!ATTLIST CmdFault  category NMTOKEN  #REQUIRED >
<!ELEMENT ErrString  (#PCDATA )>
<!ELEMENT Details    (#PCDATA )>
<!-- EVENT -->
<!ELEMENT Event (Param*)>
<!ATTLIST Event  name    NMTOKEN  #REQUIRED >
<!ATTLIST Event  version NMTOKEN  #REQUIRED >
<!ATTLIST Event  serviceDomain NMTOKEN  #IMPLIED >
<!ATTLIST Event  clientId NMTOKEN  #IMPLIED >
<!-- Stuff in Common -->
<!ELEMENT Param  (Value | Struct | Array )>
<!ATTLIST Param  name NMTOKEN  #REQUIRED >
<!ELEMENT Value  (#PCDATA )>
<!ELEMENT Struct  (Param*)>
<!ELEMENT Array  (Value | Struct | Array )+>
```

The <APIEnvelope> Element

This topic contains the following sections:

Overview

All commands, responses, events and faults that are exchanged between the third-party application and the E-Mail Manager API are wrapped in the <APIEnvelope> element.

This topic describes the <APIEnvelope> element itself.

See Also:

Contents of the <APIEnvelope> Element.

Attributes

The <APIEnvelope> element can take the following attributes:

```
sessID

checkCache

allowResultCache

type

Attributes for internationalization
```

sessID

The sessID attribute is used to identify a session with the API server.

To create a new session, use the value "new":

```
<APIEnvelope sessID = "new">
  . . .
  . . .
</APIEnvelope>
```

When sending commands for a previously established session, use the `sessID` value returned when you made the initial connection with the agentConnect or agentDisconnectDuplicateAndConnect command:

```
<APIEnvelope sessID = "5684">
  . . .
  . . .
</APIEnvelope>
```

checkCache

The checkCache attribute is an optional attribute used to indicate whether the API Server should check for an already cached response in the API Server cache.

Use the value "`true`" if the cache should be checked:

```
<APIEnvelope sessID = "5684" checkCache = "true">
  . . .
  . . .
</APIEnvelope>
```

Use the value "`false`" if the cache should not be checked:

```
<APIEnvelope sessID = "5684" checkCache = "false">
  . . .
  . . .
</APIEnvelope>
```

The default value is "`true`"; if the checkCache attribute is not included, the API server checks the cache.

allowResultCache

The `allowResultCache` attribute is an optional attribute used to indicate whether the API Server should attempt to store the response to the command in the cache.

Use the value "true" if the response should be cached:

```
<APIEnvelope sessID = "5684" allowResultCache = "true">
  . . .
  . . .
</APIEnvelope>
```

Use the value "false" if the response should not be cached:

```
<APIEnvelope sessID = "5684" allowResultCache = "false">
  . . .
  . . .
</APIEnvelope>
```

The default value is "true"; if the allowResultCache attribute is not included, the API server puts the response in cache.

type

The type attribute is an optional attribute used to indicate what the content of the XML is.

Possible values are:

"cmd", for a command

```
<APIEnvelope sessID = "5684" type = "cmd">
  . . .
  . . .
</APIEnvelope>
```

"response" for a response

```
<APIEnvelope sessID = "5684" type = "response">
  . . .
  . . .
</APIEnvelope>
```

"event" for an event

```
<APIEnvelope sessID = "5684" type = "event">
  . . .
  . . .
</APIEnvelope>
```

The default value is "cmd"; if the type attribute is not included, the API server assumes the XML content is a command.

Note: The type attribute can have different values in an empty <APIEnvelope> element.

Attributes for Internationalization

The following optional attributes are used to specify localization properties for the session:

uis_Language, the language used by the user.

uis_Country, the country the user is located in.

uis_TimeZone, the time zone the user is located in.

uis_Variant, used to support a local language variant, for example.

Note: You do not specify the character set in the <APIEnvelope> element. The character set is always UTF8. For Socket connections, UTF8 is specified in the Socket Connection Protocol. For HTTP connections, UTF8 is used automatically.

For example, to create a new session for a user using French language, the <APIEnvelope> element would be as follows:

```
<APIEnvelope sessID = "new" uis_Language = "fr">
  . . .
  . . .
</APIEnvelope>
```

These internationalization-related parameters are also set either in the connection properties files or the API properties file. The settings in these files can be overridden by including these attributes in the <APIEnvelope> element.

Empty APIEnvelope Elements

Empty APIEnvelope elements are used to manage sessions.

An empty APIEnvelope element must have a type attribute with one of the following values:

`createSession`, to create a new session:

```
<APIEnvelope sessID = "new" type = "createSession">
</APIEnvelope>
```

`deleteSession`, to delete an existing session:

```
<APIEnvelope sessID = "5463" type = "deleteSession">
</APIEnvelope>
```

noop, to ping the API Server so the timeout clock is refreshed.

```
<APIEnvelope sessID = "5463" type = "noop">
</APIEnvelope>
```

APIEnvelope Elements and New Lines

The APIEnvelope start and end tags must be on their own lines. A new line is required after the APIEnvelope start tag, as well as before and after the APIEnvelope end tag.

A new line is defined as "\r\n" The following example shows where new lines are required:

```
<APIEnvelope sessID = "new"> \r\n
  <Cmd version="1.0" name="example-command">
  . . .
  </Cmd>\r\n
</APIEnvelope>\r\n
```

Contents of the <APIEnvelope> Element

This topic contains the following sections:

Top-Level Elements

All data within an <APIEnvelope> element contain one of the following elements:

Element	Use
<Cmd>	Commands
<CmdResponse>	Successful Responses
<Event>	Events
<CmdFault>	Faults

Contents of Top-Level Elements

Commands (<Cmd> elements), responses (<CmdResponse> elements), events (<Event> elements), and faults (<CmdFault> elements) contain the following attributes and elements:

<Array> Element

<Struct> Element

Caution: Neither XML declarations (i.e. <?xml . . . ?>) nor DocType declarations ("<!DOCTYPE. . .>") are allowed in the APIEnvelope element.

version Attribute

In this release of E-Mail Manager, the value of the `version` attribute is always "1.0". The `version` is a required attribute.

name Attribute

The value of the `name` attribute is the name of the command, successful response, or event. Names are specified in description of each command, successful response, and event.

The `name` is a required attribute.

The `name` attribute is not used with faults.

code attribute

The value of the code attribute specifies the name of the fault.

The code is a required attribute for faults.

category attribute

The `category` attribute is a required attribute for faults, a `<CmdFault>` element, and is not used in any other elements.

The value of the category attribute can have one of two values:

`"client"`, which specifies that the client application is responsible for the fault.

`"server"`, which specifies that the API Server is responsible for the fault.

clientID attribute

The `clientID` attribute is a required attribute for events, an `<Event>` element. The `clientID` value identifies the event sent to the client application. The `eventID` value starts at 0 and increments by one for each event sent, so that the client application can test for missed events. The `eventID` attribute is unique for the `sessID` value.

<Param> Element

Each top-level message element may have zero, one or more <Param> elements, which contain the data being exchanged. The required <Param> elements are specified in the description of each command, successful response, or event.

Each <Param> element contains:

The attribute name, which specifies the name of the parameter. For faults, the value of the name attribute must be:

- "ErrString", if the child <Value> element contains the error string. This parameter is always used for faults.
- "Details", if the child <Value> element contains a detailed description of the error.

One of the following elements:

- <Value>
- <Array>
- <Struct>

<Value> Element

The <Value> element contains the value for its parent <Param> or <Array> element.

The <Value> element cannot contain other elements.

No attributes are used in the <Value> element.

<Array> Element

The <Array> element is used to specify multiple instances of the same type of data. The <Array> element can therefore contain multiple <Value> or <Struct> elements. The <Array> element must be enclosed within a <Param> element.

No attributes are used in the <Array> element.

The required <Array> elements are specified in the description of each command, successful response, or event.

<Struct> Element

The <Struct> element is used to group together related information. The <Struct> element can therefore contain multiple <Param> elements. The <Struct> element must be contained within a <Param> element or an <Array> element.

No attributes are used in the <Struct> element.

The required <Struct> elements are specified in the description of each command, successful response, or event.

XML Command Examples

This topic contains the following sections:

Overview

This topic contains XML examples of the E-Mail Manager API commands.

Notes:

Text in italics is meant to represent variables used in a hypothetical program.

Variables used in array elements are represented as locations in a single array variable; for example: queueID[0], queueID[1], queueID[2].

The XML examples are formatted to clearly show the different elements in the string.

These conventions are used to provide clear examples and are not meant to imply any programming standards required by the API.

Agent Command Examples

agentConnect

```
<APIEnvelope sessID = "new">
  <Cmd version = "1.0" name="cem.api.agentConnect">
    <Param name="userName">
       <Value>user_name</Value>
    </Param>
    <Param name="userPassword">
      <Value>user_password</Value>
    </Param>
 </Cmd>
</APIEnvelope>
```

agentDisconnect

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentDisconnect">
    <Param name="reasonCode">
       <Value>reason_code_value</Value>
    </Param>
 </Cmd>
</APIEnvelope>
```

agentDisconnectDuplicateAndConnect

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0"
name="cem.api.agentDisconnectDuplicateAndConnect">
    <Param name="userName">
       <Value>user_name</Value>
    </Param>
    <Param name="userPassword">
      <Value>user_password</Value>
    </Param>
 </Cmd>
</APIEnvelope>
```

agentGetLogoutReasons

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentGetLogoutReasons">
  </Cmd>
</APIEnvelope>
```

agentGetNotReadyReasons

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentGetNotReadyReasons">
  </Cmd>
</APIEnvelope>
```

agentGetProperties

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentGetProperties">
  </Cmd>
</APIEnvelope>
```

agentGetState

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentGetState">
  </Cmd>
</APIEnvelope>
```

agentMakeNotReady

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentMakeNotReady">
    <Param name="subreason">
      <Value>not_ready_reason_code</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

agentMakeNotRoutable

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentMakeNotRoutable">
  </Cmd>
</APIEnvelope>
```

agentMakeReady

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentMakeReady">
  </Cmd>
</APIEnvelope>
```

agentMakeRoutable

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.agentMakeRoutable">
  </Cmd>
</APIEnvelope>
```

Attachment Command Examples

attachGetContent

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.attachGetContent">
    <Param name = "attachID">
      <Value>attachment_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

attachGetList

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.attachGetList">
  </Cmd>
</APIEnvelope>
```

attachRegister

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.attachRegister">
      <Param name="originalFileName">
        <Value>originalFileName </Value>
      </Param>
      <Param name="attDesc">
        <Value>attDesc</Value>
      </Param>
     <Param name="mimeType">
       <Value>mimeType</Value>
     </Param>
     <Param name="attachment">
       <Value>attachment</Value>
     </Param>
   </Cmd>
</APIEnvelope>
```

attachUnregister

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.attachUnregister">
    <Param name="attachmentId">
      <Value>"attachmentId"</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

Category Command Examples

catsChangeForMsg

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.catsChangeForMsg">
    <Param name = "queueId">
      <Value>queue_id</Value>
    </Param>
    <Param name = "messageKey">
      <Value>message_id</Value>
    </Param>
    <Param name = "catsOriginal">
      <Array>
        <Value>orig_cat_1</Value>
        <Value>orig_cat_2</Value>
      </Array>
    </Param>
```

```
    <Param name = "catsSelected">
      <Array>
        <Value>new_cat_1</Value>
        <Value>new_cat_2</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

catsGetList

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.catsGetList">
  </Cmd>
</APIEnvelope>
```

catsGetListForMsg

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.catsGetListForMsg">
    <Param name = "messageKey">
      <Value>message_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

Event Command Examples

eventRegister

Following is an example of registering for events for two specific queues:

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.eventRegister">
    <Param name = "eventType">
      <Value>event_type</Value>
    </Param>
    <Param name = "queueScope">
      <Array>
        <Value>queue_guid[0]</Value>
        <Value>queue_guid[1]</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

Following is an example of registering for events for all queues:

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.eventRegister">
    <Param name = "eventType">
      <Value>event_type</Value>
    </Param>
    <Param name = "queueScope">
      <Value>all</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

Following is an example of registering for events for two specific messages:

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.eventRegister">
    <Param name = "eventType">
      <Value>event_type</Value>
    </Param>
    <Param name = "msgScope">
      <Array>
        <Value>msg_guid[0]</Value>
        <Value>msg_guid[1]</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

Following is an example of registering for events for all messages:

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.eventRegister">
    <Param name = "eventType">
      <Value>event_type</Value>
    </Param>
    <Param name = "msgScope">
      <Value>all</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

eventUnregister

```
<APIEnvelope sessId="session_id">
  <Cmd version="1.0" name="cem.api.eventUnregister" >
    <Param name="eventIds">
      <Array>
        <Value>event_id[0]</Value>
        <Value>event_id[1]</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

Message Command Examples

msgClaim

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgClaim">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="queueId">
      <Value>queue_id</Value>
    </Param>
    <Param name="unlockMessage">
      <Value>1</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgCreateNewStub

```
<APIEnvelope sessId="session_id">
  <Cmd version="1.0" name="cem.api.msgCreateNewStub">
    <Param name="subject">
      <Value>subject</Value>
    </Param>
    <Param name="text">
      <Value>message_content</Value>
    </Param>
    <Param name="extendedAttributes">
      <Array>
        <Struct>
          <Param name="attributeName">
            <Value>att_name</Value>
          </Param>
```

```
          <Param name="attributeValue">
            <Value>att_value</Value>
          </Param>
        </Struct>
        <Struct>
           <Param name="attributeName">
            <Value>att_name</Value>
          </Param>
          <Param name="attributeValue">
            <Value>att_value</Value>
          </Param>
        </Struct>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgDelete

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgDelete">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="queueId">
      <Value>queue_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgEscalate

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgEscalate">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="newOwner">
      <Value>new_queue_id</Value>
    </Param>
    <Param name="owner">
      <Value>old_queue_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgExit

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgExit">
  </Cmd>
</APIEnvelope>
```

msgGetContent

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgGetContent">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgGetExtendedAttributes

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgGetExtendedAttributes">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgGetLatestResponseDraft

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgGetLatestResponseDraft">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgGetResponses

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgGetResponses">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgGetStatus

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgGetStatus">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgMarkForArchiving

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgMarkForArchiving">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="queueId">
      <Value>queue_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgOpenByQueueAndKey

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgOpenByQueueAndKey">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="queueId">
      <Value>queue_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgOpenFromQueues

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgOpenFromQueues">
    <Param name="queueList">
      <Array>
        <Value>queue_id[0]</Value>
        <Value>queue_id[1]</Value>
        <Value>queue_id[2]</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgOpenForRead

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgOpenForRead">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgOpenForResponse

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgOpenForResponse">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

msgOpenRouted

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgOpenRouted">
  </Cmd>
</APIEnvelope>
```

MsgReassign

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgReassign">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="newOwner">
      <Value>new_queue_id</Value>
    </Param>
    <Param name="owner">
      <Value>old_queue_id</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

MsgSaveAsDraft

```
<APIEnvelope sessId="session_id">
  <Cmd version="1.0" name="cem.api.msgSaveAsDraft">
    <Param name="mailResponse">
      <Struct>
        <Param name="from">
          <Value>from_e-mail_address</Value>
        </Param>
        <Param name="trackingNumber">
          <Value>tracking_number</Value>
        </Param>
        <Param name="subject">
          <Value>subject_of_message</Value>
        </Param>
        <Param name="reply">
          <Value>text_of_message</Value>
        </Param>
        <Param name="to">
          <Array>
            <Value>to_address[0]</Value>
            <Value>to_address[1]</Value>
          </Array>
        </Param>
        <Param name="cc">
          <Array>
            <Value>cc_address[0]</Value>
            <Value>cc_address[1]</Value>
          </Array>
        </Param>
        <Param name="bcc">
          <Array>
            <Value>bcc_address[0]</Value>
```

```
            <Value>bcc_address[1]</Value>
          </Array>
        </Param>
        <Param name="attachments">
          <Array>
            <Value>attachment_id[0]</Value>
            <Value>attachment_id[1]</Value>
          </Array>
        </Param>
      </Struct>
   </Param>
  </Cmd>
</APIEnvelope>
```

msgSendKeepCurrent

```
<APIEnvelope sessId="session_id">
  <Cmd version="1.0" name="cem.api.msgSendKeepCurrent">
    <Param name="mailResponse">
      <Struct>
    <Param name="from">
      <Value>fromAddress</Value>
    </Param>
    <Param name="trackingNumber">
      <Value>trackingNumber</Value>
    </Param>
    <Param name="subject">
      <Value>agentName-sent from CATH via CEM API</Value>
    </Param>
    <Param name="reply">
      <Value>reply</Value>
    </Param>
    <Param name="to">
      <Array>
       <Value>yourAddress</Value>
      </Array>
    </Param>
    <Param name="xheader">
    <Array>
     <Struct>
     <Param name="xheaderName">
     <Value>xheaderName1</Value>
    </Param>
    <Param name="xheaderValue">
     <Value>xheaderValue1</Value>
    </Param>
    </Struct>
    <Struct>
     <Param name="xheaderName">
      <Value>xheaderName2</Value>
     </Param>
     <Param name="xheaderValue">
      <Value>xheaderValue2</Value>
     </Param>
```

```
    </Struct>
   </Array>
   </Param>
  </Struct>
  </Param>
 </Cmd>
</APIEnvelope>
```

msgSendMarkForArchiving

```
<APIEnvelope sessId="session_id">
 <Cmd version="1.0" name="cem.api.msgSendMarkForArchiving\">
<Param name="mailResponse">
      <Struct>
           <Param name="from">
              <Value>fromAddress</Value>
           </Param>
           <Param name="trackingNumber\">
              <Value>trackingNumber</Value>
           </Param>
           <Param name="subject">
              <Value>agentName-sent from CATH via CEM API</Value>
           </Param>
           <Param name="reply">
              <Value>reply</Value>
           </Param>
           <Param name="to">
             <Array>
               <Value>yourAddress</Value>
             </Array>
           </Param>
           <Param name="xheader">
           <Array>
            <Struct>
             <Param name="xheaderName">
              <Value>xheaderName1</Value>
             </Param>
             <Param name="xheaderValue">
              <Value>xheaderValue1</Value>
             </Param>
            </Struct>
            <Struct>
             <Param name="xheaderName">
              <Value>xheaderName2</Value>
             </Param>
             <Param name="xheaderValue">
              <Value>xheaderValue2</Value>
             </Param>
            </Struct>
           </Array>
           </Param>
          </Struct>
          </Param>
         </Cmd>
        </APIEnvelope>
```

msgUnarchive

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.msgUnarchive">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

Queue Command Examples

queueGetAgentList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetAgentList">
  </Cmd>
</APIEnvelope>
```

queueGetMemberSkillgroupList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetMemberSkillgroupList">
  </Cmd>
</APIEnvelope>
```

queueGetMsgInfo

```
<APIEnvelope sessID = "session_id">
  <Cmd version = "1.0" name="cem.api.queueGetMsgInfo">
    <Param name = "queueId">
       <Value>queue_GUID</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

queueGetPeerList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetPeerList">
  </Cmd>
</APIEnvelope>
```

queueGetReassignableList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetReassignableList">
  </Cmd>
</APIEnvelope>
```

queueGetSkillgroupList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetSkillgroupList">
  </Cmd>
</APIEnvelope>
```

queueGetStatistics

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.queueGetStatistics">
    <Param name="type">
      <Value>0</Value>
    </Param>
    <Param name="queueList">
      <Array>
        <Value>queue_ID[0]</Value>
        <Value>queue_ID[1]</Value>
        <Value>queue_ID[2]</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

Message Notes and History Command Examples

actionHistoryGetForMessage

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.actionHistoryGetForMessage">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="isOld">
      <Value>1</Value>
    </Param>
    <Param name="stringFormat">
      <Value>2</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

historyCheckOldForSender

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.historyCheckOldForSender">
    <Param name="sender">
      <Value>sender_email_address</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

historyCheckOldForTrackingNumber

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.historyCheckOldForTrackingNumber">
    <Param name="trackingNumber">
      <Value>tracking_number</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

historyGetForSender

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.historyGetForSender">
    <Param name="sender">
      <Value>sender_email_address</Value>
    </Param>
    <Param name="isOld">
      <Value>1</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

historyGetForTrackingNumber

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.historyGetForTrackingNumber">
    <Param name="trackingNumber">
      <Value>tracking_number</Value>
    </Param>
    <Param name="isOld">
      <Value>1</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesAddForMessage

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesAddForMessage">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="noteText">
      <Value>new_note</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesAddForSender

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesAddForSender">
    <Param name="sender">
      <Value>sender_email_address</Value>
    </Param>
    <Param name="noteText">
      <Value>new_note</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesAddForTrackingNumber

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesAddForTrackingNumber">
    <Param name="trackingNumber">
      <Value>tracking_number</Value>
    </Param>
    <Param name="noteText">
      <Value>new_note</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesGetForMessage

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesGetForMessage">
    <Param name="messageKey">
      <Value>message_key</Value>
    </Param>
    <Param name="isOld">
      <Value>1</Value>
    </Param>
    <Param name="stringFormat">
      <Value>2</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesGetForSender

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesGetForSender">
    <Param name="sender">
      <Value>sender_email_address</Value>
    </Param>
    <Param name="stringFormat">
      <Value>2</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

notesGetForTrackingNumber

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.notesGetForTrackingNumber">
    <Param name="trackingNumber">
      <Value>tracking_number</Value>
    </Param>
    <Param name="stringFormat">
      <Value>2</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

Template Command Examples

templCreate

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templCreate">
    <Param name="templateAttributes">
      <Struct>
        <Param name="name">
          <Value>template_name</Value>
        </Param>
        <Param name="description">
          <Value>template_desc</Value>
        </Param>
        <Param name="editText"
          <Value>template_text</Value>
        </Param>
        <Param name="library">
          <Value>template_library</Value>
        </Param>
        <Param name="owner">
```

```
          <Value>agent_GUID</Value>
        </Param>
        <Param name="keywords">
          <Array>
            <Value>template_keyword[0]</Value>
            <Value>template_keyword[1]</Value>
            <Value>template_keyword[2]</Value>
          </Array>
        </Param>
        <Param name="attachments">
          <Array>
            <Value>attachment_id[0]</Value>
            <Value>attachment_id[1]</Value>
            <Value>attachment_id[2]</Value>
          </Array>
        </Param>
      </Struct>
    </Param>
  </Cmd>
</APIEnvelope>
```

templDelete

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templDelete">
    <Param name="templateAttributes">
      <Struct>
     <Param name="templateKey">
       <Value>template_key</Value>
     </Param>
     <Param name="attachments">
       <Array>
         <Value>attachment_id[0]</Value>
         <Value>attachment_id[1]</Value>
         <Value>attachment_id[2]</Value>
       </Array>
     </Param>
     <Param name="library">
       <Value>template_library</Value>
     </Param>
     <Param name="owner">
       <Value>agent_GUID</Value>
     </Param>
      </Struct>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetAllKeywords

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetAllKeywords">
  </Cmd>
</APIEnvelope>
```

templGetAttachments

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetAttachments">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetDefaultDynamicText

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetDefaultDynamicText">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetDynamicText

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetDynamicText">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
    <Param name="to">
      <Value>to_header</Value>
    </Param>
    <Param name="from">
      <Value>from_header</Value>
    </Param>
    <Param name="replyTo">
      <Value>replyTo_header</Value>
    </Param>
    <Param name="subject">
      <Value>subject_header</Value>
    </Param>
    <Param name="date">
      <Value>date_header</Value>
    </Param>
    <Param name="trackingNumber">
      <Value>tracking_number</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetDynamicTokenInfo

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetDynamicTokenInfo">
  </Cmd>
</APIEnvelope>
```

templGetEditFields

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetEditFields">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetKeywords

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetKeywords">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetList

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetList">
    <Param name="type">
      <Value>LIBRARY</Value>
    </Param>
    <Param name="libraryName">
      <Value>SUPPORT</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templGetPublicLibraries

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetPublicLibraries">
  </Cmd>
</APIEnvelope>
```

templGetText

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templGetText">
    <Param name="templateKey">
      <Value>template_key</Value>
    </Param>
  </Cmd>
</APIEnvelope>
```

templReplace

```
<APIEnvelope sessID = "session_id">
  <Cmd version="1.0" name="cem.api.templReplace">
    <Param name="templateAttributes">
      <Struct>
        <Param name="templateKey">
          <Value>template_key</Value>
        </Param>
        <Param name="name">
          <Value>template_name</Value>
        </Param>
        <Param name="description">
          <Value>template_desc</Value>
        </Param>
        <Param name="editText"
          <Value>template_text</Value>
        </Param>
        <Param name="library">
          <Value>template_library</Value>
        </Param>
        <Param name="owner">
          <Value>agent_GUID</Value>
        </Param>
        <Param name="keywords">
          <Array>
            <Value>template_keyword[0]</Value>
            <Value>template_keyword[1]</Value>
            <Value>template_keyword[2]</Value>
          </Array>
        </Param>
        <Param name="attachments">
          <Array>
            <Value>attachment_id[0]</Value>
            <Value>attachment_id[1]</Value>
            <Value>attachment_id[2]</Value>
          </Array>
        </Param>
      </Struct>
    </Param>
  </Cmd>
</APIEnvelope>
```

XML Event Examples

This topic contains the following sections:

Overview

This topic contains XML examples of the E-Mail Manager API events.

msgOverdue Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-1">
  <Event version="1.0" name="msgOverdue" clientId="1">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="messageKey">
      <Value>000000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
      <Value>1</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_reassigned Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-1">
  <Event version="1.0" name="messageOnQueue_reassigned" clientId="3">
    <Param name="automaticReassign">
      <Value>1</Value>
    </Param>
    <Param name="doneBy">
      <Value>1</Value>
    </Param>
    <Param name="eventType">
      <Value>2</Value>
    </Param>
    <Param name="action">
      <Value>3</Value>
    </Param>
    <Param name="sourceQueueType">
      <Value>1</Value>
    </Param>
    <Param name="sourceQueueId">
      <Value>dd8203216f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param
name="messageKey"><Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
    <Param name="actor">
      <Value>00000000000000000000000000000000</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_received Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-1">
  <Event version="1.0" name="messageOnQueue_received" clientId="4">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="messageKey">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
      <Value>2</Value>
    </Param>
    <Param name="action">
```

```
      <Value>1</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_unarchived Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-1">
  <Event version="1.0" name="messageOnQueue_unarchived" clientId="5">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="sendStatus">
      <Value>1</Value>
    </Param>
    <Param name="messageKey">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
      <Value>2</Value>
    </Param>
    <Param name="actor">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="action">
      <Value>2</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

queueOverload Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-1">
  <Event version="1.0" name="queueOverload" clientId="2">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="eventType">
      <Value>3</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

agentNotRoutable Event Example

```
<APIEnvelope lastAckedEventId="-1" sessId="5hbmzsnnr9" type="event">
  <Event clientId="2" name="agentNotRoutable" version="1.0">
    <Param name="eventType">
      <Value>4</Value>
    </Param>
    <Param name="agentId">
      <Value>ecf21a1062de11d59e01000000000000</Value>
    </Param>
  </Event>
</APIEnvelope>
```

Commands, Responses, and Faults

Overview of API Commands and Responses

This topic contains the following sections:

Before reading this topic and proceeding with commands, you should read the following topics:

API DTD

The <APIEnvelope> Element

Contents of the <APIEnvelope>

Overview of Connections, and HTTP or Socket connections

Designing Command Workflows

Commands as XML

This section contains the following information:

Overview

You send commands to the API Server as XML. The general XML command structure adheres to the following format:

```
<APIEnvelope sessID = "5hbmzsnnr9">
  <Cmd version="1.0" name="cem.api.command_name">
    //One or more Param elements.
    //Param elements contain Value, Array, and Struct elements
  </Cmd>
</APIEnvelope>
```

Contents of Commands

Commands are wrapped in the APIEnvelope element, which contains an attribute specifying the session for which the command is being sent. The value for this attribute, sessID, must be maintained for all commands sent for the agent.

A command contains a <Cmd> element, with the following attributes:

version, which always equals "1.0".

name, which equals the name of the command.

Input parameters, depending on the command's requirements.

Example of a Simple Command

The following example, for the agentMakeNotReady command, shows the use of two parameters, with no Struct or Array elements.

```
<APIEnvelope sessID = "5hbmzsnnr9">
  <Cmd version = "1.0" name="cem.api.agentMakeNotReady">
    <Param name = "notReadyReason">
      <Value>2</Value>
    </Param>
 </Cmd>
</APIEnvelope>
```

Example of a Command with an Array Element

The following example, for the msgOpenFromQueues command, shows the use of a top-level Param element, containing an Array element, which contains multiple Value elements.

```
<APIEnvelope sessID = "5hbmzsnnr9">
  <Cmd version="1.0" name="cem.api.msgOpenFromQueues">
    <Param name="queueList">
      <Array>
        <Value>4ybmzsnst</Value>
        <Value>8tnhzsnlk7</Value>
      </Array>
    </Param>
  </Cmd>
</APIEnvelope>
```

Example of a Command with Array and Struct Elements

The following example, for the queueGetInfo command, shows the use of a top-level Param element, containing an Array element, which contains multiple Struct elements.

```
<APIEnvelope sessID = "5hbmzsnnr9">
  <Cmd version="1.0" name="cem.api.queueGetInfo">
    <Param name="QueueList">
      <Array>
        <Struct>
          <Param name="queue">
            <Value>4ybmzsnst8</Value>
          </Param>
        </Struct>
        <Struct>
          <Param name="queue">
            <Value>8tnhzsnlk7</Value>
          </Param>
        </Struct>
      </Array>
  </Cmd>
</APIEnvelope>
```

Sending Command Input

To send command input, you must have an active connection for an agent. You send the input to the API server over a socket connection or an HTTP connection. When you send command input, either a response or a fault is returned to your program.

State and Mode Issues

Certain commands require the agent, message, or both to be in a particular state, and the agent to be in a certain mode. Certain commands also modify the agent or message state, or the agent mode. This information is described in the topic for each command, as well as in the topics on agent state, message state, and agent mode.

Responses as XML

This section contains the following information:

Overview

When a command succeeds, data for the successful response is returned as XML in the following format:

```
<APIEnvelope type="response" sessID="session_ID">
  <CmdResponse version = "1.0" name="command_name">
    //One or more <Param> elements,
    //which can contain <Array> and <Struct>
    //elements, which can contain multiple
    //other <Param> elements.
  </CmdResponse>
</APIEnvelope>
```

Contents of Responses

Responses are wrapped in the `APIEnvelope` element, which contains session identifier as the value of the `sessID` attribute.

The response contains:

A `<CmdResponse>` element, with the following attributes:

- `version`, which in this release always equals `"1.0"`.
- `name`, which equals the name of the command.

The `<Param>` named `returnCode`, which contains a `<Value>` element indicating whether the command succeeded, or if there were problems.

Possibly one or more other <Param> elements, which may contain <Value> elements with data, or <Array> or <Struct> elements, which can contain other <Param> or <Value> elements.

The description in this guide of each command specifies the response data.

When a command fails, a fault is returned instead of a response.

Example of a Simple Response

The simplest response a command may receive would contain only a return code, as a single parameter and value. The following example shows this, as a response to the msgDelete command.

```
<APIEnvelope type="response" sessID="5hbmzsnnr9">
  <CmdResponse version="1.0" name="cem.api.msgDelete">
    <Param name="returnCode">
      <Value>0</Value>
    </Param>
  </CmdResponse>
</APIEnvelope>
```

Example of a Response with Values in a Struct

A response may contain parameters and values within a Struct element. The following example shows this, as a response to the agentGetState command.

```
<APIEnvelope type="response" sessID="5hbmzsnnr9">
  <CmdResponse version="1.0" name="cem.api.agentGetState">
    <Param name="returnCode">
      <Value>0</Value>
    </Param>
    <Param name="agentState">
      <Struct>
        <Param name="name">
          <Value>working</Value>
        </Param>
        <Param name="code">
          <Value>1</Value>
        </Param>
      </Struct>
    </Param>
    <Param name="agentMode">
      <Struct>
        <Param name="name">
          <Value>nonpush</Value>
        </Param>
        <Param name="code">
          <Value>1</Value>
```

```
        </Param>
      </Struct>
    </Param>
  </CmdResponse>
</APIEnvelope>
```

Example of a Response with Values in an Array

A response may contain a parameter that contains multiple values in an Array element, with no Struct element. The following example shows this, as a response to the `msgGetLatestResponseDraft` command.

```
<APIEnvelope type="response" sessID="5hbmzsnnr9">
  <CmdResponse version="1.0"
name="cem.api.msgGetLatestResponseDraft">
    <Param name="returnCode">
      <Value>0</Value>
    </Param>
    <Param name="to">
      <Array>
        <Value>customer@aol.com</Value>
        <Value>support@company.com</Value>
      </Array>
    </Param>
    <Param name="cc">
      <Array>
        <Value>anothercustomer@aol.com</Value>
        <Value>managers@company.com</Value>
      </Array>
    </Param>
    <Param name="bcc">
      <Array>
        <Value>supervisorlist@aol.com</Value>
        <Value>jeff@company.com</Value>
      </Array>
    </Param>
    <Param name="subject">
      <Value>Re: product inquiry</Value>
    </Param>
    <Param name="text">
      <Value>Your inquiry is currently being investigate,
sincerely</Value>
    </Param>
  </CmdResponse>
</APIEnvelope>
```

Example of a Response with an Array of Structs

A response may contain an Array, which contains one or more Structs, each of which contains the same parameters. The following example shows this, as a response to the `msgGetExtendedAttributes` command.

```
<APIEnvelope type="response" sessID="5hbmzsnnr9">
  <CmdResponse version="1.0" name="cem.api.msgGetExtendedAttributes">
    <Param name="returnCode">
      <Value>0</Value>
    </Param>
    <Param name="extendedAttributes">
      <Array>
        <Struct>
          <Param name="name">
            <Value>keyword</Value>
          </Param>
          <Param name="value">
            <Value>Information Request</Value>
          </Param>
        </Struct>
        <Struct>
          <Param name="name">
            <Value>kekyword</Value>
          </Param>
          <Param name="value">
            <Value>Product Complaint</Value>
          </Param>
        </Struct>
      </Array>
    </Param>
  </CmdResponse>
</APIEnvelope>
```

Command Definition File

API commands are defined in the apiserver-commands.xml that is installed with the UI/API Server in the following location:

installation-folder\uiroot\WEB-INF\properties\default\cem

In this file, API commands are defined in the CMDNAMESPACE element named "api". Each command definition is enclosed in the CMD element and includes:

The associated command handler on the UI Server, specified by the value of the hndlr attribute in the CMD element.

Caution: You should not change the value of the hndlr attribute; doing so will cause the API command to fail.

A description.

Required input parameters, each specified with a PARAM element.

A definition of the response, specified with the RESPONSE element.

Note: The format of the documentation file may be changed in later releases.

Overview of Faults

This topic contains the following sections:

Before reading this topic and proceeding with faults, you should read the following topics:

API DTD

The <APIEnvelope> Element

Contents of the <APIEnvelope>

Overview of Connections, and HTTP or Socket connections

Receiving Faults

A command may receive a fault, instead of a response, when the command fails for one of many possible reasons.

In some situations, instead of a fault, the command may receive a response with a `returnCode` value indicating that the command was not completely successful.

Fault Properties

Fault properties are defined in the file `cem_faults.properties`, located in the `installation-folder\uiroot\WEB-INF\properties\common\cem` folder.

Contents of Faults

A fault contains:

A <CmdFault> element, with the following attributes:

- version, which always equals "1.0".
- code, which equals the name of the fault.
- category, which has the value "Client" or "Server". "Client" faults are faults related to problems with how the application sent in a command; for example, a command with incorrect input parameters would cause a "Client" fault. "Server" faults are faults generated because of an internal API Server error.

The parameter named ErrString, which contains a value with a description of the event. This value string can be internationalized.

Optionally, a parameter named Details. To have this parameter included in faults, set the stackTraceOnFault property in the connection property file to true.

Faults as XML

When a command fails, data for the failure is returned as XML in the following format:

```
<APIEnvelope  type="response" sessId="session_ID">
  <CmdFault version="1.0" code="fault.fault_name" category="Client">
    <Param name="ErrString">
      <Value>Description of Error</Value>
    </Param>
    <Param name="Details">
      <Value>Details of the error</Value>
    </Param>
  </CmdFault>
</APIEnvelope>
```

Fault Definitions

The following tables lists and describes the faults that can be returned by commands:

Fault Name	Error String
unknownError	An expected error occurred. The error message is: {0}
invalidSession	This browser does not have an active session to CEM because you never logged on, the session was logged out by another user, or the session was logged out because of inactivity. Session ID is {0}.
notAuthorized	The userId and/or password are not valid.
accountDisabled	The userID was disabled.
alreadyLoggedIn	The user, '{0}', is already logged into CEM.
sessionAlreadyLoggedIn	A user attempted to log in from a browser that already has a logged in session.
forcePasswordChange	You must change your password {0}.
confirmPasswordDidntMatch	The New Password and the Confirm New Passwords that you entered were not the same.
passwordDidntChange	The New Password that was entered is the same as the Old Password.
passwordTooShort	The new password is too short.
stateChangeRejected	TServer rejected the agent state change.
exceedLicenseLimit	Login failed: Number of valid logins has exceeded the limit for the current license. Please contact your administrator.
illegalStateChange	The attempted agent state change is illegal.
illegalModeChange	The attempted agent mode change

	is illegal.
networkConnectivityException	The agent cannot log in because the connection between E-Mail Manager and ICM software is not working.
unexpectedTServerError	Unexpected TServer Error occurred. Handler ID is {0}. Result code is {1}.
invalidDBHostname	Missing or invalid DB_HOSTNAME property in TServer properties file.
invalidDBPortnum	Missing or invalid DB_PORTNUM property in TServer properties file.
invalidVersion	Invalid or missing DB_VERSION parameter in TServer properties file.
invalidDBLogin	Invalid or missing DB_LOGIN parameter in TServer properties file.
invalidDBPass	Invalid or missing DB_PASS parameter in TServer properties file.
invalidNumConnections	Invalid or missing DB_NUMCONNECTIONS parameter in TServer properties file.
dataReadException	Server data read error: {0}
wlPasswordException	WLPasswordException occurred. Error string is: {0}
headerParseException	HeaderParseException occurred. Error string is: {0}
socketException	SocketException occurred. Error string is: {0}
ioException	IOException occurred. Error string is: {0}
adapterNotInitialized	Attempt to do a TServer connection handler request when the connection adapter has not been initialized!
noTcaAvailable	There are currently no

	TServerConnectionAdapter''s available to run this request.
connectionFreeError	Attempt to free a connection that is not in the in Use List.
invalidUCR	Invalid User Connection Resource. Class name is {0}
stateChangeRejected	A state change request was rejected by TServer. Current state is: {0}. Requested new state is: {1}
illegalStateChange	An illegal state change request was attempted. Current state is: {0}. Requested new state is: {1}
modeChangeRejected	A mode change request was rejected by TServer. Current mode is: {0}. Requested new mode is: {1}
illegalModeChange	An illegal mode change request was attempted. Current mode is: {0}. Requested new mode is: {1}
eventRegisterTypeParameter	eventRegister eventType {0} is invalid.
eventRegisterNoScopeParameter	eventRegister queueScope or msgScope required.
eventRegisterScopeParameters	eventRegister queueScope and msgScope is invalid.
eventRegisterMismatchedParameters	eventRegister eventType and msgScope is invalid.
eventRegisterNotAllParameter	eventRegister scope value '{0}' is not 'all'.
eventRegisterScopeNotArray	eventRegister scope type {0} is not array.
eventRegisterAlreadyRegistered	eventRegister all events already registered.
eventRegisterDuplicateArrayElement	eventRegister duplicate array element '{0}'.

eventRegisterJmsError	eventRegister JMS exception {0}.
eventRegisterSubscriptionStateError	eventRegister Subscription State exception {0}.
eventRegisterDuplicateSubscription	eventRegister Duplicate subscription {0}.
eventRegisterExceptionError	eventRegister exception {0}.
eventUnregisterParameterType	eventUnregister eventIds type {0} is invalid.
eventUnregisterDuplicateArrayElement	eventUnregister duplicate array element {0}.
eventUnregisterArrayElementParse	eventUnregister array element not parsed {0}.
eventUnregisterNoRegistrations	eventUnregister no event registrations. No subscription.
eventUnregisterNoActiveIds	eventUnregister no event registrations. No active ids.
eventUnregisterNoMatch	eventUnregister no match exception {0}.
eventUnregisterJmsError	eventUnregister JMS exception {0}.
eventUnregisterExceptionError	eventUnregister exception {0}
eventUnregisterInvalidId	eventUnregister invalid id {0}.
notIPTA	Messages are not externally routed to this user.
notPush	The user is not in routable mode.
inPushMode	The user is in routable mode.
inWrongState	The user is in the wrong state for this command.
noMessageRouted	No message was routed to the user.
noMessageAvailable	There is no message available.

noMessageOpen	No message was open.
noDraft	No draft response exists for the message.
couldNotGetDraft	Error getting draft response for message.
noResponses	No responses exist for the message.
couldNotGetResponses	Getting response information for a message failed, error: {0}.
stateChangeFailed	A message state change failed, error: {0}.
chownFailed	A message ownership change failed, error: {0}.
messageDeleteFailed	Message delete failed, error: {0}.
unarchiveFailed	Unarchiving a message failed, error: {0}.
newBaseMessageFailed	Could not create new message stub, error: {0}.
storeMetadataFailed	Could not store extended attributes, error: {0}.
setoldFailed	Marking a message for archiving failed, error: {0}.
couldNotGetMStat	Getting status flags for message failed, error: {0}.
couldNotGetMessageContent	Getting message content failed, error: {0}.
couldNotGetMessageStatus	Getting dynamic message information failed, error: {0}.
processingFailure	Message processing failure, error: {0}.
noExtendedAttributes	No extended attributes were found for the message.

extendedAttrError	Could not get extended attributes, error: {0}.
messageNotOnAgentQueue	Message not found on agent queue.
messageNotOnQueue	Message not found on queue.
replyMailFailure	Replying to mail failed, a: {0}, b: {1}.
invalidMessageKey	The message key, {0}, is not the message key for a message that is currently stored in E-Mail Manager.
invalidTrackingNumber	The tracking number, {0}, is not a tracking number for a thread that is currently stored in E-Mail Manager.
invalidSender	The sender, {0}, is not a sender for a message that is currently stored in E-Mail Manager.
noTemplates	No templates were found.
noLibraries	No libraries were found.
noKeywords	No keywords were found.
notMsgArray	Parameter {0} is not an Array.
notMsgProps	A parameter is not a Struct.
missingProp	Required parameter {0} is missing or empty.
paramNotValue	Parameter {0} does not have a value.
valueNotNumeric	Parameter {0} is not numeric.
emptyMsgArray	An Array parameter has no elements.
paramLengthError	The length of parameter {0} is wrong.
missingQueueIds	A queueIds parameter is missing.

emptyArrayValue	An Array value is empty.
arrayValueLengthError	An Array value length is wrong.
arrayElementType	An Array element is not a string.
invalidValue	An invalid value {0} was specified for the parameter {1}.
noAttachments	No attachments were found.
invalidAgentGuid	An invalid agent Guid was specified for parameter {0}.
incomingEventMsgMgrInitError	IncomingEventMsgMgr initialization failed.
messageAdapterStartError	Message adapter start failed.
ARMlogindenied	Agent Controller denied login.
webviewNotAllowed	Agent not allowed to log in.
webviewNotExist	Agent does not exist.
webviewNotAuthenticated	Agent not authenticated.
noDuplicateLogin	Supplied information does not match logged on agent
duplicateUserNotFound	Duplicate agent not logged on.
incorrectDataType	An incorrect data type was encountered while processing returned information.
incorrectCatsFormat	Malformed categories data for parameter {0}.
noSuchField	No such field exception: {0}.

Designing Command Workflows

This topic contains the following sections:

Overview

When using the E-Mail Manager Agent Desktop, agents typically retrieve and respond to messages using one of three methods:

Pick Mode

Pull Mode

Push Mode (including messages assigned through ICM software)

The workflow also depends on whether the agent must use the Message or Wrap states, as determined by role settings.

The E-Mail Manager API provides commands that enable you to duplicate these workflows, as described in the rest of this topic.

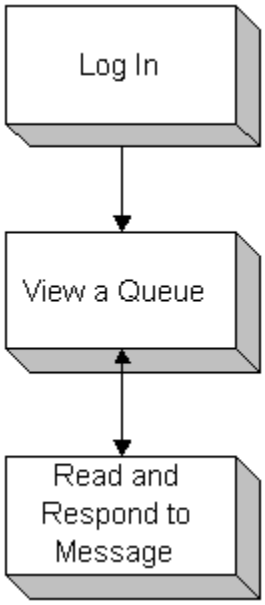
For more information, see the *Cisco E-Mail Manager Administration Guide* and the *Cisco E-Mail Manager Agent Guide.*

Disclaimer

This topic is intended to provide suggestions on how a third-party application can work. It provides simple examples of how you can design an application and are not intended to imply limitations in your design choices. The examples and suggestions may or may not apply to your specific needs.

Pick Mode

Pick mode refers to the workflow in which agents select messages from their personal queue and from skill group queues in which they are members.



Note: This image does not show the mandatory use of the Message screen or Wrap screen.

To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the `queueGetAgentList` or `queueGetMemberSkillgroupList` command.

3. Get information about messages in the queue using the `queueGetMsgInfo` command.

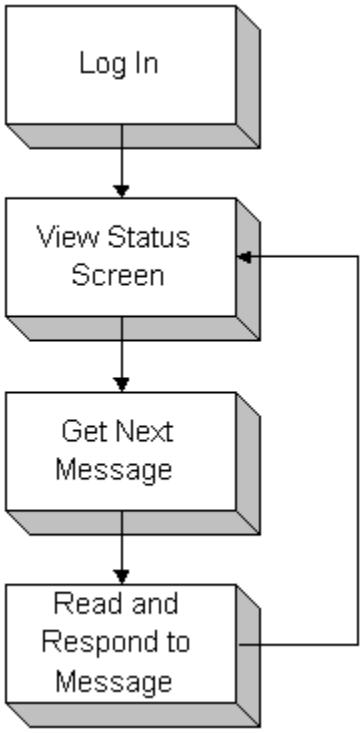
4. Open a specific message using the `msgOpenByQueueAndKey` command. The message is in the `agentInResponse` state.

5. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

6. Repeat steps 4 and 5 for other messages.

Pull Mode

Pull mode refers to the workflow in which agents retrieve messages from the Status screen by clicking **Get Next** for a single queue or for all queues in which the agent is a member. When the agent clicks **Get Next**, the message with the highest priority and longest wait time, from a single queue or all queues listed, is presented.



Note: This image does not show the mandatory use of the Message screen or Wrap screen.

To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pull from using the `queueGetAgentList` or `queueGetMemberSkillgroupList` command.

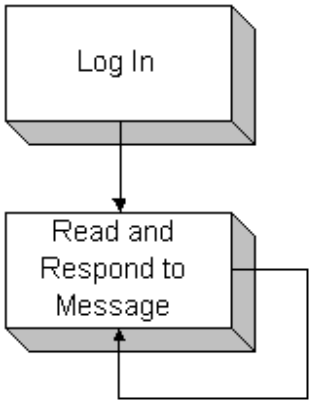
3. Open the next message using the `msgOpenFromQueues` command. The message is in the `agentInResponse` state.

4. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

5. Repeat steps 3 and 4 for other messages.

Push Mode

The Push mode refers to the workflow in which agents do not select messages from their queue. Messages from the agent's personal queue, as well as queues in which the agent is a member, are automatically presented for reading and responding to, as shown in the figure below.



Note: This workflow does not show the mandatory use of the Message screen or Wrap screen.

Messages with the highest priority and longest wait time are presented first. Push routing ensures that the most important messages are responded to first. The agent does not have a choice of messages or queues to work with.

Agents who belong to ICM Routing skill groups and are assigned messages through ICM software are presented these messages in push mode, regardless of their role settings.

To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command.

2. Use the `eventRegister` command to register for Type 2 events for the agent's queue.

3. Make the agent routable using the `agentMakeRoutable` command.

4. When the agent receives a Type 2 event for a message on the queue, use the `msgOpenRouted` command to open the message. The message is in the `agentInResponse` state.

5. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

6. Repeat steps 4 and 5 for other messages.

Note: Agents who use push mode are able to leave the push mode to pick messages from their personal queues by using the `agentMakeNotRoutable` command.

The Message Screen

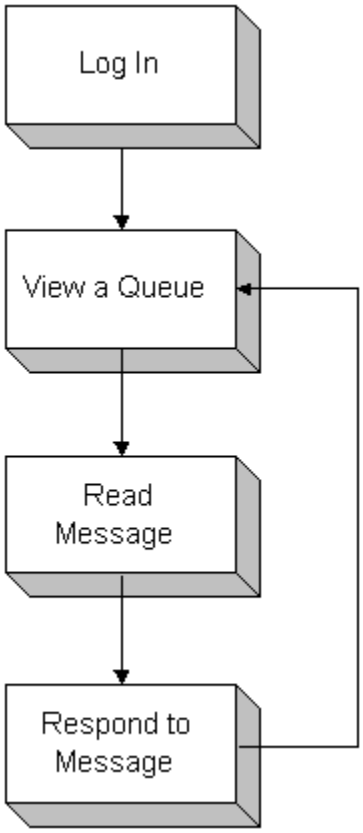
If an agent is based on a role that has the access privilege `For (role): display Message screen before allowing response` checked, the agent is required to view the Message screen before viewing the Response screen for each message. By dividing reading and responding into two steps, managers can track the time spend on both activities separately.

In a third-party application using the E-Mail Manager API, there may or may not be a Message Screen that is equivalent to the Message Screen in the Agent Desktop; however, agents are required to enter the `read` state.

Whether or not agents must use the Message screen is independent of whether they use pick, pull, or push mode, or whether they view the Wrap screen.

Pick Mode with the Message Screen

The following figure shows the workflow for agents who use pick mode and are required to view the Message screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the `queueGetAgentList` or `queueGetMemberSkillgroupList` command.

3. Get information about messages in the queue using the `queueGetMsgInfo` command.

4. Open a specific message using the `msgOpenByQueueAndKey` command. The message is in the `agentInRead` state.

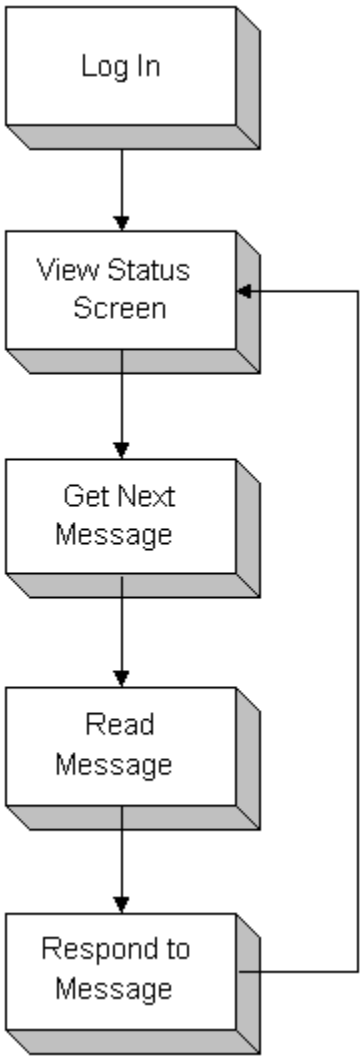
5. Open the message for response using the `msgOpenForResponse` command.

6. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

7. Repeat steps 4 through 6 for other messages.

Pull Mode With the Message Screen

The following figure shows the workflow for agents who use pull mode and are required to view the Message screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the `queueGetAgentList` or queueGetMemberSkillgroupList command.

3. Open the next message using the `msgOpenFromQueues` command. The message is in the `agentInRead` state.

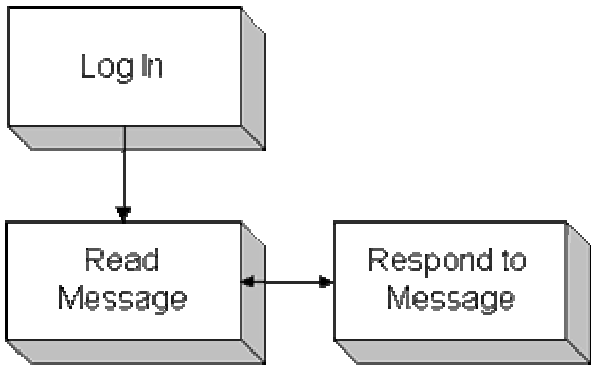
4. Open the message for response using the `msgOpenForResponse` command.

5. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

6. Repeat steps 3 through 5 for other messages.

Push mode With the Message Screen

The following figure shows the workflow for agents who use push mode and are required to view the Message screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command.

2. Use the `eventRegister` command to register for Type 2 events for the agent's queue.

3. Make the agent mode push using the `agentMakeRoutable` command.

4. When the agent receives a Type 2 event for a message on the queue, use the `msgOpenRouted` command to open the message. The message is in the `agentInRead` state.

5. Open the message for response using the `msgOpenForResponse` command.

6. Send a response using the `msgSendKeepCurrent` or `msgSendMarkForArchiving` command.

7. Repeat steps 4 through 6 or other messages.

The Wrap Screen

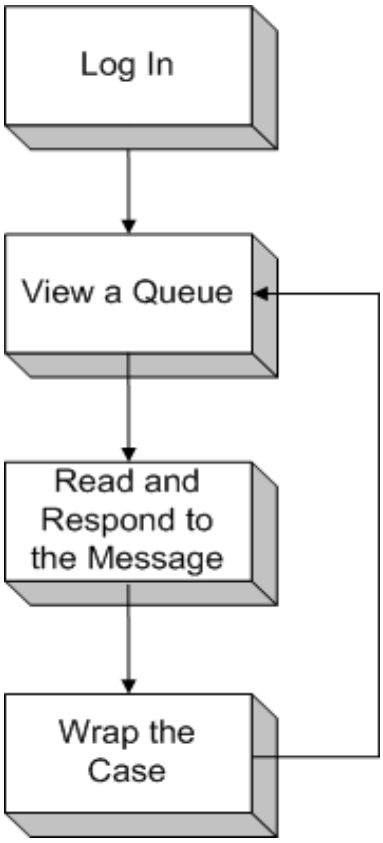
Depending on the setting of the access privilege `Allow (role) to use wrap screen`, agents may be required to view the Wrap screen after responding to a message. By forcing agents to view this screen while they wrap the case in an external system, managers can track the time spent wrapping cases separately.

In a third-party application using the E-Mail Manager API, there may or may not be a Wrap Screen that is equivalent to the Wrap Screen in the Agent Desktop; however, agents are required to enter the `wrap` state.

Whether or not agents must use the Wrap screen is independent of whether they use pick, pull, or push mode, or whether they view the Message screen.

Pick Mode with the Wrap Screen

The following figure shows the workflow for agents who use pick mode and are required to view the Wrap screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the queueGetAgentList or queueGetMemberSkillgroupList command.

3. Get information about messages in the queue using the queueGetMsgInfo command.

4. Open a specific message using the msgOpenByQueueAndKey command. The message is in the agentInResponse state.

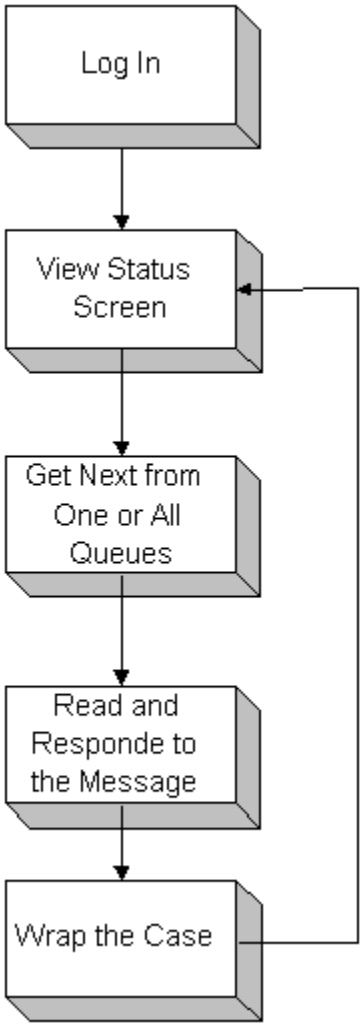
5. Send a response using the msgSendMarkForArchiving command. (The msgSendKeepCurrent command does not take you to wrap state.)

6. Wrap the case and use the msgExit command to close the message.

7. Repeat steps 4 through 6 for other messages.

Pull Mode with the Wrap Screen

The following figure shows the workflow for agents who use pull mode and are required to view the Wrap screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the `queueGetAgentList` or `queueGetMemberSkillgroupList` command.

3. Open the next message using the `msgOpenFromQueues` command. The message is in the `agentInResponse` state.

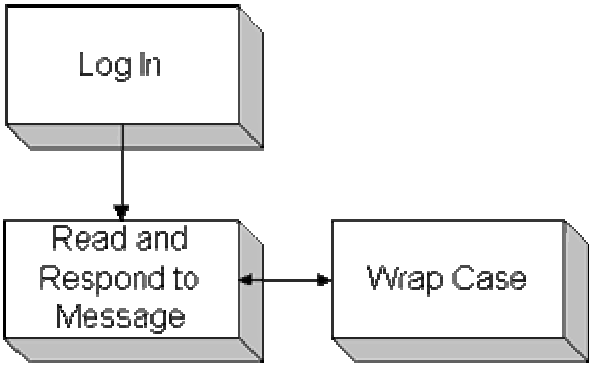
4. Send a response using the `msgSendMarkForArchiving` command. (The `msgSendKeepCurrent` command does not take you to wrap state.)

5. Wrap the case and use the `msgExit` command to close the message.

6. Repeat steps 3 through 5 for other messages.

Push Mode With the Wrap Screen

The following figure shows the workflow for agents who use push mode and are required to view the Wrap screen:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command.

2. Use the `eventRegister` command to register for Type 2 events for the agent's queue.

3. Make the agent routable using the `agentMakeRoutable` command.

4. When the agent receives a Type 2 event for a message on the queue, use the `msgOpenRouted` command to open the message. The message is in the `agentInResponse` state.

5. Send a response using the `msgSendMarkForArchiving` command. (The `msgSendKeepCurrent` command does not take you to wrap state.)

6. Wrap the case and use the `msgExit` command to close the message.

7. Repeat steps 4 through 6 for other messages.

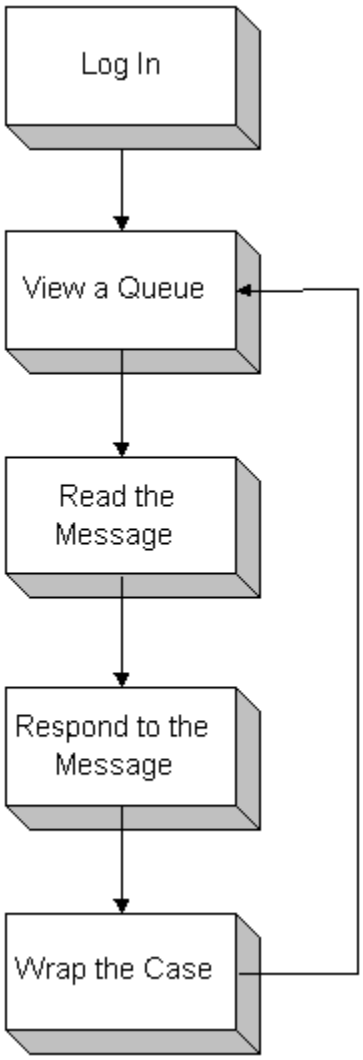
The Message and Wrap Screens Together

You can require agents to use both the Message and Wrap screens if you want to track the time spent on both these tasks. You must base such agents on a role that has both the For (role): display Message screen before allowing response and Allow (role) to use Wrap screen access privileges set.

Whether or not agents must use the both the Message screen and the Wrap screen is independent of whether they use pick, pull, or push mode.

Pick mode with the Message and Wrap Screens

The following figure shows the workflow for agents who use pick mode and are required to view the Message and Wrap screens:



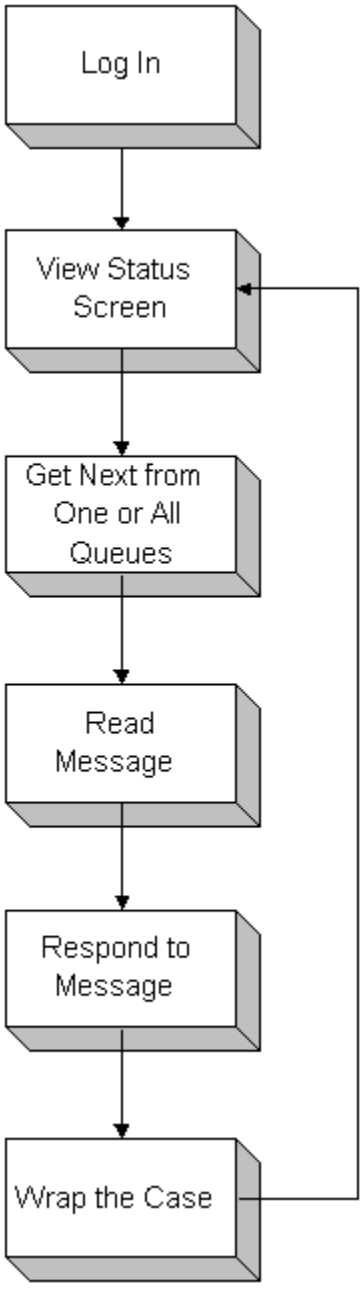
To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.
2. Get information about queues from which the agent can pick from using the `queueGetAgentList_` or queueGetMemberSkillgroupList command.
3. Get information about messages in the queue using the `queueGetMsgInfo` command.
4. Open a specific message using the `msgOpenByQueueAndKey` command. The message is in the `agentInRead` state.
5. Open the message for response using the `msgOpenForResponse` command.

6. Send a response using the `msgSendMarkForArchiving` command. (The `msgSendKeepCurrent` command does not take you to wrap state.)
7. Wrap the case and use the `msgExit` command to close the message.
8. Repeat steps 4 through 7 for other messages.

Pull mode with the Message and Wrap Screens

The following figure shows the workflow for agents who use pull mode and are required to view the Message and Wrap screens:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command and make the agent ready using the `agentMakeReady` command.

2. Get information about queues from which the agent can pick from using the `queueGetAgentList` or queueGetMemberSkillgroupList command.

3. Open the next message using the `msgOpenFromQueues` command. The message is in the `agentInRead` state.

4. Open the message for response using the `msgOpenForResponse` command.

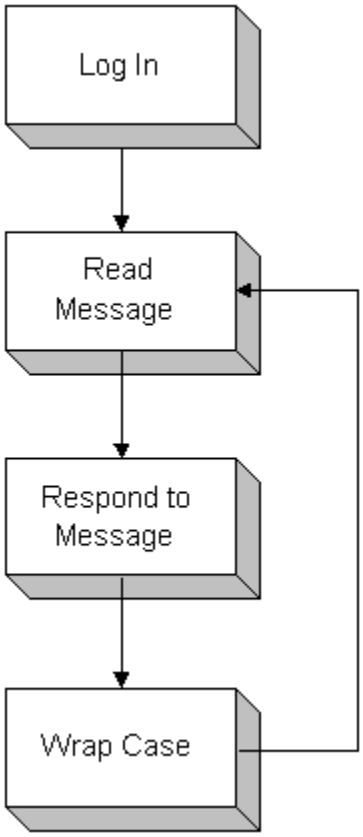
5. Send a response using the `msgSendMarkForArchiving` command. (The `msgSendKeepCurrent` command does not take you to wrap state.)

6. Wrap the case and use the `msgExit` command to close the message.

7. Repeat steps 3 through 6 for other messages.

Push mode with the Message and Wrap Screens

The following figure shows the workflow for agents who use push mode and are required to view the Message and Wrap screens:



To duplicate this workflow with the E-Mail Manager API, use the following commands:

1. Log in with the `agentConnect` command.

2. Use the `eventRegister` command to register for Type 2 events for the agent's queue.

3. Make the agent routable using the `agentMakeRoutable` command.

4. When the agent receives a Type 2 event for a message on the queue, use the `msgOpenRouted` command to open the message. The message is in the `agentInRead` state.

5. Open the message for response using the `msgOpenForResponse` command.

6. Send a response using the `msgSendMarkForArchiving` command. (The `msgSendKeepCurrent` command does not take you to wrap state.)

7. Wrap the case and use the `msgExit` command to close the message.

8. Repeat steps 4 through 7 for other messages.

cem.api.agentConnect Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentConnect command is used to authenticate and connect an agent to the E-Mail Manager instance.

Caution: An agent can only log in once concurrently. An agent cannot have multiple concurrent sessions.

Agent State and Mode Implications

After the agentConnect command successfully executes, the agent state is working and the agent mode is nonpush.

Input Data

Input data for the `agentConnect` command includes the following:

`userName` - The login name for the agent.

`userPassword` - The agent's password.

Response Data

The agentConnect response contains multiple parameters containing data about the agent.

Note: Many of the parameters in the agentInfo struct are also contained in the roleInfo struct; the values for the agent override the values for the role.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentId` - The unique identifier for the agent.

`agentInfo` - The agentInfo field is a struct that contains multiple fields with data about the agent.

- `agentInfo.webLocale` - The location of the agent.
- `agentInfo.usePDR` - Whether the agent is configured to use his distribution rule (1) or not (0).
- `agentInfo.firstName` - The agent's first name.
- `agentInfo.sig3` - The agent's third signature.
- `agentInfo.sig2` - The agent's second signature.
- `agentInfo.sig1` - The agent's first signature.
- `agentInfo.hasQueue` - Whether the agent has a message queue (1) or not (0).
- `agentInfo.choosable` - Whether, in the E-Mail Manager Agent Desktop, the agent's queue is available through the Queue Chooser.
- `agentInfo.loginEnabled` - Whether or the agent can log in to E-Mail Manager (1) or not (0).
- `agentInfo.mailLocale` - The language used by the agent for messages.
- `agentInfo.overloadEscalationThreshold` - The number of messages over which the queue is considered overloaded, meaning additional messages will be escalated. A value of -1 means that there is no threshold.
- `agentInfo.permanentAgent` - Whether the agent is permanent and cannot be deleted (1) or not (0).
- `agentInfo.lastName` - The agent's last name.

- `agentInfo.alias` - The agent's alias.
- `agentInfo.id` - The agent's ID.
- `agentInfo.automaticOverloadEnabled` - Whether overload escalation is enabled for the agent (1) or not (0).
- `agentInfo.defaultEmail` - The agent's default e-mail address.
- `agentInfo.description` - The description of the agent.
- `agentInfo.autoSaveInterval` - The number of seconds between automatic saving of response drafts.
- `agentInfo.overloadWarningThreshold` - The number of messages over which there is a warning that the queue is close to being overloaded. A value of -1 means that there is no threshold.
- `agentInfo.timeZone` - The time zone for the agent.
- `agentInfo.reassignable` - Whether other agents can reassign messages to this agent (1) or not (0).
- `agentInfo.automaticEscalationEnabled` - Whether time-based escalation is enabled for the agent (1) or not (0).

`agentState` - The agentState field is a struct that contains fields with data about the agent's state. See the ‘Agent States’ section in this guide for more information about agent states.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode. See the ‘Agent Mode’ section in this guide for more information about agent modes.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

`roleInfo` - The roleInfo field is a struct that contains multiple fields with data about the agent's associated role.

- `roleInfo.name` - The name of the role.
- `roleInfo.mailLocale` - The language used by the agent's based on the role for messages.

- `roleInfo.webLocale` - The location of agents based on the role.
- `roleInfo.alias` - The default alias of agents based on the role.
- `roleInfo.defaultFlag` - Whether the role is provided by the default E-Mail Manager installation (1) or not (0).
- `roleInfo.defaultEmail` - The default notification e-mail address of agent's based on the role.
- `roleInfo.uiRoleFlags` - A character string which contains a large number (currently 161) of flags stating whether certain actions or functionality is available for the agent. A Y stands for Yes (true) while an N stands for No (false). It is up to the client application to parse and understand the applicable flags and not allow the agent to perform an operation that is prohibited by the role flags. The following list identifies the locations in the string and descriptions of the role flags pertinent to a client application using the API:
  - 20 - Allow to assign outside of peer skill groups.
  - 27 - Allow to CC/BCC to other users.
  - 47 - Allow to see the special Unassigned queue.
  - 66 - Allow to see their Skill Group Queues.
  - 67 - Allow to use the CC/BCC fields.
  - 68 - Allow to edit responses.
  - 71 - Allow to see their Personal Queue.
  - 74 - Allow Unarchive.
  - 81 - Allow to use Personal status screen.
  - 82 - Allow to originate individual mails.
  - 83 - Allow to create inbound messages.
  - 85 - Allow to use a Queue Chooser .
  - 86 - Automatically quote original message in response screen.
  - 87 - Do not show original message field in response screen.
  - 88 - Require push mode.
  - 89 - Display Message screen before allowing response.

    - 96 - Allow to unlock messages.
    - 97 - Allow to use wrap screen.
    - 98 - Show agent statistics on list, status, read, response, and wrap screens.
    - 103 - Allow to reassign to agents.
- `roleInfo.permanentFlag` - Whether the role is permananet and cannot be deleted (1) or not (0).
- `roleInfo.sig3` - The default third signature for agent's based on the role.
- `roleInfo.sig2` - The default second signature for agent's based on the role.
- `roleInfo.sig1` - The default first signature for agent's based on the role.
- `roleInfo.description` - The description of the role.
- `roleInfo.timeZone` - The time zone for agent's based on the role.

cem.api.agentDisconnect Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentDisconnect command is used to logout and disconnect an agent from the E-Mail Manager instance.

Agent State and Mode Implications

After the agentDisconnect command successfully executes, the agent state and agent mode become logged off.

Input Data

Input data for the `agentDisconnect` command includes the following:

`reasonCode` - The reason, as a code, that the agent is logging out. You can get the valid reason codes by using the agentGetLogoutReasons command.

Response Data

The agentDisconnect response contains only a returnCode parameter.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

cem.api.agentDisconnectDuplicateAndConnect Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentDisconnectDuplicateAndConnect command disconnects a duplicate agent who is already connected, then connects the agent back to the E-Mail Manager instance.

Note: This command should only be used if the agentConnect command results in an ALREADY_LOGGED_IN fault.

Agent State and Mode Implications

After the agentDisconnectDuplicateAndConnect command successfully executes, the agent state is working and the agent mode is nonpush.

Input Data

Input data for the `agentDisconnectDuplicateAndConnect` command includes the following:

`userName` - The login name for the agent.

`userPassword` - The agent's password.

Response Data

The agentDisconnectDuplicateAndConnect response contains multiple parameters containing data about the agent.

Note: Many of the parameters in the agentInfo response are also contained in the roleInfo response; in E-Mail Manager, the values for the agent override the values for the role.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentId` - The unique identifier for the agent.

`agentInfo` - The agentInfo field is a struct that contains multiple fields with data about the agent.

- `agentInfo.webLocale` - The location of the agent.
- `agentInfo.usePDR` - Whether the agent is configured to use his distribution rule (1) or not (0).
- `agentInfo.firstName` - The agent's first name.
- `agentInfo.sig3` - The agent's third signature.
- `agentInfo.sig2` - The agent's second signature.
- `agentInfo.sig1` - The agent's first signature.
- `agentInfo.hasQueue` - Whether the agent has a message queue (1) or not (0).
- `agentInfo.choosable` - Whether, in the E-Mail Manager Agent Desktop, the agent's queue is available through the Queue Chooser.
- `agentInfo.loginEnabled` - Whether or the agent can log in to E-Mail Manager (1) or not (0).
- `agentInfo.mailLocale` - The language used by the agent for messages.
- `agentInfo.overloadEscalationThreshold` - The number of messages over which the queue is considered overloaded, meaning additional messages will be escalated. A value of -1 means that there is no threshold.

- `agentInfo.permanentAgent` - Whether the agent is permanent and cannot be deleted (1) or not (0).
- `agentInfo.lastName` - The agent's last name.
- `agentInfo.alias` - The agent's alias.
- `agentInfo.id` - The agent's ID.
- `agentInfo.automaticOverloadEnabled` - Whether overload escalation is enabled for the agent (1) or not (0).
- `agentInfo.defaultEmail` - The agent's default e-mail address.
- `agentInfo.description` - The description of the agent.
- `agentInfo.autoSaveInterval` - The number of seconds between automatic saving of response drafts.
- `agentInfo.overloadWarningThreshold` - The number of messages over which there is a warning that the queue is close to being overloaded. A value of -1 means that there is no threshold.
- `agentInfo.timeZone` - The time zone for the agent.
- `agentInfo.reassignable` - Whether other agents can reassign messages to this agent (1) or not (0).
- `agentInfo.automaticEscalationEnabled` - Whether time-based escalation is enabled for the agent (1) or not (0).

`agentState` - The agentState field is a struct that contains fields with data about the agent's state. See the ‘Agent States’ section in this guide for more information about agent states.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode. See the ‘Agent Mode’ section in this guide for more information about agent modes.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

`roleInfo` - The roleInfo field is a struct that contains multiple fields with data about the agent's associated role.

- `roleInfo.name` - The name of the role.
- `roleInfo.mailLocale` - The language used by the agent's based on the role for messages.
- `roleInfo.webLocale` - The location of agents based on the role.
- `roleInfo.alias` - The default alias of agents based on the role.
- `roleInfo.defaultFlag` - Whether the role is provided by the default E-Mail Manager installation (1) or not (0).
- `roleInfo.defaultEmail` - The default notification e-mail address of agent's based on the role.
- `roleInfo.uiRoleFlags` - A character string which contains a large number (currently 161) of flags stating whether certain actions or functionality is available for the agent. A Y stands for Yes (true) while an N stands for No (false). It is up to the client application to parse and understand the applicable flags and not allow the agent to perform an operation that is prohibited by the role flags. The following list identifies the locations in the string and descriptions of the role flags pertinent to a client application using the API:
  - 20 - Allow to assign outside of peer skill groups.
  - 27 - Allow to CC/BCC to other users.
  - 47 - Allow to see the special Unassigned queue.
  - 66 - Allow to see their Skill Group Queues.
  - 67 - Allow to use the CC/BCC fields.
  - 68 - Allow to edit responses.
  - 71 - Allow to see their Personal Queue.
  - 74 - Allow Unarchive.
  - 81 - Allow to use Personal status screen.
  - 82 - Allow to originate individual mails.
  - 83 - Allow to create inbound messages.
  - 85 - Allow to use a Queue Chooser .
  - 86 - Automatically quote original message in response screen.

    - 87 - Do not show original message field in response screen.
    - 88 - Require push mode.
    - 89 - Display Message screen before allowing response.
    - 96 - Allow to unlock messages.
    - 97 - Allow to use wrap screen.
    - 98 - Show agent statistics on list, status, read, response, and wrap screens.
    - 103 - Allow to reassign to agents.
- `roleInfo.permanentFlag` - Whether the role is permananet and cannot be deleted (1) or not (0).
- `roleInfo.sig3` - The default third signature for agent's based on the role.
- `roleInfo.sig2` - The default second signature for agent's based on the role.
- `roleInfo.sig1` - The default first signature for agent's based on the role.
- `roleInfo.description` - The description of the role.
- `roleInfo.timeZone` - The time zone for agent's based on the role.

cem.api.agentGetLogoutReasons Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentGetLogoutReasons command is used to retrieve the possible logout reasons, to be used for the agentDisconnect command.

Input Data

No input data is required for the `agentGetLogoutReasons` command.

Response Data

The agentGetLogoutReasons response contains an array of structs, each containing the name and code for a logout reason.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`LogoutReasons` - The LogoutReasons field contains an array of structs, each containing the name and code for a logout reason.

- `LogoutReasons[].name` - The name of the logout reason.
- `LogoutReasons[].code` - The code for the logout reason.

cem.api.agentGetNotReadyReasons Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentGetNotReadyReasons command is used to retrieve the possible not ready reasons, to be used for the agentMakeNotReady command.

Input Data

No input data is required for the `agentGetNotReadyReasons` command.

Response Data

The agentGetNotReadyReasons response contains an array of structs, each containing the name and code for a not ready reason.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`NotReadyReasons` - The NotReadyReasons field contains an array of structs, each containing the name and code for a not ready reason.

- `NotReadyReasons[].name` - The name of the not ready reason.
- `NotReadyReasons[].code` - The code for the not ready reason.

cem.api.agentGetProperties Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentGetProperties command is used to retrieve information about the agent's personal and role settings.

For more information, see the *Cisco E-Mail Manager Administration Guide*.

Input Data

No input data is required for the `agentGetProperties` command.

Response Data

The agentGetProperties response contains data about the agent's personal and role settings.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`agentInfo` - The agentInfo field is a struct that contains multiple fields with data about the agent.

- `agentInfo.webLocale` - The location of the agent.
- `agentInfo.usePDR` - Whether the agent is configured to use his distribution rule (1) or not (0).
- `agentInfo.firstName` - The agent's first name.
- `agentInfo.sig3` - The agent's third signature.
- `agentInfo.sig2` - The agent's second signature.
- `agentInfo.sig1` - The agent's first signature.
- `agentInfo.hasQueue` - Whether the agent has a message queue (1) or not (0).
- `agentInfo.choosable` - Whether, in the E-Mail Manager Agent Desktop, the agent's queue is available through the Queue Chooser.
- `agentInfo.loginEnabled` - Whether or the agent can log in to E-Mail Manager (1) or not (0).
- `agentInfo.mailLocale` - The language used by the agent for messages.
- `agentInfo.overloadEscalationThreshold` - The number of messages over which the queue is considered overloaded, meaning additional messages will be escalated. A value of -1 means that there is no threshold.
- `agentInfo.permanentAgent` - Whether the agent is permanent and cannot be deleted (1) or not (0).
- `agentInfo.lastName` - The agent's last name.
- `agentInfo.alias` - The agent's alias.
- `agentInfo.id` - The agent's ID.
- `agentInfo.automaticOverloadEnabled` - Whether overload escalation is enabled for the agent (1) or not (0).
- `agentInfo.defaultEmail` - The agent's default e-mail address.
- `agentInfo.description` - The description of the agent.
- `agentInfo.autoSaveInterval` - The number of seconds between automatic saving of response drafts.

- `agentInfo.overloadWarningThreshold` - The number of messages over which there is a warning that the queue is close to being overloaded. A value of -1 means that there is no threshold.
- `agentInfo.timeZone` - The time zone for the agent.
- `agentInfo.reassignable` - Whether other agents can reassign messages to this agent (1) or not (0).
- `agentInfo.automaticEscalationEnabled` - Whether time-based escalation is enabled for the agent (1) or not (0).

`roleInfo` - The roleInfo field is a struct that contains multiple fields with data about the agent's associated role.

- `roleInfo.name` - The name of the role.
- `roleInfo.mailLocale` - The language used by the agent's based on the role for messages.
- `roleInfo.webLocale` - The location of agents based on the role.
- `roleInfo.alias` - The default alias of agents based on the role.
- `roleInfo.defaultFlag` - Whether the role is provided by the default E-Mail Manager installation (1) or not (0).
- `roleInfo.defaultEmail` - The default notification e-mail address of agent's based on the role.
- `roleInfo.uiRoleFlags` - A character string which contains a large number (currently 161) of flags stating whether certain actions or functionality is available for the agent. A Y stands for Yes (true) while an N stands for No (false). It is up to the client application to parse and understand the applicable flags and not allow the agent to perform an operation that is prohibited by the role flags. The following list identifies the locations in the string and descriptions of the role flags pertinent to a client application using the API:
  - 20 - Allow to assign outside of peer skill groups.
  - 27 - Allow to CC/BCC to other users.
  - 47 - Allow to see the special Unassigned queue.
  - 66 - Allow to see their Skill Group Queues.
  - 67 - Allow to use the CC/BCC fields.
  - 68 - Allow to edit responses.

- 71 - Allow to see their Personal Queue.
- 74 - Allow Unarchive.
- 81 - Allow to use Personal status screen.
- 82 - Allow to originate individual mails.
- 83 - Allow to create inbound messages.
- 85 - Allow to use a Queue Chooser .
- 86 - Automatically quote original message in response screen.
- 87 - Do not show original message field in response screen.
- 88 - Require push mode.
- 89 - Display Message screen before allowing response.
- 96 - Allow to unlock messages.
- 97 - Allow to use wrap screen.
- 98 - Show agent statistics on list, status, read, response, and wrap screens.
- 103 - Allow to reassign to agents.

- `roleInfo.permanentFlag` - Whether the role is permananet and cannot be deleted (1) or not (0).
- `roleInfo.sig3` - The default third signature for agent's based on the role.
- `roleInfo.sig2` - The default second signature for agent's based on the role.
- `roleInfo.sig1` - The default first signature for agent's based on the role.
- `roleInfo.description` - The description of the role.
- `roleInfo.timeZone` - The time zone for agent's based on the role.

cem.api.agentGetState Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentGetState command is used to retrieve information about the agent and the agent's queue. The same result data is returned by the agentConnect command, which you should use if the agent is not already connected. Use the agentGetState command to retrieve updated data after the agent is connected.

Input Data

No input data is required for the `agentGetState` command.

Response Data

The agentGetState response contains multiple fields containing data about the agent's state and mode.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`agentState` - The agentState field is a struct that contains fields with data about the agent's state. See the ‘Agent States’ section in this guide for more information about agent states.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode. See the ‘Agent Mode’ section in this guide for more information about agent modes.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

cem.api.agentGetStatistics Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentGetStatistics command retrieves statistics about one or more agents.

Input Data

Input data for the `agentGetStatistics` command includes the following:

`type` - The indication of the agents for which you want to retrieve statistics: '0' for statistics of the agentIds listed in the agentList parameter; '1' for the agent issuing the command; '2' for all agents.

`agentList` - An array of agentIds identifying the agents for which you want statistics. This parameter is only required when the value of the type parameter is '0'. Each agentId must be specified within the array element.

Response Data

The agentGetStatistics response contains the following:

Parameters:

`agentStatistics` - An array of structs containing statistics about one or more agents.

- `agentStatistics[].escalations` - The number of messages the agent has escalated in the current day.

- `agentStatistics[].firstName` - The first name of the agent.
- `agentStatistics[].isLoggedIn` - Whether the agent is logged in (1) or not (0) to the Agent Desktop.
- `agentStatistics[].reassignments` - The number of messages the agent has reassigned in the current day.
- `agentStatistics[].totalReassignments` - The total number of reassignments made by the agent.
- `agentStatistics[].loginName` - The login name of the agent.
- `agentStatistics[].state` - The current state of the agent.
- `agentStatistics[].timeWrapping` - The total time during the current day that the agent has been in Wrap state.
- `agentStatistics[].timeInState` - The total time the agent has been in the current state.
- `agentStatistics[].mailWraps` - The number of messages the agent has handled through MailWrap during the current day.
- `agentStatistics[].timeInResponse` - The total time during the current day that the agent has been in Responding state.
- `agentStatistics[].timeInReading` - The total time during the current day that the agent has been in Read state.
- `agentStatistics[].timeLoggedIn` - The time that has elapsed since the agent logged in.
- `agentStatistics[].lastName` - The agent's last name.
- `agentStatistics[].timeInResponse` - The total time during the current day that the agent has been in Responding state.
- `agentStatistics[].agentId` - The unique identifier for the agent.
- `agentStatistics[].lastStateChange` - The time that the agent's state last changed.
- `agentStatistics[].responsesSent` - The total number of responses the agent has sent during the current day.
- `agentStatistics[].fullName` - The agent's full name.

cem.api.agentMakeNotReady Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentMakeNotReady command is used to make the agent's state NotReady, with a reason you supply.

Agent State and Mode Implications

After the agentMakeNotReady command successfully executes, the agent state becomes working, or if you specified a reason, not ready with the reason you supply.

Input Data

Input data for the `agentMakeNotReady` command includes the following:

`subreason` - An optional parameter identifying the reason the Agent State is to become NotReady. If no parameter is specified, the default reason, working, and code, 1, are used.

Response Data

The agentMakeNotReady response contains multiple fields containing data about the agent's state and mode.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentState` - The agentState field is a struct that contains fields with data about the agent's state. See the ‘Agent States’ section in this guide for more information about agent states.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode. See the ‘Agent Mode section in this guide for more information about agent modes.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

cem.api.agentMakeNotRoutable Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentMakeNotRoutable command is used to make an agent ineligible to be pushed messages routed through ICM software. After executing this command, the agent cannot use the msgOpenRouted command.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Agent State and Mode Implications

After the agentMakeRoutable command successfully executes, the agent mode becomes nonpush.

Input Data

No input data is required for the `agentMakeNotRoutable` command.

Response Data

The agentMakeNotRoutable response contains multiple fields containing data about the agent's state and mode.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentState` - The agentState field is a struct that contains fields with data about the agent's state.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

cem.api.agentMakeReady Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentMakeReady command is used to make the agent's state idle.

Agent State and Mode Implications

After the agentMakeReady command successfully executes, the agent state becomes idle.

Input Data

No input data is required for the `agentMakeReady` command.

Response Data

The agentMakeReady response contains multiple fields containing data about the agent's state and mode.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentState` - The agentState field is a struct that contains fields with data about the agent's state.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

cem.api.agentMakeRoutable Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The agentMakeRoutable command is used to make the agent eligible to be pushed messages. After executing this command, the agent must open messages using the msgOpenRouted command. This command must be used to make an agent eligible to be routed messages through ICM software.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Agent State and Mode Implications

After the agentMakeRoutable command successfully executes, the agent mode becomes push.

Input Data

No input data is required for the `agentMakeRoutable` command.

Response Data

The agentMakeRoutable response contains multiple fields containing data about the agent's state and mode.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 200 - OK with information. The command succeeded, but there was an exception while attempting to return certain data.

`agentState` - The agentState field is a struct that contains fields with data about the agent's state.

- `agentState.name` - The name of the agent's state.
- `agentState.code` - The code of the agent's state.

`agentMode` - The agentMode field is a struct that contains fields with data about the agent's mode.

- `agentMode.name` - The name of the agent's mode.
- `agentMode.code` - The code of the agent's mode.

cem.api.attachGetContent Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The attachGetContent command retrieves the content of the specified attachment.

Input Data

Input data for the `attachGetContent` command includes the following:

`attachID` - The unique identifier of the attachment, retrieved through the attachGetList command.

Response Data

The attachGetContent response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

encodedBody - The encoded text for the attachment. The application using the API is responsible for transforming the encoded text into the desired file format.

cem.api.attachGetList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The attachGetList command retrieves a list of attachments in the E-Mail Manager instance.

Note: The attachGetList retrieves information about the attachments, not the attachments themselves. To retrieve the attachments, you must use the attachGetContent command.

Input Data

No input data is required for the `attachGetList` command.

Response Data

The attachGetList response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`attachmentsList` - An array of structs containing information about attachments.

- `attachmentsList[].attType` - The type of attachment. The value is always 1.
- `attachmentsList[].attTime` - The time the attachment was added to E-Mail Manager.
- `attachmentsList[].attachmentId` - The unique identifier for the attachment.
- `attachmentsList[].attDesc` - The description of the attachment.
- `attachmentsList[].originalFileName` - The original file name of the attachment.
- `attachmentsList[].pksUser` - The login name of the user who added the attachment.

cem.api.attachRegister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The attachRegister command is used to register an attachment using the E-Mail Manager instance.

Note: The agent should be in session to run the `attachRegister` command.

Input Data

Input data for the `attachRegister` command includes the following:

`originalFileName` - The file name for the attachment.

`attDesc` - A description of the attachment.

`mimeType` - mimeType of the attachment.

`attachment` - The attachment in base64 format.

`charset` - An optional parameter specifying the character set for the attachment.

Response Data

The `attachRegister` response contains a returnCode and an attachmentId parameter.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return the following value, a fault is returned instead.

Possible value:

- 0 - Command Succeeded.

`attachmentId` – The unique identifier for the attachment.

cem.api.attachUnregister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The attachUnregister command is used to un-register an attachment using the E-Mail Manager instance.

Note: The agent should be in session to run the `attachUnregister` command.

Input Data

Input data for the `attachUnregister` command includes the following:

`attachmentId` - The attachment ID of the attachment to un-register.

Response Data

The `attachUnregister` response contains only a returnCode parameter.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return the following values, a fault is returned instead.

Possible value are:

- 0 - Command Succeeded.
- 100 - No data found.

cem.api.catsChangeForMsg Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The catsChangeForMsg command changes the categories associated with a specific message.

Input Data

Input data for the `catsChangeForMsg` command includes the following:

`queueId` - The unique identifier for the queue.

`messageKey` - The unique identifier of the message.

`catsOriginal` - The identifiers of the categories originally associated with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.

`catsSelected` - The identifiers of the categories to associate with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.

Response Data

The catsChangeForMsg response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.catsGetList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The catsGetList command is used to retrieve a list of all categories defined for the E-Mail Manager instance.

Input Data

No input data is required for the `catsGetList` command.

Response Data

The catsGetList response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`categoryList - An array of structs containing information about message categories.`

- `categoryList[].pksCatid` - The identifier of the category.
- `categoryList[].description` - The description of the category.
- `categoryList[].pksCat` - The category.

cem.api.catsGetListForMsg Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The catsGetListForMsg command retrieves a list of categories associated with a specific message.

Input Data

Input data for the `catsGetListForMsg` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The catsGetListForMsg response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`categoryList` - An array of structs containing information about message categories.

- `categoryList[].pksCatid` - The identifier of the category.
- `categoryList[].description` - The description of the category.
- `categoryList[].pksCat` - The category.

cem.api.eventRegister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The eventRegister command is used to register to receive a specific type of event. To no longer receive the event, use the eventUnregister command. The input data may include queueScope or msgScope parameters, but not both.

Caution: The client application design should consider that the API Server does not know when an application through which a user has registered for an event terminates unexpectedly.

Input Data

Input data for the `eventRegister` command includes the following:

`eventType` - The type of event for which you are registering. Use 1, 2, 3, or 4.

queueScope - One or more queueGUIDs for which to register for events. The queue IDs are contained as values in an array. Each queueGUID must be specified within the array element. Instead of queue IDs, you can use the value all, which registers for the event for all queues. The event type '1' must have a queueScope parameter. For the event type '4', the queue's listed are the personal queues of the agent's to monitor.

`msgScope` - One or more keys for messages for which to register for events. The message keys are contained as values in an array. Each message key must be specified within the array element. Instead of message IDs, you can use the value all, which registers for the event for all messages.

Response Data

The eventRegister response provides the eventRegistrationId.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 101 - The registration request partially overlapped with an existing registration (by the same application); the new registration request(s) were successful.

`eventRegistrationId` - The unique identifier for event registration. The eventRegistrationId must be used as a parameter in the eventUnregister command, to stop receiving the type of event.

cem.api.eventUnregister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The eventUnregister command is used to stop receiving a specific type of event.

Input Data

Input data for the `eventUnregister` command includes the following:

`eventIds` - An array of eventRegistrationIds for the events types that you want to stop receiving. If no `eventIds` are specified, the command unregisters all outstanding event registrations.

Response Data

The eventUnregister response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.msgClaim Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgClaim command is used to claim a message from a skill group queue or another agent's personal queue, to the agent's personal queue.

Input Data

Input data for the `msgClaim` command includes the following:

`messageKey` - The unique identifier of the message.

`queueId` - The unique identifier for the queue.

`unlockMessage` - An optional indication of whether to unlock the message if it is locked (1) or not (0). A message is locked if it is in another agent's personal queue and that agent has the message open. If this parameter is not specified, the default value (0) is used.

Response Data

The msgClaim response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.

- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.

- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.

- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.

- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.

- 14 - System messages used internally. These messages should not be utilized for any purposes.

- 15 - Outgoing messages to a mailing list.

- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.

- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.

- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.

- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.

- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.

- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.

- 24 - Reply messages sent through MailTrack.

- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.

  - 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.

  - 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.

  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.

- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.

- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.

- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.

- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).

- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).

- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).

- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.

- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).

- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).

- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgCreateNewStub Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgCreateNewStub command is used to create a new message. This new message serves as a temporary incoming message, which you can open using the msgOpenForResponseCommand, then send a response using the msgSendKeepCurrent or msgSendMarkForArchiving command. The effect is to send a new outgoing message; the E-Mail Manager architecture requires that you create this fake incoming message to respond to in order to send a new outbound message.

Input Data

Input data for the `msgCreateNewStub` command includes the following:

`subject` - An optional parameter containing the subject of the new message. If the parameter is not included, a single space character is used by default.

`text` - An optional parameter containing the text of the new message. If the parameter is not included, a single space character is used by default.

`extendedAttributes` - An optional parameter that contains an array of structs, each containing the parameters attributeName and attributeValue, which can be used for information as you need.

`attributeName` - The name of the attribute.

`attributeValue` - The value of the attribute.

Response Data

The msgCreateNewStub response contains the unique identifier for the new message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`messageKey` - The unique identifier of the message.

cem.api.msgDelete Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgDelete command is used to delete a specific e-mail message from E-Mail Manager.

Input Data

Input data for the `msgDelete` command includes the following:

`messageKey` - The unique identifier of the message.

`queueId` - The unique identifier for the queue.

Response Data

The msgDelete response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.msgEscalate Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgEscalate command is used to place a message in different queue and raise the message's priority by one increment, if it can be raised; if the priority is at the maximum, it is not raised. To place the message in a different queue without raising the message's priority, use the msgReassign command.

Input Data

Input data for the `msgEscalate` command includes the following:

`messageKey` - The unique identifier of the message.

`newOwner` - The global unique identifier (GUID) of the queue the message is being reassigned to.

`owner` - The global unique identifier (GUID) of the queue the message is being reassigned from.

Response Data

The msgEscalate response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.
  - 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
  - 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.

- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.

  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).
- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).

- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgExit Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgExit command is used to exit a specific e-mail message that has been opened without taking further action on that message. The msgExit command is also used to end the wrapping of a message; this command is the only way to change the agent's state from BusyWrapping.

Input Data

No input data is required for the `msgExit` command.

Response Data

The msgExit response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.msgGetContent Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgGetContent command is used to retrieve the contents of a specific message. The message content retrieved is immutable; you cannot take action on a message retrieved through this command.

Input Data

Input data for the `msgGetContent` command includes the following:

`messageKey` - The unique identifier of the message.

`isOld` - An optional parameter indicating whether to retrieve messages that have been moved to the LAMBDA database (1) or not (0). The default value when not specified is 0.

Response Data

The msgGetContent response contains immutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailImmutableData` - The MailImmutableData parameter is a struct that contains fixed data about a single message. A single response can contain multiple MailImmutableData fields, depending on the command executed and the number of messages for which data is returned.

- `MailImmutableData.headerEncoding` - The character set and encoding specified by the header as detected by Euclid.
- `MailImmutableData.bodyEncoding` - The character set of the message as detected by Euclid.
- `MailImmutableData.headerEncodingDeclared` - The first declared charset found in a mail header. But if any other charsets are found in a header and are different from the first, then it will be set to UTF-8
- `MailImmutableData.bodyEncodingDeclared` - The charset declared in the Content-Type header.
- `MailImmutableData.fromRaw` - The sender's complete e-mail address, in full form.
- `MailImmutableData.MToRaw` - The To: headers in full form, which may include multiple e-mail address.
- `MailImmutableData.fromEaten` - The sender's e-mail address with all non-address text removed.
- `MailImmutableData.MSubject` - The Subject: header of the e-mail message.
- `MailImmutableData.mailMultipart` - A struct containing the following parameters, referring to one of the parts of a multipart MIME message:
  - `MailImmutableData.mailMultipart.headers` - The headers of the e-mail message.
  - `MailImmutableData.mailMultipart.body` - The body of the e-mail message.
  - `MailImmutableData.mailMultipart.textBody` - An optional parameter containing a concatenation of all the text/plain parts of the message if it is a true multi-part MIME message. This parameter is not present if the message is a not true multi-part message or none of the message parts is text/plain.
  - `MailImmutableData.mailMultipart.mailParts` - An optional parameter present only when the message is a true multi-part MIME message. An array

of structs, each of which contains the following parameters pertaining to one MIME part of the message:

- `MailImmutableData.mailMultipart.mailParts[].body` - The body of the e-mail message.
- `MailImmutableData.mailMultipart.mailParts[].description` - Either the value of the Content-ID header or filename parameter of the Content-Disposition header.
- `MailImmutableData.mailMultipart.mailParts[].contentType` - The type of message.
- `MailImmutableData.mailMultipart.mailParts[].fileName` - The filename parameter from the Content-Disposition header.
- `MailImmutableData.mailMultipart.mailParts[].charset` - The character set used.
- `MailImmutableData.mailMultipart.mailParts[].encoding` - The encoding used.

- `MailImmutableData.messageKey` - The unique identifier of the message.
- `MailImmutableData.locale` - The language and character set of the message, separated by a comma, as detected by Euclid. Typically, you would use the value of this parameter in the 'locale' input parameter when sending a response.
- `MailImmutableData.timeDone` - The time the message was stored in E-Mail Manager, in UNIX time.
- `MailImmutableData.accessMode` - The ownership classification of the message currently set based on the message type. For external incoming messages, the value is 3; for outbound messages the value is 2; for internal messages, the value is 1; and for utility messages, the value if 0.
- `MailImmutableData.MDate` - The Date header of the e-mail message, translated to UNIX time.
- `MailImmutableData.MReplyTo` - The ReplyTo header of the e-mail message.
- `MailImmutableData.trackingNumber` - The tracking number of the thread of which the message is a part.
- `MailImmutableData.message` - The e-mail message as received, with all headers.

cem.api.msgGetExtendedAttributes Command

This topic contains the following sections:

Overview

Input Data

Response Data

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgGetExtendedAttributes command is used to retrieve the extended attributes of a message. A message may be assigned extended attributes by a low level rule; you can also create extended attributes with the msgCreateNewStub command.

Input Data

Input data for the `msgGetExtendedAttributes` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgGetExtendedAttributes contains the extended attribute values for the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`extendedAttributes` - The extendedAttributes field contains an array of structs, each containing the attribute name and value.

- `extendedAttributes[].name` - The name of the attribute
- `extendedAttributes[].value` - The value of the attribute.

cem.api.msgGetLatestResponseDraft Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgGetLatestResponseDraft command is used to retrieve information about the latest draft response. Draft responses can be created using the msgSaveAsDraft command.

Note: If an agent saves a draft using the msgSaveAsDraft command, then opens that draft and sends it as a response using the msgSendKeepCurrent command, then the latest draft saved in the database and retrieved by this command is not the sent response, but the previously saved draft.

Input Data

Input data for the `msgGetLatestResponseDraft` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgGetLatestResponseDraft contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`to` - The To field contains an array of e-mail address to which the message is to be sent.

`cc` - The cc field contains an array of e-mail address to which the message is to be copied.

`bcc` - The bcc field contains an array of e-mail address to which the message is to be copied without other recipients knowing.

`subject` - The subject field contains text of the subject line of the e-mail message.

`text` - The text field contains body text of the e-mail message.

`draftAttachments` - The draftAttachments field contains an array of structs, each containing information about an attachment.

- `draftAttachments[].attachmentId` - The unique identifier for the attachment.
- `draftAttachments[].originalFileName` - The file name of the attachment.
- `draftAttachments[].attachmentDescription` - The description of the attachment.
- `draftAttachments[].rowTimestamp` - The time the attachment was added to E-Mail Manager.

cem.api.msgGetResponses Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgGetResponses command is used to return information about responses sent to a message.

Input Data

Input data for the `msgGetResponses` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgGetResponses contains data about the responses that were sent to the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`messageResponses` - The messageResponses field contains an array of structs, each containing data about a response.

- `messageResponses[].timeDone` - The time the response was sent.
- `messageResponses[].pksOwnerid` - The identifier of the owner of the message.
- `messageResponses[].messageKey` - The unique identifier of the message.
- `messageResponses[].type` - The type of message the response is. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.
  - 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
  - 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
  - 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
  - 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
  - 14 - System messages used internally. These messages should not be utilized for any purposes.
  - 15 - Outgoing messages to a mailing list.
  - 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.

- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
- 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

cem.api.msgGetStatus Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgGetStatus command is used to retrieve mutable information about a message. You cannot take action on a message retrieved through this command.

Input Data

Input data for the `msgGetStatus` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgGetStatus response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain

multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.
  - 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
  - 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
  - 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
  - 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.

- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
- 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.

- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.

- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.

- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.

- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).

- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).

- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).

- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.

- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).

- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).

- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).

- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).

- `MailMutableData.commentTag` - The most recent note added to the message, if any.

- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).

- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).

- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgMarkForArchiving Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgMarkForArchiving command is used to indicate that a message should be archived.

Note: You cannot retrieve messageKey values for archived messages. Therefore, to use the msgUnarchive command, you must save the messageKey value of the message you are marking for archiving.

Input Data

Input data for the `msgMarkForArchiving` command includes the following:

`messageKey` - The unique identifier of the message.

`queueId` - The unique identifier for the queue.

Response Data

The msgMarkForArchiving response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.msgOpenByQueueAndKey Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgOpenByQueueAndKey command is used to open a specific message in a specific queue. When this command is executed, the message is moved to the queue of the agent issuing the command, and that agent then owns the message. You do not need to use the msgClaim command; the claim is executed automatically.

Agent State and Mode Implications

To execute this command, the agent must be in the state `idle`. Agents receiving messages routed by ICM software cannot use this command. After the agent executes this command:

If the agent's role requires that the agent reads messages before responding, the agent's state becomes `read`.

If the agent's role does not require that the agent reads message before responding, the agent's state become `responding`.

Message State Implications

The agent can have no other message in the state openForRead or openForResponse when executing this command. After the agent executes this command:

If the agent's role requires that the agent reads messages before responding, the message's state becomes openForRead.

If the agent's role does not require that the agent reads message before responding, the message's state become openForResponse.

Input Data

Input data for the msgOpenByQueueAndKey command includes the following:

messageKey - The unique identifier of the message.

queueId - The unique identifier for the queue.

Response Data

The msgOpenByQueueAndKey response contains mutable and immutable data about the message.

Parameters:

returnCode - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

MailMutableData - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- MailMutableData.replied - Whether the message has been replied to (1) or not (0).
- MailMutableData.agentInRead - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- MailMutableData.manualEscalateToQueue - Whether the message was escalated to the current queue (1) or not (0).
- MailMutableData.newToQueue - Whether the message has never been read in the current queue (1) or has been (0).
- MailMutableData.type - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.

- 1 - Messages from customers that are currently still active.
- 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
- 6 - Reply messages that are in a draft state.
- 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
- 9 - Reply messages that have been sent successfully and remain in the primary database.
- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.

- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
- 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).

- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).
- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.

- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

`MailImmutableData` - The MailImmutableData parameter is a struct that contains fixed data about a single message. A single response can contain multiple MailImmutableData fields, depending on the command executed and the number of messages for which data is returned.

- `MailImmutableData.headerEncoding` - The character set and encoding specified by the header as detected by Euclid.
- `MailImmutableData.bodyEncoding` - The character set of the message as detected by Euclid.
- `MailImmutableData.headerEncodingDeclared` - The first declared charset found in a mail header. But if any other charsets are found in a header and are different from the first, then it will be set to UTF-8
- `MailImmutableData.bodyEncodingDeclared` - The charset declared in the Content-Type header.
- `MailImmutableData.fromRaw` - The sender's complete e-mail address, in full form.
- `MailImmutableData.MToRaw` - The To: headers in full form, which may include multiple e-mail address.
- `MailImmutableData.fromEaten` - The sender's e-mail address with all non-address text removed.
- `MailImmutableData.MSubject` - The Subject: header of the e-mail message.
- `MailImmutableData.mailMultipart` - A struct containing the following parameters, referring to one of the parts of a multipart MIME message:
  - `MailImmutableData.mailMultipart.headers` - The headers of the e-mail message.

- `MailImmutableData.mailMultipart.body` - The body of the e-mail message.
- `MailImmutableData.mailMultipart.textBody` - An optional parameter containing a concatenation of all the text/plain parts of the message if it is a true multi-part MIME message. This parameter is not present if the message is a not true multi-part message or none of the message parts is text/plain.
- `MailImmutableData.mailMultipart.mailParts` - An optional parameter present only when the message is a true multi-part MIME message. An array of structs, each of which contains the following parameters pertaining to one MIME part of the message:
  - `MailImmutableData.mailMultipart.mailParts[].body` - The body of the e-mail message.
  - `MailImmutableData.mailMultipart.mailParts[].description` - Either the value of the Content-ID header or filename parameter of the Content-Disposition header.
  - `MailImmutableData.mailMultipart.mailParts[].contentType` - The type of message.
  - `MailImmutableData.mailMultipart.mailParts[].fileName` - The filename parameter from the Content-Disposition header.
  - `MailImmutableData.mailMultipart.mailParts[].charset` - The character set used.
  - `MailImmutableData.mailMultipart.mailParts[].encoding` - The encoding used.
- `MailImmutableData.messageKey` - The unique identifier of the message.
- `MailImmutableData.locale` - The language and character set of the message, separated by a comma, as detected by Euclid. Typically, you would use the value of this parameter in the 'locale' input parameter when sending a response.
- `MailImmutableData.timeDone` - The time the message was stored in E-Mail Manager, in UNIX time.
- `MailImmutableData.accessMode` - The ownership classification of the message currently set based on the message type. For external incoming messages, the value is 3; for outbound messages the value is 2; for internal messages, the value is 1; and for utility messages, the value if 0.
- `MailImmutableData.MDate` - The Date header of the e-mail message, translated to UNIX time.
- `MailImmutableData.MReplyTo` - The ReplyTo header of the e-mail message.

- `MailImmutableData.trackingNumber` - The tracking number of the thread of which the message is a part.

- `MailImmutableData.message` - The e-mail message as received, with all headers.

cem.api.msgOpenFromQueues Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgOpenFromQueues command is used to open a message from one or more queues. The message opened is the oldest incoming message with the highest priority that has not previously been opened. When this command is executed, the message is moved to the queue of the agent issuing the command, and that agent then owns the message. You do not need to use the msgClaim command; the claim is executed automatically.

Agent State and Mode Implications

To execute this command, the agent must be in the state `idle`. Agents receiving messages routed by ICM software cannot use this command. After the agent executes this command:

If the agent's role requires that the agent reads messages before responding, the agent's state becomes `read`.

If the agent's role does not require that the agent reads message before responding, the agent's state become `responding`.

Message State Implications

The agent can have no other message in the state `openForRead` or `openForResponse` when executing this command. After the agent executes this command:

If the agent's role requires that the agent reads messages before responding, the message's state becomes `openForRead`.

If the agent's role does not require that the agent reads message before responding, the message's state become `openForResponse`.

Input Data

Input data for the `msgOpenFromQueues` command includes the following:

`queueList` - An array of queueIds identifying the queues for which you want statistics. This parameter is only required when the value of the type parameter is '0'. Each queueId must be specified within the array element.

Response Data

The msgOpenFromQueues response contains mutable and immutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.

- 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
- 6 - Reply messages that are in a draft state.
- 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
- 9 - Reply messages that have been sent successfully and remain in the primary database.
- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.

- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
- 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).

- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).

- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).

- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.

- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).

- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

`MailImmutableData` - The MailImmutableData parameter is a struct that contains fixed data about a single message. A single response can contain multiple MailImmutableData fields, depending on the command executed and the number of messages for which data is returned.

- `MailImmutableData.headerEncoding` - The character set and encoding specified by the header as detected by Euclid.

- `MailImmutableData.bodyEncoding` - The character set of the message as detected by Euclid.

- `MailImmutableData.headerEncodingDeclared` - The first declared charset found in a mail header. But if any other charsets are found in a header and are different from the first, then it will be set to UTF-8

- `MailImmutableData.bodyEncodingDeclared` - The charset declared in the Content-Type header.

- `MailImmutableData.fromRaw` - The sender's complete e-mail address, in full form.

- `MailImmutableData.MToRaw` - The To: headers in full form, which may include multiple e-mail address.

- `MailImmutableData.fromEaten` - The sender's e-mail address with all non-address text removed.

- `MailImmutableData.MSubject` - The Subject: header of the e-mail message.

- `MailImmutableData.mailMultipart` - A struct containing the following parameters, referring to one of the parts of a multipart MIME message:

  - `MailImmutableData.mailMultipart.headers` - The headers of the e-mail message.

  - `MailImmutableData.mailMultipart.body` - The body of the e-mail message.

- `MailImmutableData.mailMultipart.textBody` - An optional parameter containing a concatenation of all the text/plain parts of the message if it is a true multi-part MIME message. This parameter is not present if the message is a not true multi-part message or none of the message parts is text/plain.

- `MailImmutableData.mailMultipart.mailParts` - An optional parameter present only when the message is a true multi-part MIME message. An array of structs, each of which contains the following parameters pertaining to one MIME part of the message:

  - `MailImmutableData.mailMultipart.mailParts[].body` - The body of the e-mail message.

  - `MailImmutableData.mailMultipart.mailParts[].description` - Either the value of the Content-ID header or filename parameter of the Content-Disposition header.

  - `MailImmutableData.mailMultipart.mailParts[].contentType` - The type of message.

  - `MailImmutableData.mailMultipart.mailParts[].fileName` - The filename parameter from the Content-Disposition header.

  - `MailImmutableData.mailMultipart.mailParts[].charset` - The character set used.

  - `MailImmutableData.mailMultipart.mailParts[].encoding` - The encoding used.

- `MailImmutableData.messageKey` - The unique identifier of the message.

- `MailImmutableData.locale` - The language and character set of the message, separated by a comma, as detected by Euclid. Typically, you would use the value of this parameter in the 'locale' input parameter when sending a response.

- `MailImmutableData.timeDone` - The time the message was stored in E-Mail Manager, in UNIX time.

- `MailImmutableData.accessMode` - The ownership classification of the message currently set based on the message type. For external incoming messages, the value is 3; for outbound messages the value is 2; for internal messages, the value is 1; and for utility messages, the value if 0.

- `MailImmutableData.MDate` - The Date header of the e-mail message, translated to UNIX time.

- `MailImmutableData.MReplyTo` - The ReplyTo header of the e-mail message.

- `MailImmutableData.trackingNumber` - The tracking number of the thread of which the message is a part.

- `MailImmutableData.message` - The e-mail message as received, with all headers.

cem.api.msgOpenForRead Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgOpenForRead command is used to open a message for reading. When managers track reading and responding time separately, this command is typically used before msgOpenForResponse command. In order for this command to succeed, the message being opened must be owned by the agent issuing the command; that is, the message must be in that agent's queue.

Agent State and Mode Implications

To execute this command, the agent state must be `idle`. After the agent executes this command, the agent's state becomes `read`.

An agent in push mode cannot use this command.

Message State Implications

The agent can have no other message in the state openForRead or openForResponse when executing this command. After the agent executes this command, the message's state becomes openForRead.

Input Data

Input data for the `msgOpenForRead` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgOpenForRead response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.

- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.

- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.

- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.

- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.

- 14 - System messages used internally. These messages should not be utilized for any purposes.

- 15 - Outgoing messages to a mailing list.

- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.

- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.

- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.

- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.

- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.

- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.

- 24 - Reply messages sent through MailTrack.

- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.

  - 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.

  - 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.

  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.

- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.

- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.

- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.

- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).

- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).

- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).

- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.

- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).

- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).

- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgOpenForResponse Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgOpenForResponse command is used to open a message so that the agent can respond to it using the msgSendKeepCurrent of msgSendMarkForArchiving command. When managers track reading and responding time separately, this command is typically used after msgOpenForRead command. In order for this command to succeed, the message being opened must be owned by the agent issuing the command; that is, the message must be in that agent's queue. To move a message to the agent's queue, use the msgClaim command.

Agent State and Mode Implications

To execute this command, the agent state must be `idle` or `read`. The agent can be in the state `read` and execute the msgOpenForResponse command on the same message that he is reading, but not for a different message. After the agent executes this command, the agent's state becomes `responding`.

An agent in push mode cannot use this command.

Message State Implications

The agent can have no other message in the state openForRead or openForResponse when executing this command. After the agent executes this command, the message's state becomes openForResponse.

Input Data

Input data for the msgOpenForResponse command includes the following:

messageKey - The unique identifier of the message.

Response Data

The msgOpenForResponse response contains mutable data about the message.

Parameters:

returnCode - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

MailMutableData - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- MailMutableData.replied - Whether the message has been replied to (1) or not (0).
- MailMutableData.agentInRead - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- MailMutableData.manualEscalateToQueue - Whether the message was escalated to the current queue (1) or not (0).
- MailMutableData.newToQueue - Whether the message has never been read in the current queue (1) or has been (0).
- MailMutableData.type - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.

- 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
- 6 - Reply messages that are in a draft state.
- 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
- 9 - Reply messages that have been sent successfully and remain in the primary database.
- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.

- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
- 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).

- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).

- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgOpenRouted Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgOpenRouted command is used to open the next message that has been routed to the agent through ICM software. The command must be, and can only be, used to open messages that are routed through ICM software.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Agent State and Mode Implications

In order to execute the msgOpenRouted command, the agent state must be idle, and the agent mode must be push. The agent can be in the state read and execute the msgOpenForResponse command on the same message that he is reading, but not for a different message.

Message State Implications

The agent can have no other message in the openForRead or openForResponse state. After an agent successfully executes this command, the message state for the routed message becomes openForRead or openForResponse, depending on whether the agent's role requires that message be read first or not. After successfully executing this command, the agent's state becomes read or responding, depending on whether the agent's role requires that message be read first or not.

Input Data

No input data is required for the `msgOpenRouted` command.

Response Data

The msgOpenRouted response contains mutable and immutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.

- 9 - Reply messages that have been sent successfully and remain in the primary database.
- 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
- 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.
- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

  - 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
  - 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
  - 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.
  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).
- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).

- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).
- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

`MailImmutableData` - The MailImmutableData parameter is a struct that contains fixed data about a single message. A single response can contain multiple MailImmutableData fields, depending on the command executed and the number of messages for which data is returned.

- `MailImmutableData.headerEncoding` - The character set and encoding specified by the header as detected by Euclid.
- `MailImmutableData.bodyEncoding` - The character set of the message as detected by Euclid.
- `MailImmutableData.headerEncodingDeclared` - The first declared charset found in a mail header. But if any other charsets are found in a header and are different from the first, then it will be set to UTF-8
- `MailImmutableData.bodyEncodingDeclared` - The charset declared in the Content-Type header.
- `MailImmutableData.fromRaw` - The sender's complete e-mail address, in full form.
- `MailImmutableData.MToRaw` - The `To:` headers in full form, which may include multiple e-mail address.
- `MailImmutableData.fromEaten` - The sender's e-mail address with all non-address text removed.
- `MailImmutableData.MSubject` - The `Subject:` header of the e-mail message.
- `MailImmutableData.mailMultipart` - A struct containing the following parameters, referring to one of the parts of a multipart MIME message:
  - `MailImmutableData.mailMultipart.headers` - The headers of the e-mail message.
  - `MailImmutableData.mailMultipart.body` - The body of the e-mail message.
  - `MailImmutableData.mailMultipart.textBody` - An optional parameter containing a concatenation of all the text/plain parts of the message if it is a true multi-part MIME message. This parameter is not present if the message is a not true multi-part message or none of the message parts is text/plain.
  - `MailImmutableData.mailMultipart.mailParts` - An optional parameter present only when the message is a true multi-part MIME message. An array of structs, each of which contains the following parameters pertaining to one MIME part of the message:
    - `MailImmutableData.mailMultipart.mailParts[].body` - The body of the e-mail message.

    - MailImmutableData.mailMultipart.mailParts[].description - Either the value of the Content-ID header or filename parameter of the Content-Disposition header.
    - MailImmutableData.mailMultipart.mailParts[].contentType - The type of message.
    - MailImmutableData.mailMultipart.mailParts[].fileName - The filename parameter from the Content-Disposition header.
    - MailImmutableData.mailMultipart.mailParts[].charset - The character set used.
    - MailImmutableData.mailMultipart.mailParts[].encoding - The encoding used.

- MailImmutableData.messageKey - The unique identifier of the message.
- MailImmutableData.locale - The language and character set of the message, separated by a comma, as detected by Euclid. Typically, you would use the value of this parameter in the 'locale' input parameter when sending a response.
- MailImmutableData.timeDone - The time the message was stored in E-Mail Manager, in UNIX time.
- MailImmutableData.accessMode - The ownership classification of the message currently set based on the message type. For external incoming messages, the value is 3; for outbound messages the value is 2; for internal messages, the value is 1; and for utility messages, the value if 0.
- MailImmutableData.MDate - The Date header of the e-mail message, translated to UNIX time.
- MailImmutableData.MReplyTo - The ReplyTo header of the e-mail message.
- MailImmutableData.trackingNumber - The tracking number of the thread of which the message is a part.
- MailImmutableData.message - The e-mail message as received, with all headers.

cem.api.msgReassign Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgReassign command is used to place a message in different queue. The msgReassign command does not raise the message's priority. To place the message in a different queue and raise the message's priority, use the msgEscalate command.

Input Data

Input data for the `msgReassign` command includes the following:

`messageKey` - The unique identifier of the message.

`newOwner` - The global unique identifier (GUID) of the queue the message is being reassigned to.

`owner` - The global unique identifier (GUID) of the queue the message is being reassigned from.

Response Data

The msgReassign response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.
  - 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
  - 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.

- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.

  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).
- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).

- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.msgSaveAsDraft Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgSaveAsDraft command is used to create a draft response to a message. The command does not send the response. Prior to using this command, you must open the message for response using the msgOpenForResponse, msgOpenByQueueAndKey, msgOpenRouted, or the msgOpenFromQueues command. The message opened for response is the message for which a draft reply is saved. After you save a draft, you can use the msgGetLatestResponseDraft command to retrieve it.

Agent State and Mode Implications

After executing this command, the agent state remains responding.

Input Data

Input data for the `msgSaveAsDraft` command includes the following:

`mailResponse` - The struct containing all parameters with data about the outgoing message.

- `forceNewTrackingNumber` - An optional parameter specifying whether to have replies to the message use a new tracking number (1) or not (0). If not specified, the value '0' is assumed.
- `from` - The e-mail address the message is being sent from.
- `trackingNumber` - The tracking number of the thread in which the message being sent is a part.

- `subject` - The subject of the message.
- `reply` - The text of the draft reply.
- `to` - The address(es) to send the message to.
- `cc` - An optional parameter with the address(es) to copy the message to.
- `bcc` - An optional parameter with the address(es) to blind copy the message to.
- `attachments` - An optional parameter with the Ids of attachments to be sent with the message.
- `locale` - An optional parameter specifying the language and character set, which are separated by a comma. Typically, when specifying this parameter, you should use the value of the 'locale' parameter returned in the MailImmutableData struct. When this parameter is not specified, the default value used is 'en-US,Windows-1252'.
- `catsOriginal` - The identifiers of the categories originally associated with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.
- `catsSelected` - The identifiers of the categories to associate with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.

Response Data

The msgSaveAsDraft response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.msgSendKeepCurrent Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgSendKeepCurrent command is used to send a response and keep the message in the agent's personal queue. Prior to using this command, you must open the message for response using the msgOpenForResponse, msgOpenByQueueAndKey, msgOpenRouted, or the msgOpenFromQueues command. The message opened for response is the message being replied to.

Agent State and Mode Implications

After the agent executes this command:

If the agent is configured to work with messages configured through ICM, the agent's state becomes `idle`.

Otherwise, the agent's state becomes `working`.

The agent's state never becomes `Wrap` after executing msgSendKeepCurrent.

Message State Implications

The message's state must be `openForResponse` when the agent executes this command. After the agent executes this command, the message is closed.

Input Data

Input data for the `msgSendKeepCurrent` command includes the following:

`mailResponse` - The struct containing all parameters with data about the outgoing message.

- `forceNewTrackingNumber` - An optional parameter specifying whether to have replies to the message use a new tracking number (1) or not (0). If not specified, the value '0' is assumed.
- `from` - The e-mail address the message is being sent from.
- `trackingNumber` - The tracking number of the thread in which the message being sent is a part.
- `subject` - The subject of the message.
- `reply` - The text of the draft reply.
- `to` - The address(es) to send the message to.
- `cc` - An optional parameter with the address(es) to copy the message to.
- `bcc` - An optional parameter with the address(es) to blind copy the message to.
- `attachments` - An optional parameter with the Ids of attachments to be sent with the message.
- `locale` - An optional parameter specifying the language and character set, which are separated by a comma. Typically, when specifying this parameter, you should use the value of the 'locale' parameter returned in the MailImmutableData struct. When this parameter is not specified, the default value used is 'en-US,Windows-1252'.
- `catsOriginal` - The identifiers of the categories originally associated with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.
- `catsSelected` - The identifiers of the categories to associate with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.
- `xheader` - An optional parameter containing an array of structures used to specify xheader names and values to include in the response. Each structure must have these parameters:
  - `xheaderName` - The xheaderName value must begin with 'X-', cannot have a ':' and cannot be 'X-mailer'.
  - `xheaderValue` - The value for the xheader.

Response Data

The msgSendKeepCurrent contains the message key of the outgoing message.

Parameters:

returnCode - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

messageKey - The unique identifier of the message.

cem.api.msgSendMarkForArchiving Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgSendMarkForArchiving command is used to send a response and archive the original message. Prior to using this command, you must open the message for response using the msgOpenForResponse, msgOpenByQueueAndKey, msgOpenRouted, or the msgOpenFromQueues command. The message opened for response is the message being replied to.

Note: You cannot retrieve messageKey values for archived messages. Therefore, to use the msgUnarchive command, you must save the messageKey value of the message you are marking for archiving.

Agent State and Mode Implications

If the agent's role requires that the agent wrap messages, the agent's state after executing this command becomes Wrap until the case is wrapped and the agent executes the msgExit command. Otherwise:

If the agent is configured to work with messages routed through ICM, the agent's state becomes idle.

If the agent is not configured to work with messages routed through ICM, the agent's state becomes working.

Message State Implications

The message's state must be openForResponse when the agent executes this command. After the agent executes this command, the message closes.

If the agent's role requires wrapping of messages, wrap time for the message is recorded by E-Mail Manager, although the message is closed.

Input Data

Input data for the `msgSendMarkForArchiving` command includes the following:

`mailResponse` - The struct containing all parameters with data about the outgoing message.

- `forceNewTrackingNumber` - An optional parameter specifying whether to have replies to the message use a new tracking number (1) or not (0). If not specified, the value '0' is assumed.
- `from` - The e-mail address the message is being sent from.
- `trackingNumber` - The tracking number of the thread in which the message being sent is a part.
- `subject` - The subject of the message.
- `reply` - The text of the draft reply.
- `to` - The address(es) to send the message to.
- `cc` - An optional parameter with the address(es) to copy the message to.
- `bcc` - An optional parameter with the address(es) to blind copy the message to.
- `attachments` - An optional parameter with the Ids of attachments to be sent with the message.
- `locale` - An optional parameter specifying the language and character set, which are separated by a comma. Typically, when specifying this parameter, you should use the value of the 'locale' parameter returned in the MailImmutableData struct. When this parameter is not specified, the default value used is 'en-US, Windows-1252'.
- `catsOriginal` - The identifiers of the categories originally associated with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.
- `catsSelected` - The identifiers of the categories to associate with the message. Each identifier must be specified within the array element. Mandatory in catsChangeForMsg command, optional in other commands.

- xheader - An optional parameter containing an array of structures used to specify xheader names and values to include in the response. Each structure must have these parameters:
  - xheaderName - The xheaderName value must begin with 'X-', cannot have a ':' and cannot be 'X-mailer'.
  - xheaderValue - The value for the xheader.

Response Data

The msgSendMarkForArchiving contains the message key of the outgoing message.

Parameters:

returnCode - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

messageKey - The unique identifier of the message.

cem.api.msgUnarchive Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The msgUnarchive command is used to unarchive messages.

Caution: Depending on the E-Mail Manager configuration, archived messages may be moved to the Lambda (Secondary)database, in which case you cannot unarchive them. If you use the msgUnarchive command for a message that has been moved, you receive a fault.

Note: You cannot retrieve messageKey values for archived messages. Therefore, to use the msgUnarchive command, you must save the messageKey value of the message before using the msgMarkForArchiving command.

Input Data

Input data for the `msgUnarchive` command includes the following:

`messageKey` - The unique identifier of the message.

Response Data

The msgUnarchive response contains mutable data about the message.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`MailMutableData` - The MailMutableData field is a struct that contains data, which may change over time, about a single message. A single response can contain multiple MailMutable fields, depending on the command executed and the number of messages for which data is returned.

- `MailMutableData.replied` - Whether the message has been replied to (1) or not (0).
- `MailMutableData.agentInRead` - Whether the message is being read by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.manualEscalateToQueue` - Whether the message was escalated to the current queue (1) or not (0).
- `MailMutableData.newToQueue` - Whether the message has never been read in the current queue (1) or has been (0).
- `MailMutableData.type` - The numerical indication of the type of message. Possible values are:
  - 0 - Messages from customers that have been closed and are waiting to be moved to the LAMBDA database.
  - 1 - Messages from customers that are currently still active.
  - 2 - Reply messages waiting to be sent by either a Send and Archive, or, Send and Keep Current operation.
  - 6 - Reply messages that are in a draft state.
  - 8 - Utility messages that have not yet been sent. These messages are only used internally and should not be utilized for any purpose.
  - 9 - Reply messages that have been sent successfully and remain in the primary database.
  - 10 - Reply messages that have not been sent successfully. E-Mail Manager has tried to send such messages through the SMTP server the number of retries configured, and is no longer trying.
  - 11 - Messages from agents to other agents that are currently active or to himself. msgCreateNewStub creates messages of this type.

- 12 - Messages from agents to other agents that are closed and are waiting to be moved to the LAMBDA database.
- 13 - Reply messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address.
- 14 - System messages used internally. These messages should not be utilized for any purposes.
- 15 - Outgoing messages to a mailing list.
- 16 - Outgoing messages to another instance of E-Mail Manager as a transfer.
- 17 - Outgoing messages that include To addresses that are internal to E-Mail Manager.
- 18 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 19 - Messages used to communicate workflow information between TServer and RServer. These messages are used internally and should not be utilized for any purpose.
- 22 - Reply messages that have been marked by the agent to be ignored due to a permanent SMTP error. This is one outcome from type 13.
- 23 - Messages from customers that have just been replied to but await the sending of the reply. Once the reply has been sent, the message type is changed to 0.
- 24 - Reply messages sent through MailTrack.
- 30 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 42 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
- 99 - Utility messages that have been sent out successfully. These messages are used internally and should not be utilized for any purpose.
- 100 - Utility messages that have not been sent successfully because they could not be sent to the SMTP server over the retry count. These messages are used internally and should not be utilized for any purpose.
- 103 - Utility messages that have not been sent successfully because they failed to be processed by the SMTP server due to a permanent error, such as a bad e-mail address. These messages are used internally and should not be utilized for any purpose.

  - 200 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 201 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 211 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.
  - 212 - Internal LAMBDA messages. These messages are used internally and should not be utilized for any purpose.

- `MailMutableData.pksSkillid` - The skill associated with the message.
- `MailMutableData.lastMoveS_id` - The unique identifier of the last agent to reassign the message.
- `MailMutableData.lastMover_name` - The login name of the last agent to move the message.
- `MailMutableData.lastMoveS_name` - The name of the last agent to reassign the message.
- `MailMutableData.repliedInQueue` - Whether the message has been replied to and kept current (1) or not (0).
- `MailMutableData.collab` - Whether the reassigned or escalated message has a draft response created by the last owner (1) or not (0).
- `MailMutableData.slMoved` - Whether the message was ever escalated due to service level management (1) or not (0).
- `MailMutableData.agentInRespond` - Whether the message is being responded to by an agent (1) or not (0). The value is 0 when the agent is reading an internal message.
- `MailMutableData.trackRefs` - Whether other messages is E-Mail Manager have the same tracking number (1) or not(0).
- `MailMutableData.autoresponse` - Whether the message received an AutoResponse through the rules (1) or not (0).
- `MailMutableData.outgoing` - Whether the message is an outgoing message (1) or not (0).
- `MailMutableData.slWarnEver` - Whether the message was ever in a warning state (1) or not (0).

- `MailMutableData.commentTag` - The most recent note added to the message, if any.
- `MailMutableData.draft` - Whether a draft response exists for the message (1) or not (0).
- `MailMutableData.overdue` - Whether the message is overdue (1) or not (0).
- `MailMutableData.warning` - Whether the message has reached the warning threshold (1) or not (0).
- `MailMutableData.thMovedToQueue` - Whether the message is in the current queue because of overload escalation (1) or not (0).
- `MailMutableData.thMoved` - Whether the message was ever escalated because of an overloaded queue (1) or not (0).
- `MailMutableData.fromRefs` - Whether there are other message is E-Mail Manager with the same From address (1) or not (0).
- `MailMutableData.pksOwnerid` - The unique identifier of the queue that holds the message.
- `MailMutableData.lastMover_id` - The identifier of the last agent who moved the message.
- `MailMutableData.manualEscalate` - Whether the message was escalated by an agent (1) or not (0).
- `MailMutableData.slOverdueEver` - Whether the message was ever in an overdue state (1) or not (0).
- `MailMutableData.pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the number, the higher the priority.
- `MailMutableData.slMovedToQueue` - Whether the message was escalated to the current queue due to service level management (1) or not (0).
- `MailMutableData.archived` - Whether the message is archived (1) or not (0).

cem.api.queueGetAgentList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetAgentList command retrieves data about all agent queues.

Input Data

No input data is required for the `queueGetAgentList` command.

Response Data

The queueGetAgentList response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`queueList` - The queueList field contains an array of structs, each containing data about a skill group or agent queue.

- `queueList[].name` - The name of the queue.

- `queueList[].isChoosable` - Whether the agent can choose the queue (1) or not (0).
- `queueList[].isIPTA` - Whether the queue is an ICM Routing skill group (1) or not (0).
- `queueList[].hasQueue` - Whether the queue has an actual queue that can store messages (1) or not (0).
- `queueList[].queueId` - The unique identifier for the queue.
- `queueList[].isReassignable` - Whether the agent can reassign messages to the queue (1) or not (0).
- `queueList[].isSkillgroup` - Whether the queue is a skill group (1) or not (0).
- `queueList[].isPermanent` - Whether the queue is a permanent queue (1) or not (0).
- `queueList[].description` - The description of the queue.
- `queueList[].alias` - If for an agent's queue, the agent's alias.

cem.api.queueGetMemberSkillgroupList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetMemberSkillgroupList command retrieves data about skill group queues in which the agent is a member.

Input Data

No input data is required for the `queueGetMemberSkillgroupList` command.

Response Data

The queueGetMemberSkillgroupList response contains the following data:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`queueList` - The queueList field contains an array of structs, each containing data about a skill group or agent queue.

- `queueList[].name` - The name of the queue.
- `queueList[].isChoosable` - Whether the agent can choose the queue (1) or not (0).
- `queueList[].isIPTA` - Whether the queue is an ICM Routing skill group (1) or not (0).
- `queueList[].hasQueue` - Whether the queue has an actual queue that can store messages (1) or not (0).
- `queueList[].queueId` - The unique identifier for the queue.
- `queueList[].isReassignable` - Whether the agent can reassign messages to the queue (1) or not (0).
- `queueList[].isSkillgroup` - Whether the queue is a skill group (1) or not (0).
- `queueList[].isPermanent` - Whether the queue is a permanent queue (1) or not (0).
- `queueList[].description` - The description of the queue.
- `queueList[].alias` - If for an agent's queue, the agent's alias.

cem.api.queueGetMsgInfo Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetMsgInfo command is used to retrieve information about messages in a queue.

Input Data

Input data for the `queueGetMsgInfo` command includes the following:

`queueId` - The unique identifier for the queue.

Response Data

The queueGetMsgInfo response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.
- 300 - Incorrect data type.

queueId - The unique identifier for the queue.

ListMailData - The ListMailData field contains an array of structs, each containing data about a message in the queue.

- ListMailData[].owner - The ID of the owner of the message.
- ListMailData[].fromEaten - The processed From address of the e-mail message.
- ListMailData[].MSubject - The Subject of the e-mail message.
- ListMailData[].commentTag - Notes attached to the message.
- ListMailData[].messageKey - The unique identifier of the message.
- ListMailData[].timeDone - The time the message was received in E-Mail Manager.
- ListMailData[].MReplyTo - The Reply To address of the e-mail message.
- ListMailData[].pksPriority - The numerical priority of the message, with 0 normal priority, and the higher the integer, the higher the priority.
- ListMailData[].trackingNumber - The tracking number of the thread of which the message is a part.

cem.api.queueGetPeerList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetPeerList command retrieves data about skill group queues the agent is a member of, as well as data about other agents in those queues.

Input Data

No input data is required for the `queueGetPeerList` command.

Response Data

The queueGetPeerList response contains the following data:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`queueList` - The queueList field contains an array of structs, each containing data about a skill group or agent queue.

- `queueList[].name` - The name of the queue.
- `queueList[].isChoosable` - Whether the agent can choose the queue (1) or not (0).
- `queueList[].isIPTA` - Whether the queue is an ICM Routing skill group (1) or not (0).
- `queueList[].hasQueue` - Whether the queue has an actual queue that can store messages (1) or not (0).
- `queueList[].queueId` - The unique identifier for the queue.
- `queueList[].isReassignable` - Whether the agent can reassign messages to the queue (1) or not (0).
- `queueList[].isSkillgroup` - Whether the queue is a skill group (1) or not (0).
- `queueList[].isPermanent` - Whether the queue is a permanent queue (1) or not (0).
- `queueList[].description` - The description of the queue.
- `queueList[].alias` - If for an agent's queue, the agent's alias.

cem.api.queueGetReassignableList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetReassignableList command retrieves data about queues to which the agent can reassign or escalate messages.

Input Data

No input data is required for the `queueGetReassignableList` command.

Response Data

The queueGetReassignableList response contains the following data:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`queueList` - The queueList field contains an array of structs, each containing data about a skill group or agent queue.

- `queueList[].name` - The name of the queue.
- `queueList[].isChoosable` - Whether the agent can choose the queue (1) or not (0).
- `queueList[].isIPTA` - Whether the queue is an ICM Routing skill group (1) or not (0).
- `queueList[].hasQueue` - Whether the queue has an actual queue that can store messages (1) or not (0).
- `queueList[].queueId` - The unique identifier for the queue.
- `queueList[].isReassignable` - Whether the agent can reassign messages to the queue (1) or not (0).
- `queueList[].isSkillgroup` - Whether the queue is a skill group (1) or not (0).
- `queueList[].isPermanent` - Whether the queue is a permanent queue (1) or not (0).
- `queueList[].description` - The description of the queue.
- `queueList[].alias` - If for an agent's queue, the agent's alias.

cem.api.queueGetSkillgroupList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetSkillgroupList command is used to retrieve a list of all skill groups.

Input Data

No input data is required for the `queueGetSkillgroupList` command.

Response Data

The queueGetSkillgroupList response contains data about all the skill groups in the E-Mail Manager instance.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`queueList` - The queueList field contains an array of structs, each containing data about a skill group or agent queue.

- queueList[].name - The name of the queue.
- queueList[].isChoosable - Whether the agent can choose the queue (1) or not (0).
- queueList[].isIPTA - Whether the queue is an ICM Routing skill group (1) or not (0).
- queueList[].hasQueue - Whether the queue has an actual queue that can store messages (1) or not (0).
- queueList[].queueId - The unique identifier for the queue.
- queueList[].isReassignable - Whether the agent can reassign messages to the queue (1) or not (0).
- queueList[].isSkillgroup - Whether the queue is a skill group (1) or not (0).
- queueList[].isPermanent - Whether the queue is a permanent queue (1) or not (0).
- queueList[].description - The description of the queue.
- queueList[].alias - If for an agent's queue, the agent's alias.

cem.api.queueGetStatistics Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The queueGetStatistics command is used to retrieve information about one or more queues.

Input Data

Input data for the `queueGetStatistics` command includes the following:

`type` - The indication of the queues for which you want to retrieve statistics: '0' for statistics of the queueIds specified in the queueList parameter; '1' for the queue of the agent issuing the command; '2' for all agent queues; '3' for the skill groups to which the agent belongs; and '4' for all skill groups.

`queueList` - An array of queueIds identifying the queues for which you want statistics. This parameter is only required when the value of the type parameter is '0'. Each queueId must be specified within the array element.

`responseFormat` - An optional indication of the amount of detail about the queue you want in the response. Possible values are 'queueStatisticsAbbreviated' for abbreviated data about the queue, or 'queueStatistics' for complete data. The default value used when this parameter is not specified is 'queueStatistics'.

Response Data

The queueGetStatistics response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`queueGetStatistics` - The queueGetStatistics field contains an array of structs, each containing data about a queue.

- `queueGetStatistics[].overdueCount` - The number of overdue messages in the queue.
- `queueGetStatistics[].queueName` - The name of the queue.
- `queueGetStatistics[].queueId` - The unique identifier for the queue.
- `queueGetStatistics[].queueType` - The type of queue: 0 for an agent queue or 2 for a skill group queue.
- `queueGetStatistics[].totalWaitTimeUnans` - The total waiting time for unanswered messages in the queue. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].newCount` - The number of new messages in the queue. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].dayOut` - The number of messages that have been removed from the queue on this day. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].longestTimeTotal` - The longest time a message has been waiting in the queue. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].averWaitTimeTotal` - The average time messages have been waiting in the queue. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].longestTimeUnans` - The longest time a message has been in the queue without being responded to. This parameter is not included in the abbreviated response.
- `queueGetStatistics[].preCount` - The number of current messages to which replies have been sent. This parameter is not included in the abbreviated response.

- queueGetStatistics[].overdueCount - The number of overdue messages in the queue. This parameter is not included in the abbreviated response.

- queueGetStatistics[].blankCount - The number of messages that have not been replied to and are not overdue. This parameter is not included in the abbreviated response.

- queueGetStatistics[].dayIn - The number of messages that entered the queue on this day. This parameter is not included in the abbreviated response.

- queueGetStatistics[].warningCount - The number of messages flagged as Overdue Warning. This parameter is not included in the abbreviated response.

- queueGetStatistics[].avgWaitTimeUnans - The average wait time of unanswered messages in the queue. This parameter is not included in the abbreviated response.

- queueGetStatistics[].totalUnans - The number of unanswered messages in the queue. This parameter is not included in the abbreviated response.

- queueGetStatistics[].actionCount - The number of active messages in the queue. This parameter is not included in the abbreviated response.

cem.api.actionHistoryGetForMessage Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The actionHistoryGetForMessage command is used to retrieve the action history for a message.

Input Data

Input data for the `actionHistoryGetForMessage` command includes the following:

`messageKey` - The unique identifier of the message.

`isOld` - An optional parameter indicating whether to retrieve messages that have been moved to the LAMBDA database (1) or not (0). The default value when not specified is 0.

`stringFormat` - An optional parameter that specifies how the text string is to be returned. Possible values are 0 (the default) to return the localized string using the language specified by the User Connection locale value; 1, to return the XML definition of the string; or 2, to return both the localized string and the XML definition of the string.

Response Data

The actionHistoryGetForMessage response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`messageActions` - The messageActions field contains an array of structs, each containing information about a specific message action.

- `messageActions[].pksRule` - If this action was logged by a rule, the name of the rule. Otherwise: BeforeRules if the action occurred before rule processing; FromRServer if RServer caused the action; EMSender if the action occurred when sending e-mail; Non-Rule if an error occurred during rule processing; or StatusManager if TServer caused the error.
- `messageActions[].pksAction` - The integer action code. See the database table PksAction_Ref for a list of codes.
- `messageActions[].userDefinedLocalized` - A string description of the action that occurred.
- `messageActions[].timeActed` - The time the action took place.
- `messageActions[].pksStatus` - The success code, usually 0 or 200. API users should use the value of the returnCode field to judge success.

cem.api.historyCheckOldForSender Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The historyCheckOldForSender command is used to determine if messages from a specific e-mail address have been moved to the LAMBDA database.

Input Data

Input data for the `historyCheckOldForSender` command includes the following:

`sender` - The e-mail address of the sender.

Response Data

The historyCheckOldForSender response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`oldExists` - Whether (true) or not (false) messages have been moved to the LAMBDA database.

cem.api.historyCheckOldForTrackingNumber Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The historyCheckOldForTrackingNumber command is used to determine if messages with a specific tracking number have been moved to the LAMBDA database.

Input Data

Input data for the `historyCheckOldForTrackingNumber` command includes the following:

`trackingNumber` - The unique tracking number for the thread.

Response Data

The historyCheckOldForTrackingNumber response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`oldExists` - Whether (true) or not (false) messages have been moved to the LAMBDA database.

cem.api.historyGetForSender Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The historyGetForSender command is used to retrieve all historical messages from a specific sender.

Input Data

Input data for the `historyGetForSender` command includes the following:

`sender` - The e-mail address of the sender.

`isOld` - An optional parameter indicating whether to retrieve messages that have been moved to the LAMBDA database (1) or not (0). The default value when not specified is 0.

Response Data

The historyGetForSender response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`ListMailData` - The ListMailData field contains an array of structs, each containing data about a message in the queue.

- `ListMailData[].owner` - The ID of the owner of the message.
- `ListMailData[].fromEaten` - The processed From address of the e-mail message.
- `ListMailData[].MSubject` - The Subject of the e-mail message.
- `ListMailData[].commentTag` - Notes attached to the message.
- `ListMailData[].messageKey` - The unique identifier of the message.
- `ListMailData[].timeDone` - The time the message was received in E-Mail Manager.
- `ListMailData[].MReplyTo` - The Reply To address of the e-mail message.
- `ListMailData[].pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the integer, the higher the priority.
- `ListMailData[].trackingNumber` - The tracking number of the thread of which the message is a part.

cem.api.historyGetForTrackingNumber Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The historyGetForTrackingNumber command is used to retrieve all historical messages with a specific tracking number.

Input Data

Input data for the `historyGetForTrackingNumber` command includes the following:

`trackingNumber` - The unique tracking number for the thread.

`isOld` - An optional parameter indicating whether to retrieve messages that have been moved to the LAMBDA database (1) or not (0). The default value when not specified is 0.

Response Data

The historyGetForTrackingNumber response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

`ListMailData` - The ListMailData field contains an array of structs, each containing data about a message in the queue.

- `ListMailData[].owner` - The ID of the owner of the message.
- `ListMailData[].fromEaten` - The processed From address of the e-mail message.
- `ListMailData[].MSubject` - The Subject of the e-mail message.
- `ListMailData[].commentTag` - Notes attached to the message.
- `ListMailData[].messageKey` - The unique identifier of the message.
- `ListMailData[].timeDone` - The time the message was received in E-Mail Manager.
- `ListMailData[].MReplyTo` - The Reply To address of the e-mail message.
- `ListMailData[].pksPriority` - The numerical priority of the message, with 0 normal priority, and the higher the integer, the higher the priority.
- `ListMailData[].trackingNumber` - The tracking number of the thread of which the message is a part.

cem.api.notesAddForMessage Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesAddForMessage command is used to add a note to a specific message

Input Data

Input data for the `notesAddForMessage` command includes the following:

`messageKey` - The unique identifier of the message.

`noteText` - The note to add to the message.

Response Data

The notesAddForMessage response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.notesAddForSender Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesAddForSender command is used to add a note to all messages sent from a specific e-mail address.

Input Data

Input data for the `notesAddForSender` command includes the following:

`sender` - The e-mail address of the sender to which to add the note.

`noteText` - The note to add to messages associated with the sender.

Response Data

The notesAddForSender response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.notesAddForTrackingNumber Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesAddForTrackingNumber command is used to add a note to all messages associated with a specific tracking number.

Input Data

Input data for the `notesAddForTrackingNumber` command includes the following:

`trackingNumber` - The unique tracking number for the thread.

`noteText` - The note to add to the tracking number.

Response Data

The notesAddForTrackingNumber response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.notesGetForMessage Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesGetForMessage command retrieves all notes associated with a specific message.

Input Data

Input data for the `notesGetForMessage` command includes the following:

`messageKey` - The unique identifier of the message.

`isOld` - An optional parameter indicating whether to retrieve messages that have been moved to the LAMBDA database (1) or not (0). The default value when not specified is 0.

`stringFormat` - An optional parameter that specifies how the text string is to be returned. Possible values are 0 (the default) to return the localized string using the language specified by the User Connection locale value; 1, to return the XML definition of the string; or 2, to return both the localized string and the XML definition of the string.

Response Data

The notesGetForMessage response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`Notes` - The Notes field contains an array of structs, each containing information about a note attached to a message, tracking number, or sender.

- `Notes[].timeStamp` - The time the stamp was created.
- `Notes[].commentLocalized` - The localized text of the note. This field is not returned if the localized text was not requested.
- `Notes[].commentObject` - The XML definition of the string. This field is not returned if the XML definition of the string was not requested.
- `Notes[].pksUser` - The ID of the user who added the note.

cem.api.notesGetForSender Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesGetForSender command is used to retrieve all notes associated with a specific sender.

Input Data

Input data for the `notesGetForSender` command includes the following:

`sender` - The e-mail address of the sender.

`stringFormat` - An optional parameter that specifies how the text string is to be returned. Possible values are 0 (the default) to return the localized string using the language specified by the User Connection locale value; 1, to return the XML definition of the string; or 2, to return both the localized string and the XML definition of the string.

Response Data

The notesGetForSender response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`Notes` - The Notes field contains an array of structs, each containing information about a note attached to a message, tracking number, or sender.

- `Notes[].timeStamp` - The time the stamp was created.
- `Notes[].commentLocalized` - The localized text of the note. This field is not returned if the localized text was not requested.
- `Notes[].commentObject` - The XML definition of the string. This field is not returned if the XML definition of the string was not requested.
- `Notes[].pksUser` - The ID of the user who added the note.

cem.api.notesGetForTrackingNumber Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The notesGetForTrackingNumber command is used to retrieve all notes associated with a specific tracking number.

Input Data

Input data for the `notesGetForTrackingNumber` command includes the following:

`trackingNumber` - The unique tracking number for the thread.

`stringFormat` - An optional parameter that specifies how the text string is to be returned. Possible values are 0 (the default) to return the localized string using the language specified by the User Connection locale value; 1, to return the XML definition of the string; or 2, to return both the localized string and the XML definition of the string.

Response Data

The notesGetForTrackingNumber response contains the following:

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`Notes` - The Notes field contains an array of structs, each containing information about a note attached to a message, tracking number, or sender.

- `Notes[].timeStamp` - The time the stamp was created.
- `Notes[].commentLocalized` - The localized text of the note. This field is not returned if the localized text was not requested.
- `Notes[].commentObject` - The XML definition of the string. This field is not returned if the XML definition of the string was not requested.
- `Notes[].pksUser` - The ID of the user who added the note.

cem.api.templCreate Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templCreate command is used to create a new template.

Input Data

Input data for the `templCreate` command includes the following:

`templateAttributes` - A struct containing information about the template.

- `templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.
- `name` - The name of the template. This parameter is ignored for the templDelete command.
- `description` - The description of the template. This parameter is ignored for the templDelete command.
- `editText` - The text of the template. The following characters require special XML formatting in the template text: & < > " and '. Use an ampersand, followed by the abbreviation for the character (amp, lt, gt, quot, and apos), followed by a semi-colon. This parameter is ignored for the templDelete command.
- `library` - The name of the library containing the template.
- `owner` - The unique identifier of the owner of the template. If the value of the library parameter is 'autoresponse', the owner parameter is not required and is ignored.

- `keywords` - An optional parameter containing an array of keywords to associate with the template. This parameter is ignored for the templDelete command.

- `attachments` - An optional parameter containing an array of the unique identifiers of attachments to associate with the template.

Response Data

The templCreate response contains the template key of the template created.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- `0` - Command Succeeded.

`templateKey` - The unique identifier of the template.

cem.api.templDelete Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templDelete command is used to delete an existing template.

Input Data

Input data for the `templDelete` command includes the following:

`templateAttributes` - A struct containing information about the template.

- `templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.
- `name` - The name of the template. This parameter is ignored for the templDelete command.
- `description` - The description of the template. This parameter is ignored for the templDelete command.
- `editText` - The text of the template. The following characters require special XML formatting in the template text: & < > " and '. Use an ampersand, followed by the abbreviation for the character (amp, lt, gt, quot, and apos), followed by a semi-colon. This parameter is ignored for the templDelete command.
- `library` - The name of the library containing the template.
- `owner` - The unique identifier of the owner of the template. If the value of the library parameter is 'autoresponse', the owner parameter is not required and is ignored.

- `keywords` - An optional parameter containing an array of keywords to associate with the template. This parameter is ignored for the templDelete command.

- `attachments` - An optional parameter containing an array of the unique identifiers of attachments to associate with the template.

Response Data

The templDelete response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

cem.api.templGetAllKeywords Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetAllKeywords command is used to retrieve the list of all template keywords in the E-Mail Manager instance.

Input Data

No input data is required for the `templGetAllKeywords` command.

Response Data

The templGetAllKeywords response contains data about the keywords in the E-Mail Manager instance.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`keywordList` - The keywordList field contains an array of keywords.

cem.api.templGetAttachments Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetAttachments command is used to retrieve the list of attachments for a specific template.

Input Data

Input data for the `templGetAttachments` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

Response Data

The templGetAttachments response contains data about the attachments associated with the specified template.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`attachmentList` - The attachmentList field contains an array of structs, each containing information about an attachment associated with the specific template.

- `attachmentList[].attDesc` - The description of the template.
- `attachmentList[].attachmentId` - The unique identifier for the attachment.
- `attachmentList[].originalFileName` - The name of the attachment file.

cem.api.templGetDefaultDynamicText Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetDefaultDynamicText command is used to get the default text for tokens in a dynamic template.

Input Data

Input data for the `templGetDefaultDynamicText` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

Response Data

The templGetDefaultDynamicText response contains the dynamic template text with the default token values inserted.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`defaultText` - The defaultText field contains the text of a dynamic template with the default token values inserted.

cem.api.templGetDynamicText Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetDynamicText command is used to get the text for a dynamic template with the custom data inserted for the tokens.

Input Data

Input data for the `templGetDynamicText` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

`to` - The value of the 'to' header of the message.

`from` - The value of the 'from' header of the message.

`replyTo` - The value of the 'replyTo' header of the message.

`subject` - The value of the subject of the message.

`date` - The value of the 'date' header of the message.

`trackingNumber` - The tracking number of the message.

Response Data

The templGetDynamicText response contains the dynamic template text with the custom data inserted for the tokens.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`dynamicText` - The dynamicText field contains the text of a dynamic template with the custom data inserted for the tokens.

cem.api.templGetDynamicTokenInfo Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetDynamicTokenInfo command is used to retrieve a list of dynamic tokens configured for use in the E-Mail Manager instance.

Input Data

No input data is required for the `templGetDynamicTokenInfo` command.

Response Data

The templGetDynamicTokenInfo response contains data about the dynamic tokens configured for the E-Mail Manager instance.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`tokenList` - The tokenList field contains an array of structs, each containing information about a dynamic token.

- `tokenList[].tokenName` - The name of the token.
- `tokenList[].tokenHelp` - The help text describing the token.

cem.api.templGetEditFields Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetEditFields command is used to retrieve the text of a template that you intend to modify using the templReplace command.

Input Data

Input data for the `templGetEditFields` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

Response Data

The templGetEditFields response contains data about the template.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`editText` - The editText field contains the text of the specified template.

cem.api.templGetKeywords Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetKeywords command is used to retrieve the list of keywords associated with a specific template.

Input Data

Input data for the `templGetKeywords` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

Response Data

The templGetKeywords response contains data about the keywords associated with the specified template.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`keywordList` - The keywordList field contains an array of keywords.

cem.api.templGetList Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetList command is used to retrieve a list of templates. You can use this command to retrieve a list of personal, public, or suggested templates, or all templates in a specific library.

Input Data

Input data for the `templGetList` command includes the following:

`type` - The type of templates you want to retrieve. Possible values are: 'personal' for templates in the agent's personal library; 'public' for all public templates; 'suggested' for templates suggested for the message by the rules; or 'library' for templates from a specific library. When the value 'library' is used, the additional parameter libraryName must be included. When the value 'suggested' is used, the additional parameter messageKey must be included.

`libaryName` - The name of the library from which to retrieve templates. This parameter is only used when the value of the type parameter is 'library'.

`messageKey` - The unique identifier of the message for which to retrieve suggested templates. This parameter is only used when the value of the type parameter is 'suggested'.

Response Data

The templGetList response contains data about the requested templates.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`templateList` - The templateList field contains an array of structs, each containing information about a specific template.

- `templateList[].isSmart` - Whether the template is dynamic (1) or not (0).
- `templateList[].library` - The name of the library that contains the template.
- `templateList[].keyTag` - A comma-separated list of keywords associated with the template.
- `templateList[].pksOwnerid` - The unique identifier for the owner of the template.
- `templateList[].key` - The unique identifier of the template.
- `templateList[].description` - The description of the template.
- `templateList[].title` - The title of the template.

cem.api.templGetPublicLibraries Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetPulbicLibraries command is used to retrieve the list of all public template libraries in the E-Mail Manager instance.

Input Data

No input data is required for the `templGetPublicLibraries` command.

Response Data

The templGetPulbicLibraries response contains data about the public libraries in the E-Mail Manager instance.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`libraryList` - The libraryList field contains an array of library names.

cem.api.templGetText Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templGetText command is used to retrieve the text of a specific non-dynamic template for use in a response.

Input Data

Input data for the `templGetText` command includes the following:

`templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.

Response Data

The templGetText response contains the text of the specified template.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 100 - No data found.

`templateText` - The templateText field contains the text of the specified template.

cem.api.templReplace Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The templReplace command is used to modify an existing template.

Input Data

Input data for the `templReplace` command includes the following:

`templateAttributes` - A struct containing information about the template.

- `templateKey` - The unique identifier of the template. This parameter is ignored with the templCreate command. You retrieve templateKeys using the templGetList command.
- `name` - The name of the template. This parameter is ignored for the templDelete command.
- `description` - The description of the template. This parameter is ignored for the templDelete command.
- `editText` - The text of the template. The following characters require special XML formatting in the template text: & < > " and '. Use an ampersand, followed by the abbreviation for the character (amp, lt, gt, quot, and apos), followed by a semi-colon. This parameter is ignored for the templDelete command.
- `library` - The name of the library containing the template.
- `owner` - The unique identifier of the owner of the template. If the value of the library parameter is 'autoresponse', the owner parameter is not required and is ignored.

- `keywords` - An optional parameter containing an array of keywords to associate with the template. This parameter is ignored for the templDelete command.

- `attachments` - An optional parameter containing an array of the unique identifiers of attachments to associate with the template.

Response Data

The templReplace response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

Events

Overview of Events

This topic contains the following sections:

Before reading this topic and proceeding with events, you should read the following topics:

API DTD

The <APIEnvelope> Element

Contents of the <APIEnvelope>

Overview of Connections, and HTTP or Socket connections

Designing Event Notification

Registering and Unregistering for Events

In order to be notified of events, a client application must execute the `eventRegister` command for the type of event to be received, specifying the queues and messages for which you want to receive events. You must issue the eventRegister command for each type of event separately. When registering for events, you specify the messages or queues for which you want to receive events.

The client application then receives notification of that type of event, until it executes the `eventUnregister` command for the specific event it registered for, identified by the `eventRegistrationId`. If not `eventRegistrationId` is specified, the command unregisters the user for all events previously registered for.

Types of Events

There are four types of events. You indicate the type of event to receive with the `eventType` parameter in the `eventRegister` command. The `eventType` parameter is also part of the event structure sent to the application.

The eventType parameter values and descriptions are listed in the following table:

Value of eventType Parameter	Description
1	Overdue Message
2	A message was added to the queue. Specifically, one of the following: A message was reassigned to the queue. A message was received by the queue. A message was unarchived and placed in the queue.
3	The Queue became overloaded.
4	The agent became not routable.

Contents of Events

An event contains:

A <Event> element, with the following attributes:

- version, which always equals "1.0".
- name, which equals the name of the event.
- clientID, which identifies the event sent to the client application. The clientID value starts at 0 and increments by one for each event sent, so that the client application can test for missed events. The eventID attribute is unique for the sessID value.

The <Param> named eventType, which contains a <Value> element with an integer indicating the type of event.

One or more other <Param> elements, which contain data about the event.

For events of type 1, 3, and 4, you can detect the specific event by testing the value of the eventType parameter. For events of type 2, you must also test either the name attribute of the <Event> element or the action parameter in order to discern the exact event type.

Events as XML

Events are sent as XML in the following format:

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9"
lastAckedEventId="integer_value">
  <Event version="1.0" name="name_of_event"
clientId="integer_value">
    //Multiple <Param> elements with event data
  </Event>
</APIEnvelope>
```

Following is an example of Event Type 4:

```
<APIEnvelope lastAckedEventId="-1" sessId="5hbmzsnnr9" type="event">
  <Event clientId="2" name="agentNotRoutable" version="1.0">
    <Param name="eventType">
      <Value>4</Value>
    </Param>
    <Param name="agentId">
      <Value>ecf21a1062de11d59e01000000000000</Value>
    </Param>
  </Event>
</APIEnvelope>
```

Event Documentation File

API events are documented in the apiserver-commands.xml that is installed with the UI/API Server in the following location:

installation-folder\uiroot\WEB-INF\properties\default\cem

In this file, API events are documented in the CMDNAMESPACE element named "event". Each event definition is enclosed in the EVENT element and includes:

A description.

Fields in the event.

Note: The format of the documentation file may be changed in later releases.

XML Event Examples

This topic contains the following sections:

Overview

This topic contains XML examples of the E-Mail Manager API events.

msgOverdue Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-
1">
  <Event version="1.0" name="msgOverdue" clientId="1">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="messageKey">
      <Value>000000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
      <Value>1</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_reassigned Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-
1">
  <Event version="1.0" name="messageOnQueue_reassigned"
clientId="3">
    <Param name="automaticReassign">
      <Value>1</Value>
    </Param>
    <Param name="doneBy">
      <Value>1</Value>
    </Param>
    <Param name="eventType">
      <Value>2</Value>
    </Param>
    <Param name="action">
      <Value>3</Value>
    </Param>
    <Param name="sourceQueueType">
      <Value>1</Value>
    </Param>
    <Param name="sourceQueueId">
      <Value>dd8203216f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param
name="messageKey"><Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
    <Param name="actor">
      <Value>00000000000000000000000000000000</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_received Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-
1">
  <Event version="1.0" name="messageOnQueue_received" clientId="4">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="messageKey">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
```

```
      <Value>2</Value>
    </Param>
    <Param name="action">
      <Value>1</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

messageOnQueue_unarchived Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-
1">
  <Event version="1.0" name="messageOnQueue_unarchived"
clientId="5">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="sendStatus">
      <Value>1</Value>
    </Param>
    <Param name="messageKey">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="eventType">
      <Value>2</Value>
    </Param>
    <Param name="actor">
      <Value>00000000000000000000000000000000</Value>
    </Param>
    <Param name="action">
      <Value>2</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

queueOverload Event Example

```
<APIEnvelope  type="event" sessId="5hbmzsnnr9" lastAckedEventId="-
1">
  <Event version="1.0" name="queueOverload" clientId="2">
    <Param name="queueId">
      <Value>dd8203206f3611d5907d00d0b7849ef8</Value>
    </Param>
    <Param name="eventType">
      <Value>3</Value>
    </Param>
    <Param name="queueType">
      <Value>1</Value>
    </Param>
  </Event>
</APIEnvelope>
```

agentNotRoutable Event Example

```
<APIEnvelope lastAckedEventId="-1" sessId="5hbmzsnnr9" type="event">
  <Event clientId="2" name="agentNotRoutable" version="1.0">
    <Param name="eventType">
      <Value>4</Value>
    </Param>
    <Param name="agentId">
      <Value>ecf21a1062de11d59e01000000000000</Value>
    </Param>
  </Event>
</APIEnvelope>
```

cem.event.msgOverdue Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

Event Type 1 indicates that a message has become overdue.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Parameters in Event

Event `msgOverdue` contains the following parameters:

`queueId` - The unique identifier for the queue.

`messageKey` - The unique identifier of the message.

`eventType` - The type of event.

`queueType` - The type of queue: (1) for an agent's personal queue, or (2) for a skill group queue.

cem.event.messageOnQueue_reassigned Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

EventType 2 - messageOnQueue_reassigned indicates that a message has been reassigned or escalated to a queue.

Parameters in Event

Event `messageOnQueue_reassigned` contains the following parameters:

`automaticReassign` - How the message was reassigned to the queue: (2) if it was reassigned through overdue escalation; (3) if it was reassigned through overload escalation; (1) if it was reassigned through any other mechanism. If the value of the parameter doneBy is (1), the value of this parameter is (2) or (3).

`doneBy` - How the message was reassigned to the queue: (1) if the message was reassigned by E-Mail Manager; (2) if the message was reassigned by ICM software in an integrated environment; or (3) if the message was reassigned by an agent. If the value is (1), the value of the parameter automaticReassign specifies how the message was automatically reassigned.

`eventType` - The type of event.

`action` - The code for the how the message reached the queue. For this event, the value is always "3".

`sourceQueueType` - The type of queue the message was reassigned from: (1) for an agent's personal queue, or (2) for a skillgroup queue.

`sourceQueueId` - The global unique identifier (GUID) for the queue from which the message was reassigned.

`queueId` - The unique identifier for the queue.

`messageKey` - The unique identifier of the message.

`queueType` - The type of queue: (1) for an agent's personal queue, or (2) for a skill group queue.

`actor` - The global unique identifier (GUID) for the agent who reassigned the message. If the message was not reassigned by an agent, this parameter is not included.

cem.event.messageOnQueue_received Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

Event Type 2 - messageOnQueue_received indicates that a message has been received in a queue.

Parameters in Event

Event `messageOnQueue_received` contains the following parameters:

`queueId` - The unique identifier for the queue.

`messageKey` - The unique identifier of the message.

`eventType` - The type of event.

`action` - The code for the how the message reached the queue. For this event, the value is always "2".

`queueType` - The type of queue: (1) for an agent's personal queue, or (2) for a skill group queue.

cem.event.messageOnQueue_unarchived Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

Event Type 2 - messageOnQueue_unarchived indicates that a message has been unarchived and placed in a queue.

Parameters in Event

Event `messageOnQueue_unarchived` contains the following parameters:

`queueId` - The unique identifier for the queue.

`sendStatus` - Whether the message was sent (1), or not (2).

`messageKey` - The unique identifier of the message.

`eventType` - The type of event.

`actor` - The global unique identifier (GUID) for the agent who unarchived the message.

`action` - The code for the how the message reached the queue. For this event, the value is always "2".

`queueType` - The type of queue: (1) for an agent's personal queue, or (2) for a skill group queue.

cem.event.queueOverload Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

Event Type 3 - queueOverload indicates that a queue has become overloaded.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Parameters in Event

Event `queueOverload` contains the following parameters:

`queueId` - The unique identifier for the queue.

`eventType` - The type of event.

`queueType` - The type of queue: (1) for an agent's personal queue, or (2) for a skill group queue.

cem.event.agentNotRoutable Event

This topic contains the following sections:

You should read or review the topic Overview of Events before reading this topic.

You can also view an XML example of this event.

Overview

Event Type 4 - agentNotRoutable indicates that an agent's mode has become not routable. In an integrated environment, this agent cannot be assigned messages through ICM software.

For more information, see the *Cisco E-Mail Manager Administration Guide.*

Parameters in Event

Event `agentNotRoutable` contains the following parameters:

`eventType` - The type of event.

`agentId` - The unique identifier for the agent.

cem.api.eventRegister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The eventRegister command is used to register to receive a specific type of event. To no longer receive the event, use the eventUnregister command. The input data may include queueScope or msgScope parameters, but not both.

Caution: The client application design should consider that the API Server does not know when an application through which a user has registered for an event terminates unexpectedly.

Input Data

Input data for the `eventRegister` command includes the following:

`eventType` - The type of event for which you are registering. Use 1, 2, 3, or 4.

queueScope - One or more queueGUIDs for which to register for events. The queue IDs are contained as values in an array. Each queueGUID must be specified within the array element. Instead of queue IDs, you can use the value all, which registers for the event for all queues. The event type '1' must have a queueScope parameter. For the event type '4', the queue's listed are the personal queues of the agent's to monitor.

`msgScope` - One or more keys for messages for which to register for events. The message keys are contained as values in an array. Each message key must be specified within the array element. Instead of message IDs, you can use the value all, which registers for the event for all messages.

Response Data

The eventRegister response provides the eventRegistrationId.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.
- 101 - The registration request partially overlapped with an existing registration (by the same application); the new registration request(s) were successful.

`eventRegistrationId` - The unique identifier for event registration. The eventRegistrationId must be used as a parameter in the eventUnregister command, to stop receiving the type of event.

cem.api.eventUnregister Command

This topic contains the following sections:

You should read or review the topics Overview of Commands and Responses and Overview of Faults before reading this topic.

You can also view an XML example of this command.

Overview

The eventUnregister command is used to stop receiving a specific type of event.

Input Data

Input data for the `eventUnregister` command includes the following:

`eventIds` - An array of eventRegistrationIds for the events types that you want to stop receiving. If no `eventIds` are specified, the command unregisters all outstanding event registrations.

Response Data

The eventUnregister response contains the return code.

Parameters:

`returnCode` - The returnCode field contains an integer value representing the different possible results of the command. If the command does not return one of the following values, a fault is returned instead.

Possible values are:

- 0 - Command Succeeded.

State Issues

Agent State

This topic contains the following sections:

Overview of Agent States

The agent state indicates what the agent is currently doing. The agent state can signify a specific task related to messages, or indicate that the agent is not currently engaged in a message-related task.

Following are the possible states an agent can be in:

`loggedoff`

`working`

`notready - <subreason>`

`idle`

`read`

`responding`

`wrap`

Receiving the Agent State

A client application receives information about an agent's state in the agentState parameter. The agentState parameters always returned with the agentMode parameter, as part of the response to the following commands:

agentConnect

agentDisconnectDuplicateandConnect

agentGetState

agentMakeReady

agentMakeRoutable

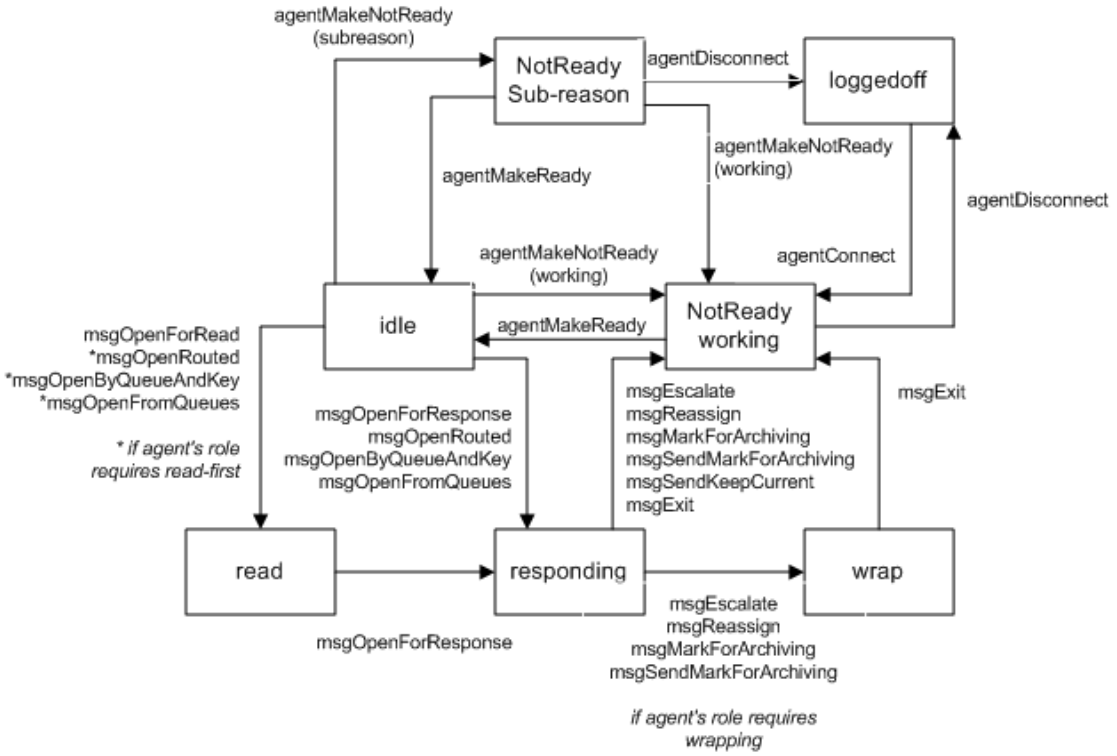
agentMakeNotReady

agentMakeNotRoutable

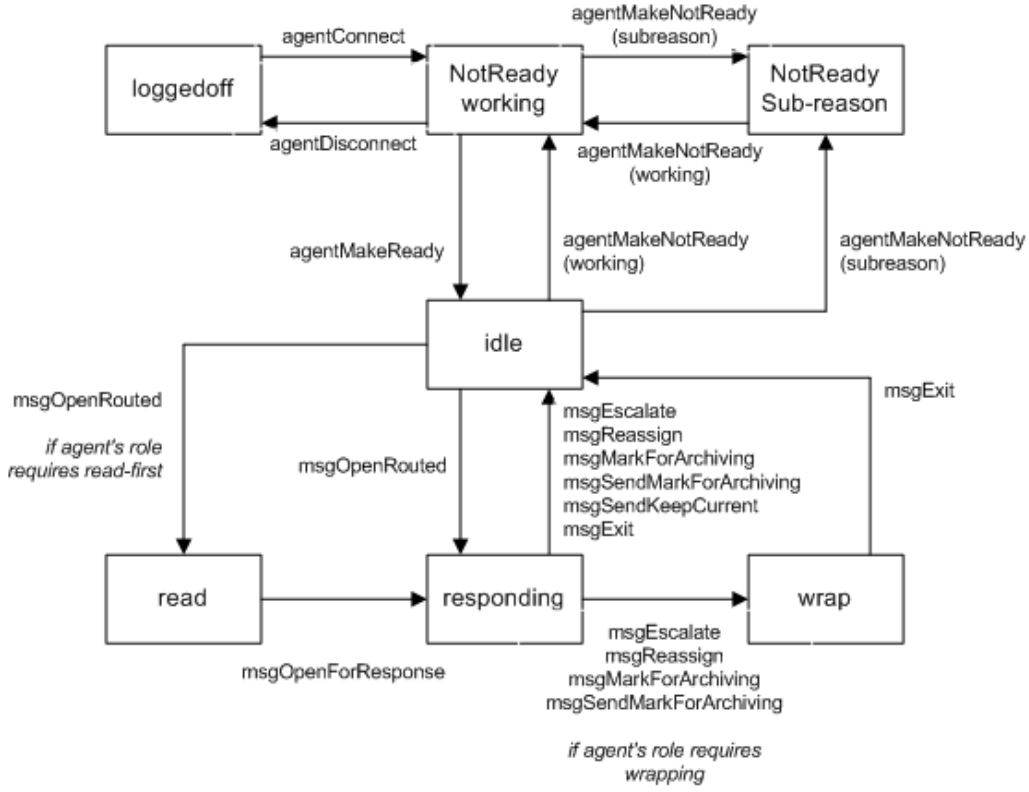
Agent State Transition Diagrams

Agent state transitions for certain commands are different depending on the agent mode, as shown in the following two diagrams.

The following diagram shows how the agent state can transition when the agent is in nonpush mode:



The following diagram shows how the agent state can transition, when the agent is in push mode:



Possible Values in agentState Response

The following table shows the possible name values and corresponding code values in the agentState response:

Name	Code
loggedoff	0
working	1
idle	2
read	3
responding	4
wrap	5
notready <subreason>	6

loggedoff

An agent is in the loggedoff state when the agent does not have an active session between the client application and E-Mail Manager.

The agent's state becomes loggedoff when the agent successfully executes the agentDisconnect command.

The agent's state ceases to be loggedoff when the agent successfully executes the agentConnect command.

working

This section contains the following subsections:

Overview

Commands that Always Transition to working State

Commands that Conditionally Transition to working State

Overview

An agent is in the working state when the agent is not processing messages.

Commands that Always Transition to working State

The agent's state becomes working after the agent successfully executes:

agentConnect

agentDisconnectDuplicateandConnect

agentMakeNotReady, with a reason code of '1', or with no specified reason

Commands that Conditionally Transition to working State

If the agent is not configured to work with messages routed through ICM, the agent's state becomes working after the agent successfully executes the msgExit or msgSendKeepCurrent command.

If the agent's role does not require that the agent wrap messages, the agent's state becomes working after the agent successfully executes:

msgEscalate

msgMarkForArchiving

msgReassign

msgSendMarkForArchiving

notready - <subreason>

Cisco E-Mail Manager can be customized to support reasons agents are not ready other than 'working'. If the instance of E-Mail Manager being used supports customized reasons:

The additional not ready reason codes are returned by the agentGetNotReadyReasons command.

A customer code can be used in the agentMakeNotReady command.

idle

This section contains the following subsections:

Overview

An agent is in the `idle` state when the agent is available to work on a message, but is not currently working on a message.

Commands that Always Transition to idle State

The agent's state becomes `idle` after the agent successfully executes the `agentMakeReady` command.

Commands that Conditionally Transition to idle State

If the agent is configured to work with messages routed through ICM, the agent's state becomes `idle` after the agent successfully executes the `msgExit` or `msgSendKeepCurrent` command.

If the agent's role does not require that the agent wrap messages, and agent is configured to work with messages routed through ICM, the agent's state becomes `idle` after the agent successfully executes:

`msgEscalate`

`msgMarkForArchiving`

`msgReassign`

`msgSendMarkForArchiving`

Otherwise, the agent state becomes `Wrap`.

read

An agent is in the read state when the agent has opened a message for reading. When an agent's state is read, the agent has a message in the openForRead message state.

The agent's state becomes read when:

The agent successfully executes the msgOpenForRead command.

An agent based on a role that requires that the agent read messages before responding successfully executes the msgOpenByQueueAndKey, msgOpenFromQueues, or msgOpenRouted command.

responding

An agent is in the responding state when the agent has opened a message for responding. When an agent's state is responding, the agent has a message in the openForResponse message state.

The agent's state becomes responding when:

And agent in the state read, as required by the agent's role, successfully executes the msgOpenForResponse command.

An agent based on a role that does not require that the agent read messages before responding successfully executes the msgOpenByQueueAndKey, msgOpenFromQueues, or msgOpenRouted command.

wrap

An agent is in the wrap state when the agent is wrapping a message. When an agent's state is Wrap, the message state of the message the agent was working on becomes closed.

If an agent's role requires cases to be wrapped, the agent's state becomes Wrap after the agent executes the following commands:

msgEsclate

msgMarkForArchiving

msgReassign

msgSendMarkForArchiving

Agent Mode

This topic contains the following sections:

Overview of Agent Modes

The agent mode specifies whether or not agents are currently eligible to be pushed messages routed by ICM software to be processed, or whether agents must choose specific messages routed through E-Mail Manager.

For more information on how agents can work with messages in push, pull, or pick mode, see either the *Cisco E-Mail Manager Agent Guide* or the *Cisco E-Mail Manager Administration Guide*.

Following are the possible modes an agent can be in:

`loggedoff`

`nonpush`

`push`

Receiving the Agent Mode

A client application receives information about an agent's mode in the `agentMode` parameter. The `agentMode` parameter is always returned with the `agentState` parameter, as part of the response to the following commands:

`agentConnect`

`agentDisconnectDuplicateandConnect`

`agentGetState`

agentMakeReady

agentMakeRoutable

agentMakeNotReady

agentMakeNotRoutable

Possible Values in agentMode Response

The following table shows the possible name values and corresponding code values in the agentMode response:

Name	Code
loggedoff	0
nonpush	1
push	2

loggedoff

An agent is in the loggedoff mode when the agent does not have an active session between the client application and E-Mail Manager.

The agent's mode becomes loggedoff when the agent successfully executes the agentDisconnect command.

The agent's mode ceases to be loggedoff when the agent successfully executes the agentConnect command.

nonpush

An agent is in nonpush mode when the agent is not eligible to receive messages through push routing. An agent in nonpush mode must use one of the following commands, instead of the msgOpenRouted command, to open messages:

msgOpenByQueueAndKey

msgOpenFromQueues

msgOpenForRead

msgOpenForResponse

The agent's mode becomes nonpush after the agent successfully executes the agentMakeNotRoutable command.

push

An agent is in the push mode when the agent is eligible to receive messages through push routing.

Agents in push mode who are configured to receive messages routed through ICM software use the msgOpenRouted command. For more information about integrated environments and the routing of messages through ICM software, see the *Cisco E-Mail Manager Administration Guide*.

The agent's mode becomes push after the agent successfully executes the agentMakeRoutable command.

Message State

This topic contains the following sections:

Overview

The message state indicates if and how an agent is currently working with a message.

A message can be in one, and only one, of the following states:

`agentInRead`

`agentInResponse`

`closed`

Receiving the Message State

A client application receives information about a message's state in the `MailMutableData` parameter, which is part of the response to the following commands:

`msgOpenByQueueAndKey`

`msgOpenFromQueues`

`msgGetStatus`

`msgOpenForRead`

```
msgOpenForResponse

msgClaim

msgReassign

msgEscalate

msgOpenRouted

msgUnarchive
```

The `MailMutableData` struct contains two parameters, `agentInRead` and `agentInResponse`, that let the user know his state relative to the message.

Message State and Wrapping

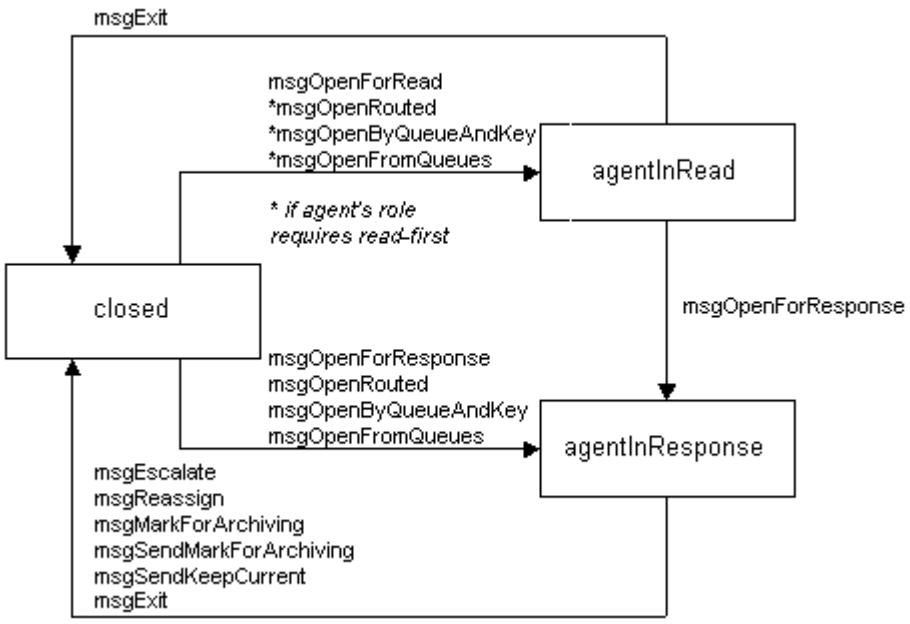
If an agent's role requires cases to be wrapped, the agent's state becomes Wrap after the agent executes the following commands:

```
msgEsclate

msgMarkForArchiving

msgReassign

msgSendKeepCurrent

msgSendMarkForArchiving
```

When the agent state becomes Wrap, the message's state becomes closed; there is no corresponding wrap state for messages. Therefore, a different agent can work with a message while the original agent is wrapping it. However, E-Mail Manager tracks the time the original agent spends wrapping the message.

Message State Transition Diagram

The following diagram shows how the agent mode can transition, in relation to specific commands:



agentInRead

A message is in the state `agentInRead` when an agent has opened a message for reading. The `agentInRead` state is indicated by the value "1" for the `agentInRead` parameter in the `MailMutableData` response.

An agent can specifically open a message for reading with the `msgOpenForRead` command.

If an agent based on a role that requires that the agent read messages before responding successfully executes the following commands, the message is in the `agentInRead` state:

`msgOpenByQueueAndKey`

`msgOpenFromQueues`

`msgOpenRouted`

agentInResponse

A message is in the state `agentInResponse` when an agent has opened a message for responding. The `agentInResponse` state is indicated by the value "1" for the `agentInResponse` parameter in the `MailMutableData` response.

An agent can specifically open a message for responding with the msgOpenForResponse command.

If an agent based on a role that does not require that the agent read messages before responding successfully executes the following commands, the message is in the agentInResponse state:

msgOpenByQueueAndKey

msgOpenFromQueues

msgOpenRouted

closed

The message state closed is not positively indicated in the MailMutableData response. Rather, a message is closed when its state is neither openForRead or openForResponse. The value of both the openForRead or openForResponse parameters in the MailMutableData response is "0".

Appendices

Testing the API

This topic contains the following sections:

Overview

While you are developing applications to use the E-Mail Manager API, you most likely will want to test the commands you send to the API Server. You may develop your own test tools to do this.

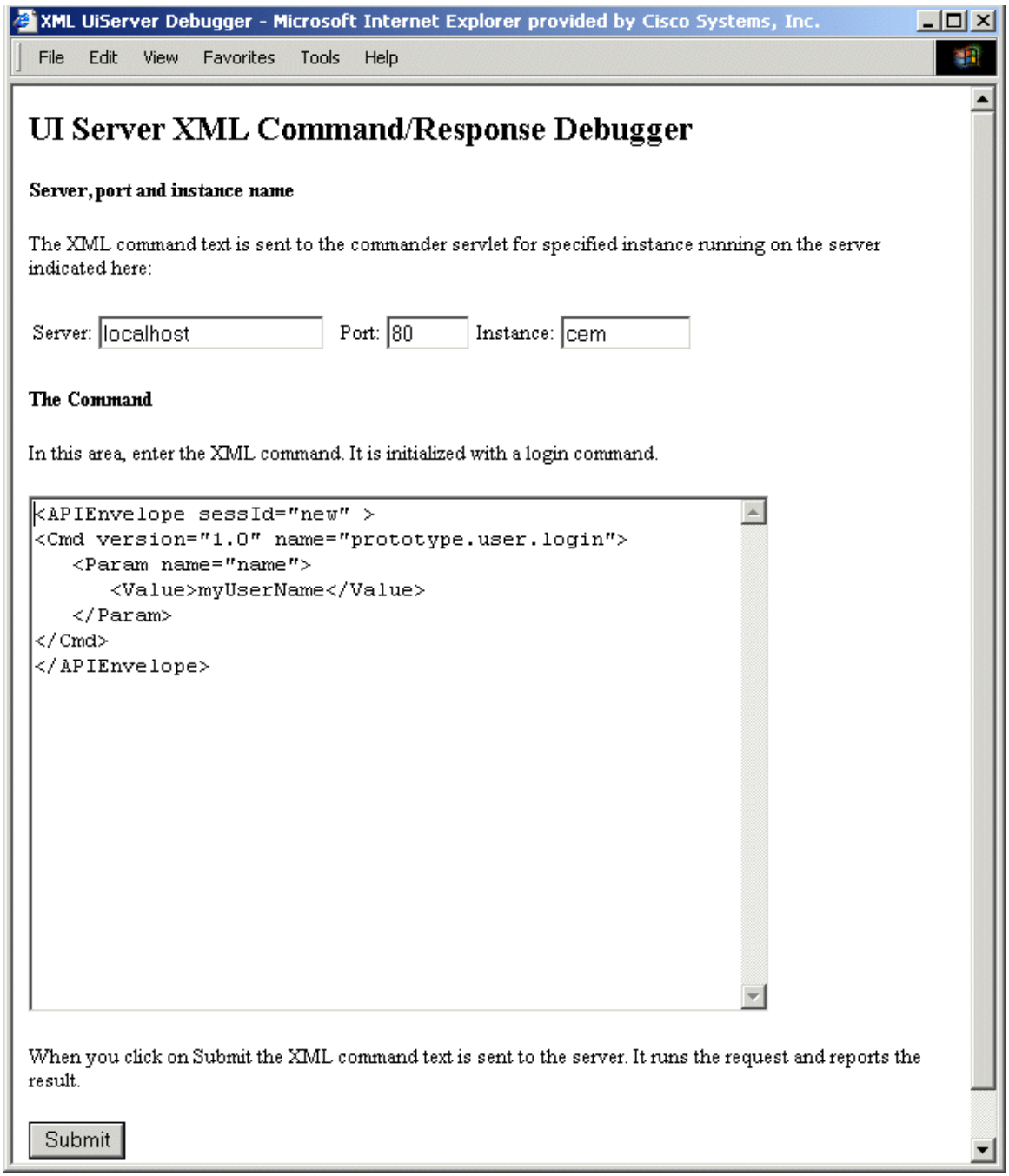
The API Server provides a simple Web-based tool in which you can enter XML for commands and view the XML that is returned after you submit the commands, as described in the rest of this section.

Accessing the UI Server XML Command/Response Debugger Page

Cisco E-Mail Manager is delivered with an HTML page that you can use to facilitate command and response testing, the UI Server XML Command/Response Debugger page.

The file for this page is named testxml1.htm; it is installed with the CEM Server in the *installation-folder*/bin folder. You can open the page from that folder, or copy the file to any other location or machine as needed.

Following is the UI Server XML Command/Response Debugger page:



Testing API Commands

To test API commands in the UI Server XML Command/Response Debugger page:

1. Enter the Server and the Port where the UI Server is running.

   Note: The Server name entered by default is "localhost". This refers to the computer on which you are running the browser, not the local computer on which

the API Server is running. If you are running the browser on a computer other than the one running the API Server, you must change the Server name.

Note: By default, the UI Server uses Port 80.

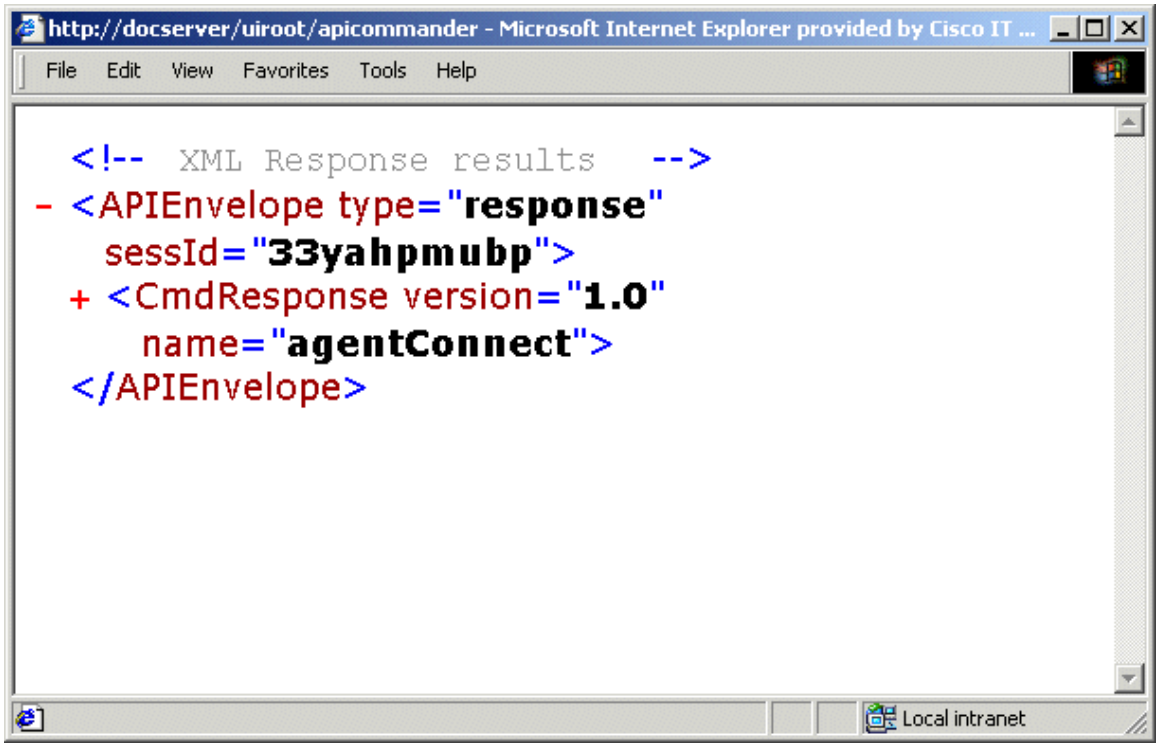
2. Enter the name of the E-Mail Manager instance to test against.

3. Enter the XML for the command in the text box on this page.

4. Click **Submit**.

Viewing Command Results

After you test an API command, the result displays in the browser, as shown below:



Roles and the API

This topic contains the following sections:

Overview of Roles

In E-Mail Manager, the capabilities of each agent are controlled by the roles. Each agent is associated with a role that grants and denies permissions to various functions.

For more information on roles, see the *Cisco E-Mail Manager Administration Guide*.

The API and Role Settings

With the exception of the role settings that affect agent state transitions, the E-Mail Manager API does not check an agent's role settings when the agent executes a command; therefore, an agent could be able to perform tasks through the API that he would not be able to perform using the Agent Desktop.

Using the roleInfo Return Structure

To provide security and limit an agent's tasks to those allowed by the agent's role, an application using the E-Mail Manager API must parse the `roleInfo` return structure and use the returned values to appropriately limit available tasks.

The `roleInfo` return structure is part of the response to the following commands:

`agentConnect`

`agentDisconnectDuplicateAndConnect`

`agentGetProperties`

Roles Settings, Agent State, and Workflow Design

Although in most cases the E-Mail Manager API does not check an agent's role settings when the agent executes commands, it does consider role settings when changing an agent's state. commands. Details are described in the Agent State topic.

Specifically, there are three roles that determine and agent's workflow:

Require push mode

Display Message screen before allowing response

Allow to use Wrap Up screen

These settings are specified in the `roleInfo` return structure.

The settings of these roles also determine how you design command workflows.

Integration with ICM

This topic contains the following sections:

Working with Messages Routed through ICM Software

Agents in ICM Routing skill groups can receive and respond to messages using the API. To open a message routed to the agent through ICM software, the agent must:

Be in the idle state.

Be in the push mode.

Use the `msgOpenRouted` command. This opens the next message routed from ICM software. The agent cannot open messages with other commands.

Configuration Requirements

Whether you are using the API or just the Agent Desktop, you configure E-Mail Manager for integration with ICM software the same way. There are no special integration configuration requirements for the API. For information on configuration tasks in an integrated environment, see the *Cisco E-Mail Manager Installation and Configuration Guide*.

Administration Requirements

Whether you are using the API or just the Agent Desktop, you must set up peripherals, skill groups, agents, and rules for integration with ICM software the same way. There are no special integration administration requirements for the API. For information on administration tasks in an integrated environment, see the *Cisco E-Mail Manager Administration Guide*.

Internationalization and the API

This topic contains the following sections:

Euclid

E-Mail Manager includes the third-part OEM product Euclid. Euclid attempts to detect the language and encoding of a sequence of bytes based on pre-determined statistics. The values of encoding fields in the MailImmutableData parameter, which is returned my many message-related commands, is set by Euclid.

There is no guarantee the Euclid correctly interprets the encoding of all e-mail messages. The accuracy of Euclid's interpretations increases with the amount of text passed into it. The minimum number of bytes Euclid requires is 16.

UTF Character Set

The E-Mail Manager API uses the UTF8 character set. This character set is determined as follows:

For Socket connections, UTF 8 is specified in the Socket Connection Protocol.

For HTTP connections, UTF8 is used automatically.

The character set is never specified in the commands sent to the API Server.

Locale Attribute

Each message has a `locale` attribute, which is returned in the MailImmutableData struct in response to the following commands:

The `locale` attribute is mapped to the character set specified in the `properties.xml` file located at:

/installation-folder/uiroot/WEB-INF/properties/default/cem/

An example of properties relevant to the locale attribute is shown below:

```
<PROPERTY name="SUPPORTED_UI_LANGUAGES" value="en-US,es-ES,x-ES,ja-
JP,x-JP"/>

<PROPERTY name="SUPPORTED_EMAIL_REPLY_LANGUAGES" value="en-US,es-
ES,x-ES,ja-JP,x-JP"/>

<PROPERTY name="SUPPORTED_EMAIL_REPLY_ENCODINGS" value="Windows-
1252,Windows-1252,Windows-1252,ISO-2022-JP,ISO-2022-JP"/>
```

Commands use the SUPPORTED_EMAIL_REPLY_ENCODINGS value as a parameter to determine how to encode e-mail before sending it.

Caution: Transcoding fails if the target encoding does not have a value for every Unicode value it transcodes. For example, the Euro symbol, \u20AC, does not have a corresponding character in ISO-2022-JP. In cases like this, the character will be substituted by a default replacement character, often a '?'.

Glossary

A

access privileges: Access privileges define which functions users based on the role are able to perform. You set access privileges when defining a role.

Administration Desktop: The Web application for completing typical administration tasks such as defining rules and maintaining agents and skill groups.

agent: An individual - a customer-contact agent, manager, or administrator - who has a distinctID with which to log in to E-Mail Manager. Each agent has an associated queue for assigned messages. Agents are specific to the E-Mail Manager instance and cannot be shared across instances.

Agent Desktop: The Web application for completing typical agent and manager tasks such as retrieving and responding to messages, working with templates, and viewing real-time reports.

alias: An e-mail address used for responses that make responses appear as if they came from a different e-mail address.

application programming interface (API): The programming interface that allows third-party applications to access E-Mail Manager functions.

archive: A holding place for messages after agents have completed working with them. Messages in the archive may still reside in the Primary Transaction Database until they have been moved to the LAMBDA database.

assign: The placement of a message in an agent or skill group queue.

attachment: A file that can be stored in E-Mail Manager for use in responses, or a file sent with an incoming message.

autoresponse: A response, using a template that is sent automatically by a rule.

autosuggestion: A template that a rule determines may be an appropriate response to a message. Such templates are not sent automatically; agents view the suggestions and must manually choose to use them.

C

category: A word or phrase that can be associated with a message by an agent (at the Response screen) or by a rule.

CEMWatcher: A tool for monitoring E-Mail Manager from other Windows computers.

Cisco Independent Reporting (CIR) database: The database that stores replicated data for reporting purposes.

Configuration Utility: The application on the E-Mail Manager server used to configure instances.

D

default notification e-mail address: An Internet e-mail address used to alert a user to a message in E-Mail Manager, if specified by rules, and for MailTrack, which is used in distribution rules.

distribution rule: A rule that processes a message at the time the message is assigned to a user's or skill group's queue. A distribution rule executes when a message is assigned to a queue by rules, or by manual reassignment from a different queue. A distribution rule does not execute when an agent claims a message from a group queue.

dynamic template: A template that contains variables that allow e-mail responses based on the template to pull data from a message, an external database, or both. Thus, responses based on the dynamic template are automatically customized for specific messages and customers.

E

escalate: The reassignment of a message while raising its priority one level.

external data access: The bringing of data in a non-E-Mail Manager database into E-Mail Manager, either for display in the user interface or into rules for additional processing.

external data access (EDA): The bringing of data in a non-E-Mail Manager database into E-Mail Manager, either for display in the user interface or into rules for additional processing.

I

InBasket: The component of E-Mail Manager that retrieves messages from one or more POP3 Mailboxes and Web forms and sends them to system rules for processing.

instance: An instance is a partition of E-Mail Manager that uses its own databases, has its own configuration, and processes e-mail messages separately. Data in one instance is not accessible to other instances. One E-Mail Manager installation can support multiple instances.

integration: For E-Mail Manager, the use of ICM software for routing messages to agents and for reporting across multiple channels.

J

JavaScript customization: The feature that allows you to use JavaScript to customize the Agent Desktop.

K

keyword: A word or phrase that can be associated with one or more templates.

L

LAMBDA: The acronym for Load Adaptive Message-Base Data Archiving. The E-Mail Manager component that moves older messages to a secondary database for storage.

library: A storage area for templates. Each template is saved in one (and only one) library, based on its content and intended use.

M

mailing list: A group of multiple e-mail addresses (list members) under one list name.

MailTrack: The forwarding of a message, as well as the tracking number, suggested response templates, comments, and other system information associated with the message.

matrix: A set of instances that can exchange messages through rules.

message access privileges: Role settings that determine which messages agents based on the role can work with, and what actions they can perform on messages.

O

opt-out list: A special list of e-mail addresses that is checked before a broadcast message is sent out. Opt-out list e-mail addresses do not receive broadcast messages, even if they are members of personal and public mailing lists.

overdue escalation: The reprocessing of messages after they have waited in a queue for longer than the time period you set.

Overdue rule tree: The rule tree that processes messages that are escalated because they have been in the queue longer than the default or queue-specific Overdue Escalation Time.

overload escalation: The reprocessing of the oldest messages in a queue after that queue contains more messages than the number allowed that you set.

Overload rule tree: The rule tree that processes messages that are escalated because the number of messages in the queue was greater than the default or queue-specific Overload Escalation Threshold.

P

pick mode: The workflow in which agents select messages from their personal queue and from skill group queues in which they are members.

POP3 Mailbox: An account on an e-mail server that E-Mail Manager polls for incoming messages destined for the InBasket.

Primary Transaction Database: The main database used by E-Mail Manager. This database stores such things as messages and agent and skill group settings.

priority: The importance of a message, which can be set automatically by rules or manually by an agent.

pull mode: The workflow in which agents retrieve messages from the Status screen by clicking Get Next for a single queue or for all queues in which the agent is a member.

push mode: The workflow in which agents do not select messages from their queue. Messages from the agent's personal queue, as well as queues in which the agent is a member, are automatically presented for reading and responding to.

Q

queue: A storage space for messages, associated with an agent or skill group.

R

real-time displays: Reports that show the current status of agents and queues.

reassign: The act of moving a message from one queue to another.

rebranding: The feature that allows you customize the interface to display your own company's information.

response: The reply to a message.

role: A collection of settings and access privileges associated with agents. Each agent is based on a role.

round-robin: The even distribution of messages to logged-in members of a skill group.

RServer: The E-Mail Manager component that processes incoming messages and serves the Administration Desktop.

rule: An object that tests a message for certain criteria and performs an action on the message if it meets those criteria. Rules are grouped into rule trees and subroutines.

rule tree: The complete series of rules that process a message.

S

service level management: A collection of several areas of functionality that enable you to manage messages so that you can ensure that: (1)agents in a skill group are working with an equal number of messages. (2)Messages that are not responded to in a set amount of time are escalated and re-processed according to rules you define. (3) Messages in queues with more messages than the number you set are escalated and reprocessed according to rules you define.

skill group: A collection of agents who work with messages assigned to the skill group's own queue.

SMTP Server: The server used by E-Mail Manager to send outgoing messages.

subroutine: A single rule, or a linked series of rules, that is not part of the main rule tree through which messages pass. Subroutines process messages when they are called by other rules.

T

team: A collection of agents grouped for reporting purposes.

template: A predefined response stored in a library for agents' use.

TServer: The E-Mail Manager component that controls database transactions.

W

WebView: The application used for reporting, accessible through the Agent Desktop.

wrap: The state agents go into when they are required to close a case after responding to a message.