Oracle® Mail

Application Developer's Guide

10*g* Release 1 (10.1.1)

B15789-01

September 2005

ORACLE®

Oracle Mail Application Developer's Guide, 10*g* Release 1 (10.1.1)

B15789-01

Primary Author: Nick Taylor

Contents

2 Oracle Mail Java API

Preface

This document contains developer examples and reference information for Oracle Mail.

Audience

Oracle Mail Application Developer's Guide is intended for developers who perform the following tasks:

- Writing PL/SQL plug-ins for Oracle Mail
- Using the Rule Management or Directory Management APIs, or writing Java Mail code to interact with Oracle Mail
- Writing custom server action rules
- Using the low-level plug-in framework

To use this document, it is assumed that you have some experience programming in PL/SQL, C, or Java.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see these Oracle resources:

- *Oracle Collaboration Suite Installation and Configuration Guide*
- *Oracle Mail Administrator's Guide*

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

PL/SQL API Reference

In recent years a number of applications like order processing, bill payment, and marketing systems rely on email as a mechanism for customer interaction. Since most of these applications already have an Oracle database back-end, they alo benefit greatly by having a PL/SQL SDK for messaging operations.

This chapter describes a set of PL/SQL packages with a set of PL/SQL functions and procedures for performing messaging and folder-related tasks like listing folders, retrieving a listing list of messages, and reading and sending messages. The SDK offers the complete set of functionality needed to develop an email client. This chapter describes each of the interfaces and how to use them along with their limitations and restrictions of use.

The audience for this chapter includes application developers and client developers who wish to integrate with an Oracle Mail server and write email-enabled applications. The reader is assumed to be familiar with Internet standards for email, such as SMTP, MIME and S/MIME.

The PL/SQL SDK

The PL/SQL SDK consists of three sets of functionality: Session Management, Folder Operations, and Message Operations.

Session Management

Session management includes setting up an email server session, authentication, and disconnect. Both simple text-password authentication and strong authentication are supported. Session Management uses and depends on the Directory and Security API for authentication.

Folder Management

Folder functions are all operations for a folder. This includes functions for listing folders and subscription management, as well as creating, deleting, renaming, searching, and sorting on a folder. The second set of folder functions are for manipulating messages in the folder, these are functions for listing messages in the folder, copying messages, saving messages, deleting messages, checking for modifications to the folder, and retrieving new mail.

Message Management

The Message functions are all operations for a message. This includes functions for reading, composing, sending, and appending messages. To hide the complexity of MIME structure and message encoding, it contains functions to get headers, to get individual body parts, to get body part content, and to construct messages using

individual body parts. It also includes S/MIME functionality to verify message signatures, decrypt messages, and send encrypted and/or signed messages.

Overview

Concepts

This section discusses the following e-mail related concepts:

- Folder UIDL
- Message UID
- Message Flags
- New and Recent Messages
- MIME Level

Folder UIDL

When a folder is created, a validity value is assigned to the folder guaranteeing that as long as the value does not change, the message unique identifier (UID) assigned to the messages in the folder is valid. This value is called folder unique identifier validity (UIDL).

Each message in a folder is assigned a unique identifier validity (UID) value. The UID value persists across sessions, so application may cache these message UID values for message reference in the future. Message UID is always assigned in ascending order in the folder. As each message is added to the folder, it is assigned a higher UID value. The maximum UID permitted is $2^{32} - 1 = 4$ billion. Once the maximum has been reached the message UIDs in the folder are re-assigned. The holes in the message UID sequence created by messages deletion are compacted. To reflect this, the folder validity value (Folder UIDL) is changed. When the folder validity value is changed, the application must discard any message UID cache for this folder and re-fetch the message UID values.

Message Flags

Message content cannot be modified, but mail users can change the flag property. When a message is delivered into the user's INBOX folder, the message is not marked with any flag. After the user read a message, the application marks the message with the **seen** flag.The supported flags are described below:

Table 1–1 Supported Message Flags

Flag	Description
MAIL_MESSAGE.GC_ SEEN_FLAG	The message has been read
MAIL_MESSAGE.GC_ FLAGGED_FLAG	Marks messages for urgent or special attention
MAIL_MESSAGE.GC_ ANSWERED_FLAG	The message has been replied to
MAIL_MESSAGE.GC_ DELETED_FLAG	The message is marked to be removed later with the expunge command
MAIL_MESSAGE.GC_ DRAFT_FLAG	The message has not completed composition

New and Recent Messages

New and recent messages only differ when there are multiple client sessions accessing the same mail folder at the same time. A message is considered recent when no other mail client session has retrieved the message. A message is considered new if the current mail user session has not seen the message.

MIME Level

A message MIME level is a string used to identify a specific part of a message. The application does not have to know anything about it. The MIME level is incorporated as part of the message and body-part objects that are passed around. It is for internal use to identify a specific message part.

Mail Objects

There are three distinct mail objects in the Oracle Mail Server:

- mail account or session
- mail folder
- mail message

A mail account contains many mail folders. A mail folder contains many mail messages. An INBOX mail folder is a special folder created when the mail user account is created. The INBOX folder cannot be deleted. All incoming messages are delivered into the INBOX folder.

A mail user must be authenticated before performing mail operations. The MAIL_SESSION package provides the functions. When a user is authenticated, a valid session identifier is returned from the authentication routine. This session identifier is passed around to other mail operations that require a valid authenticated session. The mail user session identifier is only valid in the same database session that it has been authenticated in. Multiple mail user sessions are supported so that multiple mail users can authenticate on the same database session. Each authenticated session is associated with a unique identifier.

To access any message in a folder, the user must open the folder. There is only one opened folder in a mail session at any given time. The open folder changes when the folder is closed or another folder is opened.

Note: The PL/SQL API contains references to `dbms_sql.varchar2_table` and `dbms_sql.number_table` types

---

See Also: The Oracle Supplied Packages Reference manual for the description of these two types.

MAIL_FOLDER_OBJ

The MAIL_FOLDER_OBJ object uniquely identifies a mail folder. It is defined as follows:

```
MAIL_FOLDER_OBJ(
     name VARCHAR2(1024),
     id   NUMBER
);
MAIL_FOLDER_LIST AS TABLE OF MAIL_FOLDER_OBJ;
```

- name is the full path of the folder name.
- id is a unique folder identifier.
- MAIL_FOLDER_LIST is a nested table of MAIL_FOLDER_OBJ objects.

MAIL_FOLDER_DETAIL

The MAIL_FOLDER_DETAIL object contains information pertaining to a specific folder. It is defined as follows:

```
MAIL_FOLDER_DETAIL (
    uidl             NUMBER,
    num_voice_recent NUMBER,
    num_fax_recent   NUMBER,
    total_recent     NUMBER,
    num_voice_unseen NUMBER,
    num_fax_unseen   NUMBER,
    total_unseen     NUMBER,
    total_msgs       NUMBER
);
CREATE TYPE MAIL_FOLDER_LIST AS TABLE OF MAIL_FOLDER_OBJ;
```

The MAIL_FOLDER_DETAIL object contains information on the folder UIDL, number of messages, total number of unseen and recent messages, and number of unseen and recent voice and fax messages.

MAIL_SORT_CRITERIA_ELEMENT

The MAIL_SORT_CRITERIA_ELEMENT object is used to represent a sort criteria. It is defined as follows:

```
MAIL_SORT_CRITERIA_ELEMENT(
    sort_header VARCHAR2(240),
    sort_order  INTEGER
);
MAIL_SORT_CRITERIA AS TABLE OF MAIL_SORT_CRITERIA_ELEMENT;
```

- sort_header is the message header used to perform the sort. The supported header names are:
  - CC
  - DATE
  - SUBJECT
  - SIZE
  - INTERNAL_DATE
- sort_order indicates whether to sort the header field in ascending or descending order. The values are:
  - MAIL_FOLDER.SORT_ASC
  - MAIL_FOLDER.SORT_DESC
- MAIL_SORT_CRITERIA is a nested table of MAIL_SORT_CRITERIA_ELEMENT objects.

MAIL_MESSAGE_OBJ

The MAIL_MESSAGE_OBJ object uniquely identifies a mail message. It is defined as follows:

```
MAIL_MESSAGE_OBJ (
    folder_id  NUMBER,
    msg_uid    NUMBER,
    mime_level VARCHAR2(240)
);
MAIL_MESSAGE_LIST IS TABLE OF MAIL_MESSGAGE_OBJ;
```

- folder_id is a unique folder identifier
- msg_uid uniques identifies a message within the folder.
- mime_level either has the value 0 to indicate that this message object is a top-level message, or other value to indicate an included message object.
- MAIL_MESSAGE_LIST is a nested table of MAIL_MESSAGE_OBJ objects.

MAIL_BODYPART_OBJ

The MAIL_BODYPART_OBJ object uniquely identifes a mail message part. It is defined as follows:

```
MAIL_BODYPART_OBJ (
    content_type VARCHAR2(240),
    mime_level VARCHAR2(240),
    folder_id NUMBER,
    msg_uid NUMBER,
    smime_ind NUMBER
);
MAIL_BODYPART_LIST IS TABLE OF MAIL_BODYPART_OBJ;
```

- folder_id is a unique folder identifier.
- msg_uid uniquely identifies a message within the folder.
- mime_level identifies the specific body-part of the message.
- smime_ind is a value of 1 that indicates that the body-part is from a decrypted message.
- mail_bodypart_list is a nested table of MAIL_BODYPART_OBJ objects.

MAIL_MESSAGE_HEADER

The MAIL_MESSAGE_HEADER object identifes a mail message header. It is defined as follows:

```
MAIL_MESSAGE_HEADER (
    msg_uid            NUMBER,
    from_str           VARCHAR2(2000),
    subject            VARCHAR2(2000),
    content_type       VARCHAR2(2000),
    x_orcl_messagetype VARCHAR2(2000),
    x_orcl_mediatype   VARCHAR2(2000),
    sensitivity        VARCHAR2(2000),
    importance         VARCHAR2(2000)
);
TYPE MAIL_MESSAGE_HEADER_LIST IS TABLE OF MAIL_MESSAGE_HEADER;
```

- msg_uid uniquely identifies a message within the folder.
- from_str identifies the "From" part of the header
- subject identifies the message subject
- content_type is the "Content-type" of the message
- x_orcl_messagetype is the X-ORCL-MESSAGETYPE header value
- x_orcl_mediatype is the X-ORCL-MEDIATYPE header value
- sensitivity is the sensitivity header value
- importance is the importance header value

MAIL_RECOVERY Package

> **Note:** This package will no longer be supported from 10.1.1 onwards. Please do not use this package. It will be deprecated in future releases.

The MAIL_RECOVERY package is provided to recover deleted e-mails, using the Oracle log miner feature.

The redo list file is used by the mail_recovery package to retrieve the list of redo logs. This file must be provided by anadministrator, and can be created manually through a text editor or by outputting a directory list command on UNIX or Windows NT.

The file contains redo log filenames with their full path, and must be listed in separate lines. The redo logs can either be online redo logs, archived logs or both.

It is important to verify that `init.ora` parameter UTL_FILE_DIR is set to access the redo list file. For example:

```
/oracle/database/redo01.log
/oracle/database/redo02.log
/oracle/database/redo03.log
```

The MAIL_RECOVERY package contains the following procedures:

- SETUP_LOGMNR Procedure
- RECOVER_MESSAGES Procedure

SETUP_LOGMNR Procedure

The `setup_logmnr` procedure initializes the log miner for recovery.

Syntax

```
PROCEDURE setup_logmnr(
    p_dictionary_filename   IN VARCHAR2,
    p_redolist_location     VARCHAR2,
    p_redolist_filename     VARCHAR2
    p_starttime              DATE DEFAULT '01-jan-1988',
    p_endtime                DATE DEFAULT '01-jan-2099')
```

Parameters

Table 1–2 SETUP_LOGMNR Procedure

Parameters	Description
`p_dictionary_filename`	Name of the log miner data dictionary file with the full path
`p_redolist_location`	Directory location of redo list file
`p_redolist_filename`	File name of redo list file
`p_starttime`	Start time of the redo list. Only consider redo records with the time stamp greater than or equal to the start time specified
`p_endtime`	End time of the redo list. Only consider redo records with time stamp less than or equal to the end time specified

RECOVER_MESSAGES Procedure

The `recover_messages` procedure performs the recovery for a user and restores the messages in a specified folder.

Syntax

```
PROCEDURE recover_messages(
    p_domainname IN VARCHAR2,
    p_username   IN VARCHAR2,
    p_foldername IN VARCHAR2,
    p_autocommit IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 1–3 RECOVER_MESSAGES Procedure

Parameter	Description
`p_domainname`	Domain name of the user
`p_username`	Oracle Mail user name for which recovery is performed
`p_foldername`	Folder name where recovered messages are restored. If NULL, it is passed, and recover_messages creates a new folder with name `RECMSG_current_date_time`.
`p_autocommit`	If True: Frequent commits are performed within `recover_messages`. If False: No commits are performed inside the `recover_messages` procedure. The `end_logmnr` procedure finishes the logminer session.

MAIL_SESSION Package

The MAIL_SESSION package provides user authentication and log out functionality. A user can create multiple mail sessions using the same database session by calling `MAIL_SESSION.login()` multiple times. Each session ID identifies one valid mail session.

The MAIL_SESSION Package contains the following procedures:

- LOGIN Procedure
- LOGOUT Procedure
- GET_CURRENT_USAGE Procedure

LOGIN Procedure

This procedure authenticates a user by user name and password. One of the overloaded versions of this API also accepts an authenticated ES_LOGIN_CONTEXT and returns a mail session identifier.

Throws Exceptions

mail_errors.login_err

Syntax

```
PROCEDURE login (
    user_name   IN VARCHAR2,
    password    IN VARCHAR2,
    domain      IN VARCHAR2,
    ldap_host   IN VARCHAR2,
    session_id  OUT NUMBER,
    ldap_port   IN NUMBER DEFAULT 389
);
PROCEDURE login (
    user_address  IN VARCHAR2,
    password      IN VARCHAR2,
    ldap_host     IN VARCHAR2,
    session_id    OUT NUMBER,
    ldap_port     IN NUMBER DEFAULT 389
);
```

Parameters

Table 1–4 LOGIN parameters

Parameter	Description
user_name	User account name, without the domain part
password	User password
user_address	User Internet address: user_name@domain
domain	User domain
ldap_host	The host name where Oracle Internet Directory is configured
ldap_port	The port Oracle Internet Directory is listening to; the default is 389
esds_context	The authenticated ES_LOGIN_CONTEXT object.
session_id	An unique identifier that represents this user authenticated session

LOGOUT Procedure

This procedure releases all resources associated with this user session.

Syntax

```
PROCEDURE logout (
    session_id IN NUMBER
);
```

Parameters

Table 1–5 LOGOUT Procedure parameters

Parameters	Description
session_id	An identifier that represents a user's authenticated session

GET_CURRENT_USAGE Procedure

This procedure returns the current user's usage amount.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE get_current_usage (
    session_id IN NUMBER,
    usage OUT NUMBER
);
```

Parameters

Table 1–6 GET_CURRENT_USAGE Procedure parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`usage`	User's current usage in bytes

MAIL_FOLDER Package

The MAIL_FOLDER package provides folder-related functionality. The validity of the session is checked before performing any operation. The search and sort features are based on the IMAP4 protocol. The search feature includes Oracle Text based searches.

The MAIL_FOLDER contains the following procedures and functions:

- GET_FOLDER_OBJ Procedure
- GET_SEPARATOR
- LIST_TOPLEVEL_FOLDERS Procedure
- LIST_FOLDERS Procedure
- LIST_TOPLEVEL_SUBDFLDRS Procedure
- LIST_SUBSCRIBED_FOLDERS Procedure
- IS_FOLDER_SUBSCRIBED Function
- SUBSCRIBE_FOLDER Procedure
- UNSUBSCRIBE_FOLDER Procedure
- HAS_FOLDER_CHILDREN Function
- GET_FOLDER_EXPIRY Procedure
- SET_FOLDER_EXPIRY Procedure
- GET_FOLDER_DETAILS Procedure
- CREATE_FOLDER Procedure
- DELETE_FOLDER Procedure
- RENAME_FOLDER Procedure
- OPEN_FOLDER Procedure
- GET_FOLDER_MESSAGES Procedure
- GET _MESSAGE Procedure
- CLOSE_FOLDER Procedure
- GET_MSG_FLAGS Procedure
- SET_MSG_FLAGS Procedure
- SET_MSG_USRFLAGS Procedure
- DELETE_MESSAGES Procedure
- EXPUNGE_FOLDER Procedure
- IS_FOLDER_OPEN Function
- CHECK_NEW_MESSAGES Function
- CHECK_RECENT_MESSAGES Function
- GET_NEW_MESSAGES Procedure
- COPY_MESSAGES Procedure
- MOVE_MESSAGES Procedure

- IS_FOLDER_MODIFIED Function
- SORT_FOLDER Procedure
- SEARCH_FOLDER Procedure

GET_FOLDER_OBJ Procedure

This procedure returns a folder object, given a folder name. If the folder does not exist on the mail store, a `FOLDER_NOT_FOUND` exception is raised.

Throws Exception

`mail_errors.unauthenticated_err`

`mail_errors.folder_not_found_err`

Syntax

```
PROCEDURE get_folder_obj (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2,
    folder_obj    OUT MAIL_FOLDER_OBJ
);
```

Parameters

Table 1–7 GET_FOLDER_OBJ Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of a folder name.
`folder_obj`	The MAIL_FOLDER_OBJ returned.

GET_SEPARATOR

This function returns the folder separator.

Syntax

```
FUNCTION get_separator return VARCHAR2;
```

LIST_TOPLEVEL_FOLDERS Procedure

This procedure returns a list of top-level folder objects.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE list_toplevel_folders (
    session_id    IN NUMBER,
    folder_list   OUT MAIL_FOLDER_LIST
);
```

Parameters

Table 1–8 LIST_TOPLEVEL_FOLDERS Parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_list`	A list of top-level folder objects

LIST_FOLDERS Procedure

This procedure returns a list of direct child folder objects, given the parent folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_not_found_err`

Syntax

```
PROCEDURE list_folders (
    session_id    IN NUMBER,
    parent_name   IN VARCHAR2,
    folder_list   OUT MAIL_FOLDER_LIST
);
PROCEDURE list_folders (
    session_id  IN  NUMBER,
    parent_obj  IN  MAIL_FOLDER_OBJ,
    folder_list OUT MAIL_FOLDER_LIST
);
```

Parameters

Table 1–9 LIST_FOLDERS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`parent_name`	The full path of the parent folder
`parent_obj`	The MAIL_FOLDER_OBJ representing the parent folder
`folder_list`	The list of child folder objects under the parent folder

LIST_TOPLEVEL_SUBDFLDRS Procedure

This procedure returns a list of top-level subscribed folders.

Throws Exception

```
mail_errors.unauthenticated_err
```

Syntax

```
PROCEDURE list_toplevel_subdfldrs (
    session_id      IN NUMBER,
    foldername_list OUT DBMS_SQL.VARCHAR2_TABLE
);
```

Parameters

Table 1–10 LIST_TOPLEVEL_SUBDFLDRS parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`foldername_list`	The list of subscribed child folder objects under the parent folder

LIST_SUBSCRIBED_FOLDERS Procedure

This procedure returns a list of subscribed child folders, given the parent folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE list_subscribed_folders (
    session_id IN NUMBER,
    parent_name IN VARCHAR2,
    foldername_list OUT DBMS_SQL.VARCHAR2_TABLE
);
PROCEDURE list_subscribed_folders (
    session_id IN NUMBER,
    parent_obj IN MAIL_FOLDER_OBJ,
    foldername_list OUT DBMS_SQL.VARCHAR2_TABLE
);
```

Parameters

Table 1–11 LIST_SUBSCRIBED_FOLDERS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`parent_name`	The full path of the parent folder name
`parent_obj`	The MAIL_FOLDER_OBJ representing the parent folder
`foldername_list`	The list of subscribed child folder objects under the parent folder

IS_FOLDER_SUBSCRIBED Function

This function tests to see if the folder is subscribed.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
FUNCTION is_folder_subscribed (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2
) return BOOLEAN;
```

Parameters

Table 1–12 IS_FOLDER_SUBSCRIBED parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name

SUBSCRIBE_FOLDER Procedure

This procedure subscribes the specified folder. Errors are not returned if the folder has already been subscribed.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE subscribe_folder (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2
);
```

Parameters

Table 1–13 SUBSCRIBE_FOLDER parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name

UNSUBSCRIBE_FOLDER Procedure

This procedure unsubscribes the specified folder. Errors are not returned if the folder has not been subscribed at the time of the call.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE unsubscribe_folder (
    session_id  IN NUMBER,
    folder_name IN VARCHAR2
);
```

Parameters

Table 1–14 UNSUBSCRIBE_FOLDER Procedure parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name

HAS_FOLDER_CHILDREN Function

This function tests to see if any child folders exist.

Throws Exception

`mail_errors.unauthenticated_err`

Syntax

```
FUNCTION has_folder_children (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ
) return BOOLEAN;
```

Parameters

Table 1–15 HAS_FOLDER_CHILDREN Function parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_obj`	The folder object

GET_FOLDER_EXPIRY Procedure

This function returns folder expiry information; specifically, the number of days to keep messages in the folder . A value of 0 means messages will not be automatically removed by the HouseKeeper server.

Throws Exception

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE get_folder_expiry (
    session_id   IN   NUMBER,
    folder_name  IN   VARCHAR2,
    expiry       OUT  NUMBER
);
```

Parameters

Table 1–16 GET_FOLDER_EXPIRY Function parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of a folder name
`expiry`	Number of days messages will stay in the folder.

SET_FOLDER_EXPIRY Procedure

This function sets the folder expiry; specifically, the number of days messages in the folder will be kept. A value of 0 means messages will not be automatically removed by the HouseKeeper server.

Throws Exception

```
mail_errors.unauthenticated_err
```

Syntax

```
PROCEDURE get_folder_expiry (
    session_id   IN   NUMBER,
    folder_name  IN   VARCHAR2,
    expiry       OUT  NUMBER
);
```

Parameters

Table 1–17 SET_FOLDER_EXPIRY Function parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
folder_name	The full path of a folder name
expiry	Number of days messages will stay in the folder.

GET_FOLDER_DETAILS Procedure

This procedure returns folder information, such as the folder UIDL identifier, total message count, number of unseen messages, and number of recent messages.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE get_folder_details (
    session_id        IN NUMBER,
    folder_obj        IN MAIL_FOLDER_OBJ,
    folder_detail_obj OUT MAIL_FOLDER_DETAIL
);
```

Parameters

Table 1–18 GET_FOLDER_DETAILS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_obj`	The folder object
`folder_detail_obj`	A folder detail object that contains information about the folder

CREATE_FOLDER Procedure

This procedure creates a folder on the mail store with the given name, and returns a folder object representing the new folder. When this folder is created, any folders in its path that don't exist are also created. A FOLDER_TYPE_ERR is returned if the parent folder does not allow creation of subfolders.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_already_exists_err

mail_errors.folder_type_err

Syntax

```
PROCEDURE create_folder (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2,
    folder_obj    OUT MAIL_FOLDER_OBJ
);
```

Parameters

Table 1–19 CREATE_FOLDER Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
folder_name	The full path of a folder name
folder_obj	The MAIL_FOLDER_OBJ returned

DELETE_FOLDER Procedure

This procedure deletes the specified folder. If a recursive flag is set to true, all the messages in the folder, all sub-folders and their messages, and the folder itself are removed. A recursive flag set to false performs the following actions:

If no sub-folder exists, the messages in the folder and the folder itself are removed.

If sub-folders exist, the messages in the folder are removed, and the folder is marked as not selectable (the OPEN_FOLDER operation on this folder can fail).

If this procedure is called with a folder that is marked as not selectable and contains a subfolder, the MAIL_ERRORS.FOLDER_TYPE_ERR is thrown. If called with folder that has already marked as \NOSELECT and still contains sub-folder, the MAIL_ERRORS.FOLDER_TYPE_ERR will be thrown.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.operation_not_allowed`

`mail_errors.folder_type_err`

`mail_errors.folder_not_found_err`

Syntax

```
PROCEDURE delete_folder (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2,
    recursive     IN BOOLEAN DEFAULT false
);
PROCEDURE delete_folder (
    session_id   IN NUMBER,
    folder_obj   IN MAIL_FOLDER_OBJ,
    recursive    IN BOOLEAN DEFAULT false
);
```

Parameters

Table 1–20 DELETE_FOLDER Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name
`folder_obj`	The `MAIL_FOLDER_OBJ` representing the folder
`recursive`	If set to true, it deletes the folder and any sub-folders

RENAME_FOLDER Procedure

This procedure renames the specified folder and returns the new folder object. If renaming INBOX, all the messages in INBOX are moved to the new folder name, and the INBOX folder will remain with no messages.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_not_found_err`

`mail_errors.folder_already_exists_err`

`mail_errors.operation_not_allowed`

Syntax

```
PROCEDURE rename_folder (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2,
    new_folder_name  IN VARCHAR2,
    folder_obj       OUT MAIL_FOLDER_OBJ
);
PROCEDURE rename_folder (
    session_id    IN NUMBER,
    folder_obj    IN OUT MAIL_FOLDER_OBJ,
    new_folder_name   IN VARCHAR2
);
```

Parameters

Table 1–21 RENAME_FOLDER Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name
`new_folder_name`	The new name for the folder
`folder_obj`	The `MAIL_FOLDER_OBJ` representing the folder

OPEN_FOLDER Procedure

This procedure opens the specified folder and returns the folder object. If the folder is marked with the \NOSELECT flag, a `mail_errors.folder_type_err` will be thrown.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_not_found_err`

`mail_errors.folder_type_err`

Syntax

```
PROCEDURE open_folder (
    session_id   IN  NUMBER,
    folder_name  IN  VARCHAR2,
    folder_obj   OUT MAIL_FOLDER_OBJ
);
```

Parameters

Table 1–22 OPEN_FOLDER Procedure

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name
`folder_obj`	The `MAIL_FOLDER_OBJ` representing the folder

GET_FOLDER_MESSAGES Procedure

This procedure returns all of the messages with the specified message type in the current open folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get_folder_messages (
    session_id IN NUMBER,
    message_list OUT MAIL_MESSAGE_LIST,
    message_type IN NUMBER DEFAULT MAIL_MESSAGE.GC_ALL_MAIL
);
```

Parameters

Table 1–23 GET_FOLDER_MESSAGES Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_list`	A list of message objects that belongs to the folder
`message_type`	The type of message to be retrieved. The default is to retreive all types. The message types are defined in the `MAIL_MESSAGE` package specification. Values are: ■ `MAIL_MESSAGE.GC_ALL_MAIL` ■ `MAIL_MESSAGE.GC_EMAIL` ■ `MAIL_MESSAGE.GC_VOICE_MAIL` ■ `MAIL_MESSAGE.GC_FAX_MAIL` ■ `MAIL_MESSAGE.GC_NEWS_MAIL`

GET _MESSAGE Procedure

This procedure returns the message object corresponding to the message UID specified in the current open folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get _message (
    session_id  IN  NUMBER,
    message_uid IN  NUMBER,
    message_obj OUT MAIL_MESSAGE_OBJ
);
```

Parameters

Table 1–24 GET_MESSAGE Procedure

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_uid`	The message identifier
`message_obj`	The `MAIL_MESSAGE_OBJ` type that corresponds to the specified UID

CLOSE_FOLDER Procedure

This procedure closes the current open folder. If the EXPUNGE_FLAG is set to true, all messages in the folder that are marked with the Deleted flag are removed.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_closed_err

Syntax

```
PROCEDURE close_folder (
    session_id   IN NUMBER,
    expunge_flag IN BOOLEAN
);
```

Parameters

Table 1–25 CLOSE_FOLDER Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
expunge_flag	A flag indicating whether to expunge the folder before closing

GET_MSG_FLAGS Procedure

This procedure returns message flags belonging to the message list specified in the current open folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get_msg_flags (
    session_id IN NUMBER,
    message_list IN MAIL_MESSAGE_LIST,
    message_flags OUT DBMS_SQL.NUMBER_TABLE
);
PROCEDURE get_msg_flags (
    session_id IN NUMBER,
    message_uid_list IN DBMS_SQL.NUMBER_TABLE,
    message_flags OUT DBMS_SQL.NUMBER_TABLE
);
PROCEDURE get_msg_flags (
    session_id        IN  NUMBER,
    message_list      IN  MAIL_MESSAGE_LIST,
    message_flags     OUT DBMS_SQL.NUMBER_TABLE,
    message_usrflags  OUT DBMS_SQL.VARCHAR2_TABLE
);
PROCEDURE get_msg_flags (
    session_id        IN   NUMBER,
    message_uid_list  IN   DBMS_SQL.NUMBER_TABLE,
    message_flags     OUT  DBMS_SQL.NUMBER_TABLE,
    message_usrflags  OUT  DBMS_SQL.VARCHAR2_TABLE
);
```

Parameters

Table 1–26 GET_MSG_FLAGS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_list`	A list of message objects that belongs to the folder
`message_uid_list`	A list of message UIDs identifying messages in the folder
`message_flags`	A list of message flags corresponding to the list of requested messages. Flag values are defined in the MAIL_MESSAGE package specification. Values are: ■ `MAIL_MESSAGE.GC_SEEN_FLAG` ■ `MAIL_MESSAGE.GC_FLAGGED_FLAG` ■ `MAIL_MESSAGE.GC_ANSWERED_FLAG` ■ `MAIL_MESSAGE.GC_DELETED_FLAG` ■ `MAIL_MESSAGE.GC_DRAFT_FLAG`
`message_usrflags`	A list of user-defined flags. They are a list of space-separated user-defined flag strings.

SET_MSG_FLAGS Procedure

This procedure sets or unsets the message flags belonging to the list of messages specified in the current open folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE set_msg_flags (
    session_id    IN NUMBER,
    message_list  IN MAIL_MESSAGE_LIST,
    flags         IN NUMBER,
    set_flag      IN BOOLEAN
);
PROCEDURE set_msg_flags (
    session_id        IN NUMBER,
    message_uid_list  IN DBMS_SQL.NUMBER_TABLE,
    flags             IN NUMBER,
    set_flag          IN BOOLEAN
);
```

Parameters

Table 1–27 SET_MSG_FLAGS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_list`	A list of message objects that belongs to the folder
`message_uid_list`	A list of message UIDs identifying messages in the folder
`flags`	A list of message flags corresponding to the list of requested messages. Flag values are defined in the MAIL_MESSAGE package specification. Values are: ■ `MAIL_MESSAGE.GC_SEEN_FLAG` ■ `MAIL_MESSAGE.GC_FLAGGED_FLAG` ■ `MAIL_MESSAGE.GC_ANSWERED_FLAG` ■ `MAIL_MESSAGE.GC_DELETED_FLAG` ■ `MAIL_MESSAGE.GC_DRAFT_FLAG`
`set_flag`	If true, sets the value of flags. Otherwise, it unsets the value.

SET_MSG_USRFLAGS Procedure

This procedure toggles user-defined message flags for the specified message object.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE set_msg_usrflags (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    message_usrflags IN VARCHAR2,
    set_flag         IN BOOLEAN
);
```

Parameters

Table 1–28 SET_MSG_USRFLAGS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`message_usrflags`	User defined message flag string. It contains a list of user defined flag strings separated by space.
`set_flag`	If true, set the flags specified; otherwise, un-set the flags specified.

DELETE_MESSAGES Procedure

This procedure deletes the list of messages specified in the current open folder. This is equivalent to marking the messages as deleted and performing an expunge operation on the folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE delete_messages (
    session_id IN NUMBER,
    message_uid_list IN DBMS_SQL.NUMBER_TABLE
);
```

Parameters

Table 1–29 DELETE_MESSAGES Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_uid_list`	A list of message UIDs identifying messages in the folder

EXPUNGE_FOLDER Procedure

This procedure removes all messages in the current open folder if the `gc_deleted_flag` flag is set.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE expunge_folder (
    session_id IN NUMBER,
);
```

Parameters

Table 1–30 EXPUNGE_FOLDER Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session

IS_FOLDER_OPEN Function

This function tests to see if the folder is the same folder currently selected in the user's session.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
FUNCTION is_folder_open (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ
) return BOOLEAN;
```

Parameters

Table 1–31 IS_FOLDER_OPEN parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_obj`	The folder object

CHECK_NEW_MESSAGES Function

This function tests to see if there are any new messages in the currently selected folder. New messages are messages not seen by the folder since the last GET_NEW_MESSAGES call. When the folder is first opened, the last message considered retrieved is the message last seen by any mail client. After that the last message is changed accordingly by calls to GET_FOLDER_MESSAGE and GET_NEW_MESSAGES.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_closed_err

Syntax

```
FUNCTION check_new_messages (
    session_id   IN NUMBER,
    message_type IN NUMBER DEFAULT MAIL_MESSAGE.GC_ALL_MAIL
) return BOOLEAN;
```

Parameters

Table 1–32 CHECK_NEW_MESSAGES Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_type	The type of message to be retrieved. The default is to get all types. The message types are defined in the MAIL_MESSAGE package specification.

CHECK_RECENT_MESSAGES Function

This function tests to see if there are any recent messages in the specified folder. Recent messages are messages that have not been retrieved by any mail client. This procedure can be called on a closed folder.

Throws Exception

`mail_errors.unauthenticated_err`

`mail_errors.folder_not_found_err`

Syntax

```
FUNCTION check_recent_messages (
    session_id    IN NUMBER,
    folder_name   IN VARCHAR2,
    message_type  IN NUMBER DEFAULT MAIL_MESSAGE.GC_ALL_MAIL
) return BOOLEAN;
```

Parameters

Table 1–33 CHECK_RECENT_MESSAGES Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_name`	The full path of the folder name
`message_type`	The type of message to be retrieved. The default is to retreive all types. Message types are defined in the MAIL_MESSAGE package specification. Values are: ■ `MAIL_MESSAGE.GC_ALL_MAIL` ■ `MAIL_MESSAGE.GC_EMAIL` ■ `MAIL_MESSAGE.GC_VOICE_MAIL` ■ `MAIL_MESSAGE.GC_FAX_MAIL` ■ `MAIL_MESSAGE.GC_NEWS_MAIL`

GET_NEW_MESSAGES Procedure

This procedure returns all the new messages in the currently selected folder. New messages are messages not seen by the folder since last message retrieval. When the folder is first opened, the last message considered retrieved is the last message seen by any mail client after the last message is changed accordingly by calls to GET_FOLDER_MESSAGES and GET_NEW_MESSAGES. If MESSAGE_TYPE is specified, only messages in the specified type are returned.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_closed_err

Syntax

```
PROCEDURE get_new_messages (
    session_id IN NUMBER,
    message_list OUT MAIL_MESSAGE_LIST,
    message_type IN NUMBER DEFAULT MAIL_MESSAGE.GC_ALL_MAIL
);
```

Parameters

Table 1–34 GET_NEW_MESSAGES Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_list	A list of new message objects
message_type	The type of message to be retrieved. The default is to retreive all types. Message types are defined in the MAIL_MESSAGE package specification. Values are: ■ MAIL_MESSAGE.GC_ALL_MAIL ■ MAIL_MESSAGE.GC_EMAIL ■ MAIL_MESSAGE.GC_VOICE_MAIL ■ MAIL_MESSAGE.GC_FAX_MAIL ■ MAIL_MESSAGE.GC_NEWS_MAIL

COPY_MESSAGES Procedure

This procedure copies messages in the currently selected folder to another folder. If the destination folder has the \NOSELECT flag set, the MAIL_ERRORS.FOLDER_TYPE_ERR exception is thrown. If the specified message does not belong to the current open folder, the MAIL_ERRORS.PARAM_PARSE_ERR exception is thrown.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_closed_err

mail_errors.folder_not_found_err

mail_errors.folder_type_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE copy_messages (
    session_id IN NUMBER,
    message_list IN MAIL_MESSAGE_LIST,
    to_folder_name IN VARCHAR2
);
PROCEDURE copy_messages (
    session_id IN NUMBER,
    message_uid_list IN DBMS_SQL.NUMBER_TABLE,
    to_folder_name IN VARCHAR2
);
```

Parameters

Table 1–35 COPY_MESSAGES Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_list	A list of message objects
message_uid_list	A list of message UIDs
to_folder_name	The full path of the destination folder name

MOVE_MESSAGES Procedure

This procedure moves messages in the current folder to another folder. If the destination folder is marked with the \NOSELECT flag (meaning it cannot contain messages), the mail_errors.folder_type err is thrown.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.folder_closed_err

mail_errors.folder_not_found_err

mail_errors.folder_type_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE move_messages (
    session_id     IN NUMBER,
    message_list   IN MAIL_MESSAGE_LIST,
    to_folder_name IN VARCHAR2
);
PROCEDURE move_messages (
    session_id       IN NUMBER,
    message_uid_list IN DBMS_SQL.NUMBER_TABLE,
    to_folder_name   IN VARCHAR2
);
PROCEDURE move_messages (
    session_id       IN NUMBER,
    from_folder_obj  IN MAIL_FOLDER_OBJECT,
    message_uid_list INDBMS_SQL.NUMBER_TABLE,
    to_folder_name   IN VARCHAR2
);
```

Parameters

Table 1–36 MOVE_MESSAGES Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_list	A list of message objects
message_uid_list	A list of message UIDs
from_folder_name	The MAIL_FOLDER_OBJ representing the source folder.
to_folder_name	The full path of the destination folder name

IS_FOLDER_MODIFIED Function

This function tests to see if any messages in the currently selected folder have been modified from another session. A folder is modified if any changes in message flags or deletion of messages were made in the folder by another client or session. If the folder was modified, users have to re-issue the `GET_NEW_MESSAGES` procedure to be synchronized with the mail store.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
FUNCTION is_folder_modified (
    session_id IN NUMBER
);
```

Parameters

Table 1–37 IS_FOLDER_MODIFIED Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session

SORT_FOLDER Procedure

This procedure sorts the folder given the sort criteria and returns an ordered list of message UIDs.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.param_parse_err`

Syntax

```
PROCEDURE sort_folder (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ,
    sort_criteria IN MAIL_SORT_CRITERIA,
    message_uid_list OUT DBMS_SQL.NUMBER_TABLE
);
PROCEDURE sort_folder (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ,
    sort_criteria IN MAIL_SORT_CRITERIA,
    message_list OUT MAIL_MESSAGE_LIST);
};
```

Parameters

Table 1–38 SORT_FOLDER Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_obj`	The `MAIL_FOLDER_OBJ` representing the folder
`sort_criteria`	A list of sort criteria Values are: ■ Subject ■ Cc ■ From ■ Date ■ Internal_date ■ Size
`message_list`	An ordered list of message objects
`message_uid_list`	An ordered list of message UIDs

SEARCH_FOLDER Procedure

This procedure searches the folder and returns a list of message objects that meet the specified criteria. The format of the search criteria is the same as the format in the IMAP4 protocol [RFC 2060], except when specifying to search within a set of messages, the set is passed in as a parameter.

Examples:

- "subject test" (to search for messages with subject containing "test")
- "not from abc@domain.com subject deal" (to search for messages not from "abc@domain.com" and subject containing the word "deal")
- "or since 22-Oct-2001 larger 50000" (to search for message received on or after 10/22/2001 or messages with size that is greater than 50k).

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.param_parse_err`

Syntax

```
PROCEDURE search_folder (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ,
    search_criteria IN VARCHAR2,
    message _list OUT MAIL_MESSAGE_LIST
);
PROCEDURE search_folder (
    session_id IN NUMBER,
    folder_obj IN MAIL_FOLDER_OBJ,
    search_criteria IN VARCHAR2,
    in_message_list IN MAIL_MESSAGE_LIST,
    message_list OUT MAIL_MESSAGE_LIST
);
```

Parameters

Table 1–39 SEARCH_FOLDER Parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`folder_obj`	The `MAIL_FOLDER_OBJ` representing the folder
`search_criteria`	A list of search criterion per IMAP4 standard
`in_message_list`	A list of message objects to search from
`message_list`	The list of message objects that meets the search criteria

MAIL_MESSAGE Package

The MAIL_MESSAGE package provides message retrieval, message composition, and Oracle Text related functionality. The validity of the session is checked before performing any operation, except for composing send operations. Users can only compose one message at a time, and a `MSG_COMPOSE_LIMIT_ERR` is thrown if a violation occurs.

The MAIL_MESSAGE package contains the following procedures:

- GET_MESSAGE_OBJ Procedure
- GET_INCLUDED_MESSAGE Procedure
- GET_HEADER Procedure
- GET_HEADERS Procedure
- GET_CONTENT_TYPE Procedure
- GET_REPLY_TO Procedure
- GET_SENT_DATE Procedure
- GET_SUBJECT Procedure
- GET_FROM Procedure
- GET_MESSAGEID Procedure
- GET_CONTENTID Procedure
- GET_CONTENTLANG Procedure
- GET_CONTENTMD5 Procedure
- GET_CHARSET Procedure
- GET_CONTENTDISP Procedure
- GET_ENCODING Procedure
- GET_CONTENT_FILENAME Procedure
- GET_MSG_SIZE Procedure
- GET_RCVD_DATE Procedure
- GET_BODYPART_SIZE Procedure
- GET_CONTENT_LINECOUNT Procedure
- GET_MULTIPART_BODYPARTS Procedure
- GET_MSG Procedure
- GET_MSG_BODY Procedure
- GET_BODYPART_CONTENT Procedure
- GET_MSG_FLAGS Procedure
- SET_MSG_FLAGS Procedure
- GET_AUTH_INFO Procedure
- SET_MSG_USRFLAGS Procedure
- GET_MSG_COMMENT Procedure

GET_MESSAGE_OBJ Procedure

This procedure returns a message object given the message UID in the current open folder.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get_message_obj (
    session_id  IN NUMBER,
    message_uid IN NUMBER,
    message_obj OUT MAIL_MESSAGE_OBJ);
```

Parameters

Table 1–40 GET_MESSAGE_OBJ Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_uid`	The message UID
`message_obj`	The `MAIL_MESSAGE_OBJ` returned

GET_INCLUDED_MESSAGE Procedure

This procedure returns a message object representing the included message. The message type of the specified message or body-part object must be of "message" type; otherwise the MAIL_ERRORS.PARAM_PARSE_ERR exception is thrown.

Throws Exception

mail_errors.unauthenticated_err

mail_errors.param_parse_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_included_message (
    session_id  IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    message_obj OUT MAIL_MESSAGE_OBJ);
PROCEDURE get_included_message (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    message_obj  IN MAIL_MESSAGE_OBJ,
);
```

Parameters

Table 1–41 GET_INCLUDED_MESSAGE Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object with message Content-Type
bodypart_obj	The body-part object with message Content-Type
incl_message_obj	The MAIL_MESSAGE_OBJ returned

GET_HEADER Procedure

This procedure returns the corresponding header value given in the header prompt.

If a message object is passed in, the header value refers to the top-level message header.

If a body-part object is passed in, the header value refers to that specific body-part header.

When the header prompt is not found in the headers, a null value is returned. When the returned value is a VARCHAR2 type, it is truncated to a maximum length of 2000. If there are multiple headers with the same prompt, the returned value is the first header. When the returned value is a MAIL_HEADER_LIST object list, all headers with the specified prompt name are returned.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_header (
    session_id    IN NUMBER,
    message_obj   IN MAIL_MESSAGE_OBJ,
    header_prompt IN VARCHAR2,
    header_value  OUT VARCHAR2);
PROCEDURE get_header (
    session_id    IN NUMBER,
    bodypart_obj  IN MAIL_BODYPART_OBJ,
    header_prompt IN VARCHAR2,
    header_value  OUT VARCHAR2);
PROCEDURE get_header (
    session_id    IN NUMBER,
    message_obj   IN MAIL_MESSAGE_OBJ,
    header_list   OUT MAIL_HEADER_LIST);
PROCEDURE get_header (
    session_id    IN NUMBER,
    bodypart_obj  IN MAIL_BODYPART_OBJ,
    header_list   OUT MAIL_HEADER_LIST);
```

Parameters

Table 1–42 GET_HEADER Parameters

Parameters	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
bodypart_obj	The body-part object
header_prompt	The message header
header_value	The corresponding header value
header_list	A list of header objects with the specified prompt

GET_HEADERS Procedure

This procedure returns all of the header values of the given message part. If a message object is passed in, the header value refers to the top-level message header.

If a body-part object is passed in, the header value is referred to that specific body-part header.

When the returned value is a `dbms_sql.varchar2_table` type, all header values are truncated to 2000 in length if the length of the value is longer than 2000. When the headers are returned in a `MAIL_HEADER_LIST` object list, each name and value pair is represented by a `MAIL_HEADER_OBJ` object.

Throws Exception

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

Syntax

```
PROCEDURE get_headers (
    session_id     IN NUMBER,
    message_obj    IN MAIL_MESSAGE_OBJ,
    header_prompts IN dbms_sql.varchar2_table,
    header_values  OUT dbms_sql.varchar2_table);
PROCEDURE get_headers (
    session_id     IN NUMBER,
    bodypart_obj   IN MAIL_BODYPART_OBJ,
    header_prompts IN dbms_sql.varchar2_table,
    header_values  OUT dbms_sql.varchar2_table);
PROCEDURE get_headers (
    session_id     IN NUMBER,
    message_obj    IN MAIL_MESSAGE_OBJ,
    header_list    OUT MAIL_HEADER_LIST);
PROCEDURE get_headers (
    session_id     IN NUMBER,
    bodypart_obj   IN MAIL_BODYPART_OBJ,
    header_list    OUT MAIL_HEADER_LIST);
```

Parameters

Table 1–43 GET_HEADERS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`header_prompts`	An array of header names belonging to this part
`header_values`	An array of corresponding header values
`header_list`	A list of header objects belonging to this part

GET_CONTENT_TYPE Procedure

This procedure is used to obtain the Content-Type header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

Syntax

```
PROCEDURE get_content_type (
    session_id    IN NUMBER,
    message_obj   IN MAIL_MESSAGE_OBJ,
    content_type  OUT VARCHAR2);
PROCEDURE get_content_type (
    session_id    IN NUMBER,
    bodypart_obj  IN MAIL_BODYPART_OBJ,
    content_type  OUT VARCHAR2);
```

Parameters

Table 1–44 GET_CONTENT_TYPE Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`content_type`	The Content-Type header value

GET_REPLY_TO Procedure

This procedure is used to obtain the Reply-To header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exception

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_reply_to (
    session_id  IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    replyTo_str OUT VARCHAR2);
```

Parameters

Table 1–45 GET_REPLY_TO Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
replyTo_str	The Reply-To header value

GET_SENT_DATE Procedure

This procedure is used to obtain the Date header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_sent_date (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    sent_date OUT VARCHAR2);
```

Parameters

Table 1–46 GET_SENT_DATE Parameters

Parameters	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
sent_date	The Date header value

GET_SUBJECT Procedure

This procedure is used to obtain the Subject header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_subject (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    subject_str OUT VARCHAR2);
```

Parameters

Table 1–47 GET_SUBJECT Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
subject_str	The subject header value

GET_FROM Procedure

This procedure is used to obtain the From header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_from (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    from_str OUT VARCHAR2);
```

Parameters

Table 1–48 GET_FROM Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
from_str	The From header value

GET_MESSAGEID Procedure

This procedure is used to obtain the Message-ID header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exception

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_messageID (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    messageID_str OUT VARCHAR2);
```

Parameters

Table 1–49 GET_MESSAGEID Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
messageID_str	The Message-ID header value

GET_CONTENTID Procedure

This procedure is used to obtain the Content-ID header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_contentID (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    contentID_str OUT VARCHAR2);
```

Parameters

Table 1–50 GET_CONTENTID Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
bodypart_obj	The body-part object
contentID_str	The Content-ID header value

GET_CONTENTLANG Procedure

This procedure is used to obtain the Content-Language header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

Syntax

```
PROCEDURE get_contentLang (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    language OUT VARCHAR2);
```

Parameters

Table 1–51 GET_CONTENTLANG Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`bodypart_obj`	The body-part object
`language`	The Content-Language header value

GET_CONTENTMD5 Procedure

This procedure is used to obtain the Content-MD5 header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_contentMD5 (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    md5          OUT VARCHAR2);
```

Parameters

Table 1–52 GET_CONTENTMD5 Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
bodypart_obj	The body-part object
md5	The Content-MD5 header value

GET_CHARSET Procedure

This procedure is used to obtain the Content-Type header value, and extract the CHARSET attribute value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_charset (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    charset      OUT VARCHAR2);
```

Parameters

Table 1–53 GET_CHARSET Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
bodypart_obj	The body-part object
charset	The character set attribute value

GET_CONTENTDISP Procedure

This procedure is used to obtain the Content-Disposition header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_contentDisp (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    disposition  OUT VARCHAR2);
```

Parameters

Table 1–54 GET_CONTENTDISP Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
bodypart_obj	The body-part object
disposition	The Content-Disposition header value

GET_ENCODING Procedure

This procedure is used to obtain the Content-Transfer-Encoding header value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_encoding (
    session_id  IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    encoding    OUT VARCHAR2);
PROCEDURE get_encoding (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    encoding     OUT VARCHAR2);
```

Parameters

Table 1–55 GET_ENCODING Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
bodypart_obj	The body-part object
encoding	The Content-Transfer-Encoding header value

GET_CONTENT_FILENAME Procedure

This procedure is used to obtain the Content-Disposition header value and extract the filename attribute value. It internally calls the GET_HEADER procedure with the specific header prompt.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.bad_message_var

mail_errors.bad_msgpart_var

Syntax

```
PROCEDURE get_content_filename (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    filename     OUT VARCHAR2);
```

Parameters

Table 1–56 GET_CONTENT_FILENAME Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
bodypart_obj	The body-part object
filename	The filename attribute value

GET_MSG_SIZE Procedure

This procedure returns the message size.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE get_msg_id (
    session_id   IN NUMBER,
    message_obj  IN MAIL_MESSAGE_OBJ,
    message_size OUT NUMBER);
```

Parameters

Table 1–57 GET_MSG_SIZE Parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`message_size`	The message size

GET_RCVD_DATE Procedure

This procedure is used to obtain the time the message is received at the mail store.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

Syntax

```
PROCEDURE get_rcvd_date (
    session_id  IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    date_format IN VARCHAR2,
    date_str    OUT VARCHAR2);
PROCEDURE get_rcvd_date (
    session_id    IN NUMBER,
    message_obj   IN MAIL_MESSAGE_OBJ,
    received_date OUT DATE);
```

Parameters

Table 1–58 GET_RCVD_DATE Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`date_format`	The date to string format
`date_str`	The received date in the string format specified
`received_date`	The received date in Oracle date format

GET_BODYPART_SIZE Procedure

This procedure returns the size of the body-part.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

Syntax

```
PROCEDURE get_bodypart_size (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    bodypart_size OUT NUMBER);
```

Parameters

Table 1–59 GET_BODYPART_SIZE Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`bodypart_obj`	The body-part object
`bodypart_size`	The body-part size

GET_CONTENT_LINECOUNT Procedure

This procedure returns the line count of the body-part.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

Syntax

```
PROCEDURE get_content_linecount (
    session_id   IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    line_count   OUT NUMBER);
```

Parameters

Table 1–60 GET_CONTENT_LINECOUNT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`bodypart_obj`	The body-part object
`line_count`	The total number of lines in the body-part

GET_MULTIPART_BODYPARTS Procedure

This procedure returns a list of body-parts that belong to the specified multipart message or body-part. If the message or body-part object passed in is not of a multipart MIME type, a PARAM_PARSE_ERR exception is raised.

Throws Exceptions

mail_errors.unauthenticated_err

mail_errors.param_parse_err

mail_errors.bad_message_var

Syntax

```
PROCEDURE get_multipart_bodyparts (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    bodypart_list OUT MAIL_BODYPART_LIST);
PROCEDURE get_multipart_bodyparts (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    bodypart_list OUT MAIL_BODYPART_LIST);
```

Parameters

Table 1–61 GET_MULTIPART_BODYPARTS Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
bodypart_obj	The body-part object
bodypart_list	A list of body-parts

GET_MSG Procedure

This procedure returns a BLOB locator to the entire encoded message. Storage does not needs to be allocated beforehand.

Throws Exceptions

`mail_errors.unauthenticated_err`

Syntax

```
PROCEDURE get_msg (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    message_source OUT BLOB);
```

Parameters

Table 1–62 GET_MSG Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`message_source`	The whole message content in its original encoded form

GET_MSG_BODY Procedure

This procedure copies the message body into the specified BLOB locator. The locator must have enough storage for the data. If the message is not a simple MIME type, no data is returned. If the message body's Content-Transfer-Encoding header specifies that the data is encoded, using base64 or quoted-printable encodings, the content is decoded before returning.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

Syntax

```
PROCEDURE get_msg_body (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    content OUT BLOB);
```

Parameters

Table 1–63 GET_MSG_BODY Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`content`	The entire decoded message content

GET_BODYPART_CONTENT Procedure

This procedure copies the content of the body-part into the specified BLOB locator. If the body-part object is not a simple MIME type, no data is returned. If the body-part's Content-Transfer-Encoding header specifies that the data is encoded, using base64 or quoted-printable encodings, the content is decoded before returning.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.bad_message_var`

Syntax

```
PROCEDURE get_bodypart_content (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    content OUT BLOB);
```

Parameters

Table 1–64 GET_BODYPART_CONTENT

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`bodypart_obj`	The body-part object
`content`	The entire decoded message content

GET_MSG_FLAGS Procedure

This procedure returns the flags associated with a message.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get_msg_flags (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    message_flags OUT NUMBER);
PROCEDURE get_msg_flags (
    session_id        IN NUMBER,
    message_obj       IN MAIL_MESSAGE_OBJ,
    message_flags     OUT NUMBER,
    message_usrflags  OUT VARCHAR2
);
```

Parameters

Table 1–65 GET_MSG_FLAGS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	A message object
`message_flags`	The message flags are a set of well-defined bit-wise values. The bits are defined in the MAIL_MESSAGE package specification.
`message_usrflags`	User defined message flag string. It contains a list of user defined flag strings separated by spaces.

SET_MSG_FLAGS Procedure

This procedure sets and unsets the message flags for the specified message object.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE set_msg_flags (
    session_id    IN NUMBER,
    message_obj   IN MAIL_MESSAGE_OBJ,
    message_flags IN NUMBER,
    set_flag      IN BOOLEAN);
```

Parameters

Table 1–66 SET_MSG_FLAGS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	A message object
`message_flags`	The message flags are a set of well-defined bit-wise values. The bits are defined in the MAIL_MESSAGE package specification.
`set_flag`	If true, sets the specified flags, otherwise, unsets the specified flags

GET_AUTH_INFO Procedure

This procedure returns authenticated user information, if available. The authenticated user information is stored when a user authenticates before sending an e-mail.

Throws Exceptions

```
mail_errors.unauthenticated_err
```

Syntax

```
PROCEDURE get_auth_info (
    session_id  IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    auth_info   OUT VARCHAR2);
```

Parameters

Table 1–67 GET_AUTH_INFO Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
auth_info	The authenticated user information

GET_MSG_COMMENT Procedure

A comment is a name-value pair. If the specified comment name is set for the message, this procedure returns the comment value.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE get_msg_comment (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    comment_name     IN VARCHAR2,
    comment_value    OUT VARCHAR2
);
```

Parameters

Table 1–68 GET_MSG_COMMENT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`comment_name`	The name of the comment.
`comment_value`	The returned value of the comment.

SET_MSG_COMMENT Procedure

A comment is a name-value pair. If the specified comment name is set for the message, this procedure returns the comment value.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE set_msg_comment (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    comment_name     IN VARCHAR2,
    comment_value    IN VARCHAR2
);
```

Parameters

Table 1–69 SET_MSG_COMMENT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`comment_name`	The name of the comment.
`comment_value`	The new value of the comment.

SET_MSG_COMMENTS Procedure

This procedure sets a list of comments (name-value pairs) for the message.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE set_msg_comments (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    comment_names    IN dbms_sql.varchar2_table,
    comment_values   IN dbms_sql.varchar2_table
);
```

Parameters

Table 1–70 SET_MSG_COMMENTS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`comment_name`	The names of the comment.
`comment_value`	The new values of the comments.

REMOVE_MSG_COMMENT Procedure

This procedure removes a message comment (a name-value pair) given the comment name.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE remove_msg_comment (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    comment_name     IN VARCHAR2,
);
```

Parameters

Table 1–71 REMOVE_MSG_COMMENT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`comment_name`	The name of the comment to remove

REMOVE_MSG_COMMENTS Procedure

This procedure removes a list of message comments (name-value pairs) given the comment names.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.folder_closed_err`

Syntax

```
PROCEDURE remove_msg_comments (
    session_id       IN NUMBER,
    message_obj      IN MAIL_MESSAGE_OBJ,
    comment_names    IN dbms_sql.varchar2_table,
);
```

Parameters

Table 1–72 REMOVE_MSG_COMMENTS Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`comment_name`	The names of the comments to delete

COMPOSE_MESSAGE Procedure

This procedure initializes a message composition. There can be at most one message in composition at any given time.

Throws Exceptions

`mail_errors.msg_compose_limit_err`

`mail_errors.param_parse_err`

Syntax

```
PROCEDURE compose_message (
    message_obj OUT MAIL_MESSAGE_OBJ);
```

Parameters

Table 1–73 COMPOSE_MESSAGE Parameters

Parameter	Description
`message_obj`	A message object

SET_MSGHEADER Procedure

This procedure sets a list of common message headers. If null is specified, the header is not included. If the sent date is null, it is set to the current time.

Throws Exceptions

`mail_errors.msg_compose_limit_err`

Syntax

```
PROCEDURE set_msgheader (
    message_obj IN MAIL_MESSAGE_OBJ,
    to_str IN VARCHAR2,
    from_str IN VARCHAR2,
    cc_str IN VARCHAR2 DEFAULT null,
    replyto_str IN VARCHAR2 DEFAULT null,
    sent_date IN DATE DEFAULT null,
    subject_str IN VARCHAR2 DEFAULT null,
    mime_version IN VARCHAR2 DEFAULT '1.0',
    content_type IN VARCHAR2 DEFAULT 'text/plain',
    charset IN VARCHAR2 DEFAULT 'us-ascii',
    encoding IN VARCHAR2 DEFAULT '8bit');
```

Parameters

Table 1–74 SET_MSGHEADER Parameters

Parameter	Description
`message_obj`	A message object
`to_str`	The To RFC822 header
`from_str`	The From RFC822 header
`cc_str`	The Cc RFC822 header
`replyto_str`	The Reply-to RFC822 header
`sent_date`	The Date RFC822 header
`subject_str`	The Subject RFC822 header
`mime_version`	The MIME-Version RFC822 header
`content_type`	The Content-Type RFC822 header
`charset`	The Content-Type RFC822 header charset attribute
`encoding`	The Content-Transfer-Encoding RFC822 header

SET_BPHEADER Procedure

This procedure sets a list of common body-part headers. If null is specified, the header is not included. All header values are limited to 2000 in length; if it exceeds the limit, a MAIL_ERRORS.PARAM_PARSE_ERR is thrown.

Throws Exceptions

mail_errors.msg_compose_limit_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE set_bpheader (
    bodypart_obj IN MAIL_BODYPART_OBJ,
    content_type IN VARCHAR2 DEFAULT 'text/plain',
    charset      IN VARCHAR2 DEFAULT 'us-ascii',
    encoding     IN VARCHAR2 DEFAULT '8bit',
    contentID    IN VARCHAR2 DEFAULT null,
    language     IN VARCHAR2 DEFAULT null,
    contentMD5   IN VARCHAR2 DEFAULT null,
    description  IN VARCHAR2 DEFAULT null,
    disposition  IN VARCHAR2 DEFAULT 'inline',
    filename     IN VARCHAR2 DEFAULT null);
```

Parameters

Table 1–75 SET_BPHEADER Parameters

Parameter	Description
message_obj	A message object
content_type	The Content-Type header
charset	The Content-Type header charset attribute
encoding	The Content-Transfer-Encoding header
contentID	The Content-ID header
language	The Content-Language header
contentMD5	The Content-MD5 header
description	The Content-Description header
disposition	The Content-Disposition header
filename	The Content-Disposition header filename attribute

SET_HEADER Procedure

This procedure sets the header value, given the header prompt. This does not override any previous headers; it adds to them. All header values are limited to 2000 in length; if it exceeds the limit, a `MAIL_ERRORS.PARAM_PARSE_ERR` is thrown.

Throws Exceptions

`mail_errors.msg_compose_limit_err`

`mail_errors.param_parse_err`

Syntax

```
PROCEDURE set_header (
    message_obj IN MAIL_MESSAGE_OBJ,
    header_prompt IN VARCHAR2,
    header_value IN VARCHAR2);
PROCEDURE set_header (
    bodypart_obj IN MAIL_BODYPART_OBJ,
    header_prompt IN VARCHAR2,
    header_value IN VARCHAR2);
```

Parameters

Table 1–76 SET_HEADER Parameters

Parameter	Description
`message_obj`	The message object
`bodypart_obj`	The body-part object
`header_prompt`	The message or body-part header
`header_value`	The corresponding header value

ADD_BODYPART Procedure

This procedure adds a child body-part to the specified parent message or body-part of Content-Type multipart. If the parent message or body-part object does not have "message" content-type, a PARAM_PARSE_ERR exception is thrown.

Throws Exceptions

mail_errors.msg_compose_limit_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE add_bodypart (
    parent_message_obj IN MAIL_MESSAGE_OBJ,
    bodypart_obj OUT MAIL_BODYPART_OBJ);
PROCEDURE add_bodypart (
    parent_bodypart_obj IN MAIL_BODYPART_OBJ,
    bodypart_obj OUT MAIL_BODYPART_OBJ);
```

Parameters

Table 1–77 ADD_BODYPART Parameters

Parameter	Description
parent_message_obj	The parent message object
parent_bodypart_obj	The parent body-part object
bodypart_obj	The new child body-part object returned

ADD_INCLMSG_BODYPART Procedure

This procedure adds a new included message to the specified parent message or body-part of Content-Type message. If the parent message or body-part object does not have "message" content-type, a PARAM_PARSE_ERR exception is thrown.

Throws Exceptions

mail_errors.msg_compose_limit_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE add_inclmsg_bodypart (
    parent_message_obj IN MAIL_MESSAGE_OBJ,
    message_obj        OUT MAIL_MESSAGE_OBJ);
PROCEDURE add_inclmsg_bodypart (
    parent_bodypart_obj IN MAIL_BODYPART_OBJ,
    message_obj         OUT MAIL_MESSAGE_OBJ);
```

Parameters

Table 1–78 ADD_INCLMSG_BODYPART Parameters

Parameter	Description
parent_message_obj	The parent message object
parent_bodypart_obj	The parent body-part object
message_obj	The new included message object returned

SET_INCLMSG_BODYPART Procedure

This procedure sets an included message to the specified parent message or body-part of Content-Type message. The included message must already exist in the mail store. If the parent message or body-part object does not have "message" content-type, a PARAM_PARSE_ERR exception is thrown.

Throws Exceptions

mail_errors.msg_compose_limit_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE set_inclmsg_bodypart (
    parent_message_obj IN MAIL_MESSAGE_OBJ,
    message_obj        IN MAIL_MESSAGE_OBJ);
PROCEDURE set_inclmsg_bodypart (
    parent_bodypart_obj IN MAIL_BODYPART_OBJ,
    message_obj         IN MAIL_MESSAGE_OBJ);
```

Parameters

Table 1–79 SET_INCLMSG_BODYPART

Parameter	Description
parent_message_obj	The parent message object
parent_bodypart_obj	The parent body-part object
message_obj	An existing message in mail store

SET_CONTENT Procedure

This procedure sets the message or body-part content for the message in the composition. If the message or body-part is not a simple MIME type, a PARAM_PARSE_ERR is thrown. This procedure can be called multiple times. The data is connected together. The data should be in decoded form. When the composed message is sent or appended, the data is encoded according to the Content-Transfer-Encoding header specified for this part of the data.

Throws Exceptions

mail_errors.msg_compose_limit_err

mail_errors.param_parse_err

Syntax

```
PROCEDURE set_content (
    message_obj IN MAIL_MESSAGE_OBJ,
    content IN RAW);
PROCEDURE set_content (
    message_obj IN MAIL_MESSAGE_OBJ,
    content IN BLOB);
PROCEDURE set_content (
    bodypart_obj IN MAIL_BODYPART_OBJ,
    content IN RAW);
PROCEDURE set_content (
    bodypart_obj IN MAIL_BODYPART_OBJ,
    content IN BLOB);
```

Parameters

Table 1–80 SET_CONTENT Parameters

Parameter	Description
message_obj	The message object
bodypart_obj	The body-part object
content	The message or body-part content

SEND_MESSAGE Procedure

This procedure sends the message currently in composition. The message can also be sent encrypted, signed, or both.

Throws Exceptions

`mail_errors.msg_compose_limit_err`

`mail_errors.smime_err`

Syntax

```
PROCEDURE send_message (
    message_obj IN MAIL_MESSAGE_OBJ);
PROCEDURE send_message (
    message_obj       IN MAIL_MESSAGE_OBJ,
    ceritificate      IN RAW,
    private_key       IN RAW,
    recipients        IN MAIL_MESSAGE.RAW_TABLE,
    inclOrigCert      IN BOOLEAN,
    inclOrigAsRecip   IN BOOLEAN,
    digest_algorithm  IN BINARY_INTEGER,
    sign_algorithm    IN BINARY_INTEGER,
    encrypt_algorithm IN BINARY_INTEGER,
    send_option       IN NUMBER);
```

Parameters

Table 1–81 SEND_MESSAGE Parameters

Parameter	Description
`message_obj`	The message object
`certificate`	The user's certificate
`private_key`	The user's private key
`recipients`	The recipient's certificates
`inclOrigCert`	Specifies whether to include the user's certificate when encrypting, signing, or both
`inclOrigAsRecip`	Specifies whether to include the user's certificate in the recipient list
`digest_algorithm`	The algorithm to use for generating the digest when signing. It is not used if the message is only being encrypted Values are: ■ `MAIL_MESSAGE.GC_MD5` ■ `MAIL_MESSAGE.GC_SHA1`

APPEND_MESSAGE Procedure

This procedure appends the current message composition to the specified folder. The user must be authenticated and the folder must belong to the user.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.msg_compose_limit_err`

Syntax

```
PROCEDURE append_message (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    folder_name IN VARCHAR2,
    received_date IN DATE DEFAUL null,
    message_flags IN NUMBER DEFAULT 0);
PROCEDURE append_message (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    folder_obj IN MAIL_FOLDER_OBJ,
    received_date IN DATE DEFAUL null,
    message_flags IN NUMBER DEFAULT 0);
```

Parameters

Table 1–82 APPEND_MESSAGE Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`folder_name`	The folder to which the message is appended
`folder_obj`	The folder to which the message is appended
`received_date`	The message's received date
`message_flags`	A list of message flags corresponding to the list of requested messages. Flag values are defined in the MAIL_MESSAGE package specification. Values are: ■ `MAIL_MESSAGE.GC_SEEN_FLAG` ■ `MAIL_MESSAGE.GC_FLAGGED_FLAG` ■ `MAIL_MESSAGE.GC_ANSWERED_FLAG` ■ `MAIL_MESSAGE.GC_DELETED_FLAG` ■ `MAIL_MESSAGE.GC_DRAFT_FLAG`

ABORT_MESSAGE Procedure

This procedure aborts the current message in composition, and cleans up current message information. This procedure should be called for any errors during message composition.

Syntax

```
PROCEDURE abort_message;
```

ENCODE_HDRTEXT Function

This utility function can be used to encode a non-ASCII string into an RFC2047 formatted header string. If the string passed in is ASCII, the function returns the string as is.

Throws Exceptions

`mail_errors.msg_compose_limit_err`

Syntax

```
FUNCTION encode_hdrtext (
    p_string IN VARCHAR2
) RETURNS VARCHAR2;
```

Parameters

Table 1–83 ENCODE_HDRTEXT Parameters

Parameter	Description
`p_string`	A header string

DECRYPT_MESSAGE Procedure

This procedure decrypts a S/MIME message and returns a list of body-parts that belongs to the encrypted part.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.smime_err`

Syntax

```
PROCEDURE decrypt_message (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    ceritificate IN RAW,
    private_key IN RAW,
    bodypart_list OUT MAIL_BODYPART_LIST
);
PROCEDURE decrypt_message (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_MESSAGE_OBJ,
    ceritificate IN RAW,
    private_key IN RAW,
    bodypart_list OUT MAIL_BODYPART_LIST
);
```

Parameters

Table 1–84 DECRYPT_MESSAGE Parameters

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	The message object
bodypart_obj	The body-part message
certificate	The user's certificate
private_key	The user's private key
bodypart_list	The decrypted body-part list

VERIFY_MESSAGE Procedure

This procedure verifies a digitally signed message.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.smime_err`

Syntax

```
PROCEDURE verify_message (
    session_id IN NUMBER,
    ceritificate IN RAW,
    private_key IN RAW,
    original_content IN BLOB,
    certificate_list IN ES_CERT_LIST,
    signature IN BLOB,
    returned_content OUT BLOB
);
```

Parameters

Table 1–85 VERIFY_MESSAGE Parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`certificate`	The user's certificate
`private_key`	The user's private key
`original_content`	The message content
`certificate_list`	The certificate list
`signature`	The digitally signed signature

GET_THEMES Procedure

This procedure retrieves the theme for the message or body-part. If a message object is specified, the entire message is processed. If a body-part object is specified, the part must be a simple type and not contain any other body-parts.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.sql_err`

`mail_errors.imt_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

`mail_errors.no_binary_err`

Syntax

```
PROCEDURE get_themes (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    flags IN INTEGER,
    incl_binary_parts IN BOOLEAN,
    theme_buffer OUT ES_OT_API.THEME_TABLE);
PROCEDURE get_themes (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    flags IN INTEGER,
    incl_binary_parts IN BOOLEAN,
    theme_buffer OUT ES_OT_API.THEME_TABLE);
```

Parameters

Table 1–86 GET_THEMES Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`flags`	Currently not used
`incl_binary_parts`	If false, the non-text part is ignored
`theme_buffer`	The theme buffer

GET_HIGHLIGHT Procedure

This procedure retrieves the highlights for the message or body-part. If a message object is specified, the entire message is processed. If a body-part object is specified, the part must be a simple type and not contain any other body-parts.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.sql_err`

`mail_erros.imt_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

`mail_errors.no_binary_err`

Syntax

```
PROCEDURE get_highlight (
    session_id        IN NUMBER,
    message_obj       IN MAIL_MESSAGE_OBJ,
    flags             IN INTEGER,
    text_query        IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    highlight_buffer  OUT ES_OT_API.HIGHLIGHT_TABLE);

PROCEDURE get_highlight (
    session_id        IN NUMBER,
    bodypart_obj      IN MAIL_BODYPART_OBJ,
    flags             IN INTEGER,
    text_query        IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    highlight_buffer  OUT ES_OT_API.HIGHLIGHT_TABLE);
```

Parameters

Table 1–87 GET_HIGHLIGHT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`flags`	The returned buffer format Values are: ■ `MAIL_MESSAGE.GC_TEXT_FORMAT` ■ `MAIL_MESSAGE_GC_HTML_FORMAT`
`text_query`	The string you want to query
`incl_binary_parts`	If false, the non-text part is ignored
`highlight_buffer`	The highlight buffer

GET_MARKUPTEXT Procedure

This procedure retrieves the mark-up text for the message or body-part. If a message object is specified, the entire message is processed. If a body-part object is specified, the part must be a simple type and not contain any other body-parts.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.sql_err`

`mail_erros.imt_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

`mail_errors.no_markup`

`mail_errors.no_binary_err`

Syntax

```
PROCEDURE get_markuptext (
    session_id        IN NUMBER,
    message_obj       IN MAIL_MESSAGE_OBJ,
    flags             IN INTEGER,
    text_query        IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    tag_set           IN VARCHAR2 DEFAULT 'TEXT_DEFUALT',
    start_tag         IN VARCHAR2 DEFAULT NULL,
    end_tag           IN VARCHAR2 DEFAULT NULL,
    prev_tag          IN VARCHAR2 DEFAULT NULL,
    next_tag          IN VARCHAR2 DEFAULT NULL,
    buffer            OUT CLOB);
PROCEDURE get_markuptext (
    session_id        IN NUMBER,
    bodypart_obj      IN MAIL_BODYPART_OBJ,
    flags             IN INTEGER,
    text_query        IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    tag_set           IN VARCHAR2 DEFAULT 'TEXT_DEFUALT',
    start_tag         IN VARCHAR2 DEFAULT NULL,
    end_tag           IN VARCHAR2 DEFAULT NULL,
    prev_tag          IN VARCHAR2 DEFAULT NULL,
    next_tag          IN VARCHAR2 DEFAULT NULL,
    buffer            OUT CLOB);
```

Parameters

Table 1–88 GET_MARKUPTEXT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object

Table 1–88 (Cont.) GET_MARKUPTEXT Parameters

Parameter	Description
`flags`	The returned buffer format. Values are: ■ MAIL_MESSAGE.GC_TEXT_FORMAT ■ MAIL_MESSAGE_GC_HTML_FORMAT
`text_query`	The string you want to query
`incl_binary_parts`	If false, non-text part is ignored
`tag_set`	Refer to Oracle Text documentation
`start_tag`	Refer to Oracle Text documentation
`end_tag`	Refer to Oracle Text documentation
`prev_tag`	Refer to Oracle Text documentation
`next_tag`	Refer to Oracle Text documentation
`buffer`	The mark-up text buffer

GET_FILTERED_TEXT Procedure

This procedure retrieves the filtered text for the message or body-part. If a message object is specified, the entire message is processed. If a body-part object is specified, the part must be a simple type and not contain any other body-parts.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.sql_err`

`mail_errors.imt_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

`mail_errors.no_binary_err`

Syntax

```
PROCEDURE get_filtered_text (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    flags IN INTEGER,
    incl_binary_parts IN BOOLEAN,
    buffer OUT CLOB);
PROCEDURE get_filtered_text (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    flags IN INTEGER,
    incl_binary_parts IN BOOLEAN,
    buffer OUT CLOB);
```

Parameters

Table 1–89 GET_FILTERED_TEXT Parameters

Parameter	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`flags`	The returned buffer format Values are: ■ `MAIL_MESSAGE.GC_TEXT_FORMAT` ■ `MAIL_MESSAGE_GC_HTML_FORMAT`
`incl_binary_parts`	If false, the non-text part is ignored
`buffer`	The filtered text buffer

GET_TOKENS Procedure

This procedure retrieves the tokens for the message or body-part. If a message object is specified, the entire message is processed. If a body-part object is specified, the part must be a simple type and not contain any other body-parts.

Throws Exceptions

`mail_errors.unauthenticated_err`

`mail_errors.sql_err`

`mail_errors.imt_err`

`mail_errors.bad_message_var`

`mail_errors.bad_msgpart_var`

`mail_errors.no_binary_err`

Syntax

```
PROCEDURE get_tokens (
    session_id IN NUMBER,
    message_obj IN MAIL_MESSAGE_OBJ,
    language IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    token_buffer OUT ES_OT_API.TOKEN_TABLE);
PROCEDURE get_tokens (
    session_id IN NUMBER,
    bodypart_obj IN MAIL_BODYPART_OBJ,
    language IN VARCHAR2,
    incl_binary_parts IN BOOLEAN,
    token_buffer OUT ES_OT_API.TOKEN_TABLE);
```

Parameters

Table 1–90 GET_TOKENS Parameters

Parameters	Description
`session_id`	An identifier that represents a user's authenticated session
`message_obj`	The message object
`bodypart_obj`	The body-part object
`language`	The language of the message or body-part data.
`incl_binary_parts`	If false, the non-text part is ignored
`token_buffer`	The token buffer

Exceptions

external_rule_err EXCEPTION

Error No: 20001

Message: Error executing external rule

Cause: A rule defined as an external PL/SQL procedure failed during execution

Action: Check the correctness of the external PL/SQL procedure

external_cond_err EXCEPTION

Error No: 20002

Message: External condition failed

Cause: A condition defined as an external PL/SQL function failed during evaluation

Action: Check the correctness of the external PL/SQL function

too_many_rules EXCEPTION

Error No: 20003

Message: Too many rules fired, stop

Cause: Number of rules triggered for a message exceeded the maximum number of rules allowed. The default is 20

Action: Check if the rules involved causes infinite looping. Reduce the number of rules defined. Ask the administrator to increase the maximum number of rules allowed.

sql_err EXCEPTION

Error No: 20101

Message: Some SQL error occured: %s

Cause: An SQL error occured. See the error text returned for more information

Action: Refer to the Oracle9i Database Error Messages guide

imt_err EXCEPTION

Error No: 20102

Message: interMedia Text error: %s

Cause: Internal Oracle Text error occured. See the error text returned for more information

Action: Refer to the Oracle Text documentation

bad_message_var EXCEPTION

Error No: 20103

Message: No such message: %s

Cause: Exception is raised when the message or body-part object passed in is invalid. The message may have been removed by another mail session

Action: Ensure that the message or body-part object passed in is valid

bad_msgpart_var EXCEPTION

Error No: 20104

Message: No such message part: %s

Cause: Exception is raised when the MIME-level of the message or body-part object passed in is invalid

Action: Ensure that the MIME level of the message or body-part object passed in is valid

no_binary_err EXCEPTION

Error No: 20106

Message: No binary part: %s

Cause: Exception is raised when the specified message part is binary, but the API option specifies that binary is false

Action: Set the withbinary parameter to True for Oracle Text

unauthenticated_err EXCEPTION

Error No: 20201

Message: User needs to be authenticated first

Cause: User is not currently authenticated

Action: Call the `mail_session.login()` procedure to authenticate the user

folder_closed_err EXCEPTION

Error No: 20202

Message: Folder needs to be opened first!

Cause: Certain operations require that the folder involved be opened first

Action: Call the `mail_folder.open_folder()` procedure to open the folder first

msg_compose_limit_err EXCEPTION

Error No: 20203

Message: Compose one message at a time!

Cause: Exception is raised if user tries to compose more than one message at a time

Action: Send or append the current message before starting to compose the second message

folder_not_found_err EXCEPTION

Error No: 20204

Message: Folder does not exist: %s

Cause: Exception is raised if user is trying to do an operation on a folder that does not exist on the mail store

Action: Ensure that the folder exists before performing the operation

folder_already_exists_err EXCEPTION

Error No: 20205

Message: Folder already exists: %s

Cause: Exception is raised if user tries to create or rename as a folder name that already exists

Action: Choose a different folder name and retry

operation_not_allowed EXCEPTION

Error No: 20206

Message: Operation not allowed: %s

Cause: Possible causes for this exception are:

- Trying to delete the INBOX folder
- Trying to rename a folder to a descendent older name, such as x/y

Action: Do not try to re-arrange a folder hierarchy with the rename operation. Use the create and delete operations to achieve the desired folder hierarchy

param_parse_err EXCEPTION

Error No: 20208

Message: Param parsing error: %s

Cause: Errors in the parameter passed into the API. Possible causes are:

- The specified message UID does not exist in the current folder
- Trying to send or append a message not currently in composition
- Passed in message or bodypart object with wrong Content-Type value
- Invalid sort criteria
- Unmatched parentheses or quote in search string
- Unsupported search criteria
- Trying to create a folder with null foldername
- Unknown search criteria
- The header value exceeds the 2000 length limit

Action: Correct the parameter passed to the API and try again

internal_err EXCEPTION

Error No: 20209

Message: Internal error: %s

Cause: An internal assertion has failed. Data is in an inconsistent state

Action: Contact Oracle Support

folder_name_err EXCEPTION

Error No: 20210

Message: Bad folder name: %s

Cause: Trying to create or rename a folder under another user's name space

Action: Correct the folder name and try again

login_err EXCEPTION

Error No: 20211

Message: Oracle Internet Directory Login Error: %s

Cause: Exception is raised when an invalid username or password is specified

Action: Check the spelling and try again

folder_type_err EXCEPTION

Error No: 20212

Message: Folder type voilation: %s

Cause: Possible causes for this error include:

- Trying to open a non-selectable folder
- Trying to create a folder where the parent folder does not permit sub-folder creation
- Trying to delete a non-selectable folder that still has sub-folders
- Trying to copy messages to a non-selectable folder

Action: Avoid these types of folder violations

smime_err EXCEPTION

Error No: 20213

Message: S/MIME error: %s

Cause: Some error in calling S/MIME functions. The S/MIME error code reveals more details.

Action: Refer to the S/MIME documentation for the error code descriptions

2

Oracle Mail Java API

This chapter describes the Oracle Mail Java APIs that can be used to create customized clients, integrate applications, and support certain extensions.

Introduction to the Oracle JavaMail API

The Oracle Javamail API (OJMA) Service Provider implements the abstract JavaMail 1.2 API (JMA) provided by Sun Microsystems, Inc., and is designed to integrate directly with the mail store in the Oracle database. This approach does not depend upon a standards-based protocol server and calls PL/SQL APIs in the mail store over JDBC.

JMA provides a set of abstract classes that model a mail system. The API provides a platform independent and protocol independent framework to build Java technology-based mail and messaging applications. The JMA implementations typically go over a standards-based protocol such as IMAP or POP3. The JMA is implemented as a Java platform optional package and is also available as part of the Java 2 platform, Enterprise Edition. More details are available at the following URL:

```
http://java.sun.com/products/javamail
```

Within JMA, a user connects to a message store. The store contains a list of folders, and a user can perform folder and message operations based on the IMAP Protocol (RFC 2060).

In addition to the standard functionality available with JMA operating over an IMAP protocol service provider, the Oracle Javamail API Service Provider supports several extensions to the IMAP protocol and JMA. These enhancements do not interfere with the basic JMA interface, and clients using basic JMA can seamlessly integrate with the Oracle Javamail API Service Provider.

Oracle Javamail API Service Provider supports the following enhancements:

- ACL extension support based on RFC2086, which enable mail users to share their folders with other users or groups or lists
- Administrators can create public folders at the domain level that are available to all the users on that domain.
- Message sorting within a folder, based on the IMAP Sort Extension draft
- Integration with the SMIME toolkit, which enables you to encrypt outgoing e-mail, and decrypt and verify incoming e-mail
- Content-based searching using Oracle Text

- Wireless profiles and filters, which enables you to apply a filter and see only a filtered subset of the messages in the store (useful for PDAs and for wireless access to e-mail)
- Message annotations based on the IMAP ANNOTATE Extension draft, which can be used by users to add comments or annotations to messages in their folders

Also, because the Oracle Javamail API Service provider works directly with the Oracle database, system requirements are reduced since there is no overhead involved in conversing with a standards-based server, such as IMAP or POP3, and tasks such as sorting, lookups, and message sharing are efficiently handled by the Oracle database.

Directory Management API

The directory management API is a set of Java classes that can be used to create, access, and manage various entries, such as mail users, public distribution lists, and private address book entries (contact information and private distribution lists). The entries are stored in Oracle Internet Directory for a given domain.

Directory Components

In Oracle Email, an e-mail system can contain more than one domain. Mail users and public distribution lists exist for a particular domain. A mail user for a given domain is a valid user in that domain who can send and receive e-mail and use all of the exposed e-mail server functionality.

A public distribution list is a mailing list that has its own e-mail ID and contains a group of e-mail IDs or other mailing lists. When an e-mail is sent to a public distribution list, all the members of the list receive the e-mail. A valid mail user can subscribe to any public distribution list.

Private address book entries consist of private contacts and private mailing lists belonging to a particular mail user.

A private distribution list consists of private contact information, such as the contact's phone number, e-mail ID, and address. Users can use the private address book entries from WebMail to send and receive e-mails. These entries are also used by the Calendar application.

Authentication

Before a caller can access any of the directory components, they must be authenticated with the Oracle Internet Directory using the `oracle.mail.OESContext class`. Once authenticated, the `oracle.mail.OESContext` instance representing a trusted session must be passed to all of the directory APIs.

The following example shows how to authenticate an application with the debug option turned off:

```
OESContext oesctx = new OESContext(DirectoryConstants.DS_CALLERTYPE_APP, false);

// Authenticate to the directory
oesctx.authenticate(null, args[0]);
```

Retrieving the Metadata and Validation

Before an entry is created in the directory, the caller needs to retrieve the metadata for that particular entry from the directory. The metadata for a particular entry consists of

the mandatory and optional attributes the caller must set in order to create an entry. It also contains information about all the attributes, such as the syntax, multiplicity of the attributes, and default values for attributes (if any defaults are set in the directory).

When the caller sets the attribute value on the metadata object, validation is performed to ensure that the caller sets the value of an attribute that is present in that particular entry. In UI-based applications using the metadata, the caller can perform any input validations for the data entered.

The following example shows how to retrieve the metadata. It assumes that `ldapobj` is an instance of the `DirectoryObject` class.

```
// Getting the mandatory attributes from the metadata
if (ldapobj.getMandatoryAttribs() != null)
{
 Enumeration enum = ldapobj.getMandatoryAttribs().elements();
 while (enum.hasMoreElements())
 {
 String attr = enum.nextElement().toString(); // Name of the attribute

 // Retrieve the metadata for this attribute
 DirectoryAttributeMetaData mdata = ldapobj.getMetaData(attr);

 // The multiplicity of the attribute returns "SINGLE" or "MULTIPLE"
 String mult = mdata.getMultiplicity();

 // Returns the syntax of the string, "String", "byte", "boolean" or "int"
 String syntax = mdata.getAttributeType();

 // Returns a vector of String values if any default has been set,
 // else returns null
 Vector defaultvals = mdata.getDefaultValues();
 }
}
```

Similarly, the `ldapobj.getOptionalAttribs()` method returns the list of optional attributes and in a similar manner, the optional attributes and the metadata can be retrieved. After retrieving the metadata, the user can set the value of an attribute in the following manner. When creating an entry, the attribute value can be set using the `setAttributeValue` method of the `DirectoryObject` class.

This example sets the value for the telephonenumber attribute to the default values provided `mdata.getDefaultValues()` is not null.

```
ldapobj.setAttributeValue("telephonenumber", mdata.getDefaultValues());

Vector newVals = new Vector();
newVals.add("408 7394050");
newVals.add("650 7394050");
ldapobj.setAttributeValue("telephonenumber", newVals);
```

While modifying an entry, the attribute value can be set using the modifyAttributeValue method of the DirectoryObject class. The caller needs to specify the type of modification.

The permitted modifications are:

- `DirectoryConstants.DS_MODIFY_ADD`: This adds the given set of values to the existing values
- `DirectoryConstants.DS_MODIFY_DELETE`: This deletes the given set of values from the existing values

- `DirectoryConstants.DS_MODIFY_REPLACE`: This replaces all the existing values with a new set of values

This example adds two new values for the `telephonenumber` attribute.

```
Vector newVals = new Vector();
newVals.add("408 7394050");
newVals.add("650 7394050");
ldapobj.modifyAttributeValue
   ("telephonenumber", newVals, DirectoryConstants.DS_MODIFY_ADD);
```

Rule Management API

The rule management API is a set of Java classes that can be used to create, access and manage server side rules. Rules are represented as Java objects and can be saved persistently in the Oracle Internet Directory as an attribute of a user.

Server Side Rules

A mail rule is a potential action that, when a certain event happens and a certain condition is satisfied, is taken upon an e-mail message on behalf of the owner of the rule. Rules can be created and stored persistently on the mail server.

The following is an example of a rule, expressed in English:

If an e-mail message arrives in my inbox and its subject contains the phrase "Get paid to surf" then delete the message.

In this example:

- The event is the arrival of an e-mail message in the inbox
- The condition is the presence of the phrase in the subject
- The action is the deletion of the message

Each event represents a change of state of a particular message during its lifecycle in the mail server. In the above example, the event changes the state of the message from To be delivered to Delivered.

Conditions are similar to Boolean expressions, in which relational and logical operations test message attributes. In the example, the subject is the mail attribute to be tested, and the conditional expression is a relational operation that tests whether the attribute contains "Get paid to surf".

Optionally, multiple conditions can be combined using logical operators such as And and Or to form compound conditions. Additionally, a condition can be an external function call that returns a Boolean value.

Actions are operations that can act upon a message, such as the deletion of the message. In addition, an action can be any external procedure that is callable in PL/SQL from within the mail server.

Rule Components

Rules are owned by either individual users or a group of users collectively. The top-level entity owning the rules can therefore be either a mail user, a domain, or an entire e-mail system, which can contain more than one domain. The top-level entities are referred to as accounts.

For every account, one can have rules defined under a set of events, such as when new mail is delivered or when the message is read. Each event is associated with a rule list,

and an account can have several rule lists, with at most one per event. For any event, a rule list can contain a list of rules that are executed sequentially when the event occurs.

A rule is defined by a condition and a list of actions. A rule with no condition is said to be unconditional, therefore the actions are always carried out.

Conditions can be simple and complex. For example, a condition that compares an attribute with a literal value using the relational operator "contains" is a simple condition. A complex condition can combine several sub-conditions. A condition can be also be a user-defined procedure, referred to as an external condition. There is also a special kind of condition, called an InSection condition, which performs a content-based search on a message.

When a rule's conditions are met, actions are taken. Actions are defined by the rule's commands, such as "move message to a folder" or "forward message to a recipient," and associated parameters, such as the name of the folder to which the message is moved or the address of the recipient to whom the message is forwarded. Once all the rules for a user are constructed in Java objects, the RuleParser object can be used to save it.

Rule Authentication

Before a caller can access a user's rules, it must be authorized. The caller must authenticate with Oracle Internet Directory using either the `oracle.mail.OESContext` class or `oracle.mail.OESUser` class. The `OESContext` class should be used when the caller needs to manage system-wide or domain-wide rules, as this class provides application-level authentication. The `OESUser` class should be used when the caller needs to manage user-level rules. Once authenticated, the `OESContext` object or `OESUser` object must be passed to the `RuleParser` object before calling any other `RuleParser` methods.

Here is an example of system- or domain-level rule authentication:

```
appCtx.authenticate("admin_loginname", "admin_password", "ldaphost", ldapport);
OESContext appCtx = new OESContext(ESDSConstants.DS_CALLERTYPE_APP);
RuleParser parser = new RuleParser(appCtx);
```

Here is an example of user-level rule authentication:

```
OESUser ou =
OESUserFactory.getInstance().getEmailUser                ("ldaphost", ldapport,
"user_loginname", "password");
RuleParser parser = new RuleParser(ou);
```

Rule Validation

Before a rule is created on the server, the content of the rule is validated, preventing illegal rules from being executed at runtime. You can disable validation using the `RuleParser.setValidation()` method, which is useful if you want to temporarily save an incomplete rule. A non-validated rule can be saved persistently, but cannot be executed during runtime.

Rule Visibility, Activeness, and Group Affiliation

A rule has several attributes that are classified as follows:

- Visibility
- Activeness

- Group Affiliation

The following example sets the attributes of a rule:

```
RuleType rule_t = new RuleType();
rule_t.setVisible("no");
rule_t.setActive("no");
rule_t.setGroup("group1");
```

Visibility

A rule can be visible or invisible. An invisible rule functions as a normal rule, except that it is not shown to the user. The actual implementation of hiding a rule is left to the caller. The API is able to retrieve both visible and invisible rules.

Visibility is set using the `setVisible()` RuleType class method.

Activeness

A rule can be active or inactive. An inactive rule exists on the server but is not executed at runtime.

Activeness is set using the `setActive()` RuleType class method.

Group Affiliation

A rule can belong to a group. All rules belonging to the same group can be retrieved, activated, and disabled together in one call.

Group affiliation is set using the `setGroup()` RuleType class method.

Message Templates

Some rules require an action to generate a new message as a reply or a notification. The reply or notification can be stored in the rule content as templates containing substitutable parameters denoted by a parameter name enclosed by two percent (%) signs.

For example, an auto-reply template can be "Your e-mail regarding %rfc822subject% sent on %rfc822date% has been received." When the rule engine generates the reply message, the %rfc822subject% and %rfc822date% variables are replaced by the actual subject and date sent information obtained from the incoming message. The set of supported parameters is the same as the set of supported message attributes.

Message template text is used as parameter values for rule actions such as Notify, Reply, Replyall, and Forward. The following example uses message templates as desscribed above:

```
ActionType action_t = new ActionType();
action_t.addCommand("notify");
action_t.addParameter("jdoe@acme.com");
action_t.addParameter("Message Alert");
action_t.addParameter("You have received email from %rfc822from% regarding
%rfc822subject%");
```

Auto-Reply Effective Duration

To prevent auto-reply messages from flooding user inboxes, there is an effective duration for a specific reply action. The duration is specified as a number of days. If an auto-reply is sent to a particular address using a particular message template, the same reply is not sent to the same user again for the period of the effective duration, starting

from the time when the first reply is sent. The value is required in rule actions Reply and Replyall. The following example sets the duration to seven days:

```
ActionType action_t = new ActionType();
action_t.addCommand("reply");
action_t.addParameter("7");
action_t.addParameter("Message received");
action_t.addParameter("Your email regarding %rfc822subject% sent on %rfc822date%
has been received.");
```

XML Representation of Rules

Rule data can be serialized, or converted into plain text format using XML. It can then be easily transported between applications or stored off line. In fact, the rule API internally uses XML as the format to store in the Oracle Internet Directory. To flatten rule Java objects into XML text, use the `print()` method from the Account class as follows:

```
XMLOutputStream out = new XMLOutputStream(System.out);
account.print(out);
out.writeNewLine();
out.flush();
```

Code Sample

The following example demonstrates a simple rule:

```
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.util.*;
import java.io.*;
import oracle.mail.*;
import oracle.mail.sdk.rule.*;
import oracle.mail.sdk.ldap.*;

public class Demo {

  public static main (String args[]) throws Exception {

    // authenticate as user
    System.setProperty("oracle.mail.ldap.admin_dn", "cn=orcladmin");
    System.setProperty("oracle.mail.ldap.admin_password", "adminpwd");
    OESUser ou = OESUserFactory.getInstance().getEmailUser
        ("ldap_server_host", 389, "testuser1@oracle.com", "user1_pwd");
    parser = new RuleParser(ou);

    // first create the top level user account type object
    AccountType acnt_t = new AccountType();

    // set ownerType, either system, domain or user (default)
    acnt_t.setOwnerType("user");

    // for system rules, this is the string "UM_SYSTEM",
    // for domain rules this is the domain
    // name, such as "oracle.com", for user rules this is the
    // fully qualified username, such as testuser1@oracle.com
    acnt_t.setQualifiedName("testuser1@oracle.com");

    // create a rulelist type object, set the event
```

```
RuleListType rlist_t = new RuleListType();
rlist_t.setEvent("deliver");

// create a rule type object
RuleType rule_t = new RuleType();
rule_t.setDescription("a new rule");
rule_t.setGroup("group1");

// create a condition type object
ConditionType cond_t = new ConditionType();

// create a simple attribute:
cond_t.addAttribute("rfc822subject");
// or create an attribute object with parameters:
//
// AttributeType attr_t = new AttributeType("xheader");
// attr_t.setParam("X-Priority");
// cond_t.addAttribute(attr_t);

// create a simple operator:
cond_t.addOperator("contains");
cond_t.addOperand("Hello");

// create an external condition
ConditionType cond_t2 = new ConditionType();
cond_t2.addProcCall("extern_cond");

// create a negation of disjunction of above two conditions
// (i.e. not (cond1 or cond2) )
ConditionType cond_t3 = new ConditionType();
cond_t3.setJunction("or");
cond_t3.setNegation("yes");
cond_t3.addCondition(cond_t);
cond_t3.addCondition(cond_t2);

// add the condition object to the rule type object
rule_t.addCondition(cond_t3);

// create an action type object
ActionType action_t = new ActionType();
action_t.addCommand("moveto");
action_t.addParameter("/testuser1/folder1");

// add the action to the rule object
rule_t.addAction(action_t);

// create a second action object and add it in the rule type
ActionType action2_t = new ActionType();
action2_t.addCommand("call");
action2_t.addParameter("extern_action");
action2_t.addParameter("param1");
action2_t.addParameter("param2");
rule_t.addAction(action2_t);

// add the rule object in the rulelist type object
rlist_t.addRule(rule_t);

// add the rulelist object in the account type object
acnt_t.addRulelist(rlist_t);
```

```
    // create an account object on based the type object
    Account acnt = new Account(acnt_t);
    parser.setRuleObjects(acnt);
    }
}
```

Wireless Filters and Profiles API

To efficiently access e-mail over mobile devices or over a slow link, it is necessary to have filters on folders that enable only a small subset of e-mails to be downloaded. The wireless filters and profiles feature enables the user to define search criteria on folders. When a search criteria is defined and the folder is opened, only messages that meet the criteria are selected and are visible to the client. In addition, it also enables the user to define multiple criteria on a folder, through the use of profiles for accessing different sets of messages.

The different wireless filters and profiles components are:

- Filter - This is the search criteria defined on a folder, and supports all the search conditions defined in the IMAP protocol. This includes complex conditions joined by "and" and "or."
- Virtual Folder - The folder that has a defined filter. When the folder is opened by the client only the messages meeting the filter criteria are displayed.
- Profile - The name associated with a set of virtual folders, used during log on to inform the server which filters to apply. For example, if a user defines a profile on the INBOX folder called WP1 with the criteria "show urgent messages," and defines a second profile on the INBOX folder called WP2 with the criteria "show messages from sender john and from sender scott." When the user logs in as `username#wp1@domain_name` and selects INBOX, the only messages that appear are those marked urgent. However, if the user logs in as `username#wp2@domain_name` and selects INBOX, the messages from sender john and scott are displayed.

The various interfaces supporting virtual folders are:

- Java SDK - As part of the Java SDK, procedures are available for creating and modifying profiles and filters. In addition, the special log on format for accessing the virtual folders is also supported.
- IMAP server - The special log in format for accessing virtual folders is supported. This enables access to virtual folders through standards-based clients such as Outlook Express and Netscape Communicator.

Listing Wireless Filters

The following example shows how to list wireless filters:

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;

import javax.mail.search.*;

import oracle.mail.sdk.esmail.OracleEsProfile;
import oracle.mail.sdk.esmail.OracleEsFilter;
import oracle.mail.sdk.esmail.OracleStore;
```

```
public class ListFilter {

    static String password = null;
    static String user = null;
    static String host = null;
    static int port = -1;
    static String mbox = "INBOX";
    static String root = null;
    static boolean recursive = false;
    static String pattern = "*";
    static boolean verbose = false;
    static String namespace = null;
}

public static void main (String argv[]) throws Exception {

    for (int i = 0; i < argv.length; i++) {
        if (argv[i].equals("-U")) user = argv[++i];
        else if (argv[i].equals("-P")) password = argv[++i];
        else if (argv[i].equals("-D")) host = argv[++i];
        else if (argv[i].equals("-I")) port = Integer.parseInt(argv[++i]);
        else if (argv[i].equals("--")) {
            i++;
            break;
        }
        else if (argv[i].startsWith("-")) {
            System.out.println("Usage: ListFilter [-D ldap_host] [-I ldap_port]
[-U user] [-P password]");
            System.exit(1);
        }
        else {
            break;
        }
    }

    Properties props = System.getProperties();
    Session session = Session.getDefaultInstance(props,null);
    session.setDebug(true);

    Store store = null;
    store = session.getStore("esmail");

    if (user != null && password != null && port != 0 && host != null) {
        store.connect(host, port, user, password);

        OracleEsProfile wp2 =
        new OracleEsProfile(store, "wp2", "Wireless Profile 2");
        wp2.create();
        OracleEsProfile[] profiles = ((OracleStore)store).listProfile();
        System.out.println("Number of profiles = " + profiles.length);

        for (int i = 0; i < profiles.length; i++) {
            System.out.println("PROFILE " + (i+1));
            System.out.println("profile name = " + profiles[i].getName());
            System.out.println("profile description = " +
                               profiles[i].getDescription()+ "\n");
            System.out.println("List the filters:");
            OracleEsFilter[] filtersList = profiles[i].listFilters();
            System.out.println("Number of filters : " + filtersList.length);
            for (int j = 0; j < filtersList.length; j++) {
```

```
                System.out.println("filter folder = " +
                filtersList[j].getFolder().getFullName());
                System.out.println("filter description = " +
                filtersList[j].getDescription());
                SearchTerm st = filtersList[j].getCriteria();
                if (st instanceof FromTerm) System.out.println("from term");
            }
        }
    }
    store.close();
}
}
```

Adding Filters to a Profile

The following example shows how to add a wireless filter to a profile:

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;

import javax.mail.search.*;
import javax.mail.Address;
import javax.mail.internet.InternetAddress;

import oracle.mail.sdk.esmail.OracleEsProfile;
import oracle.mail.sdk.esmail.OracleEsFilter;
import oracle.mail.sdk.esmail.OracleStore;

public class AddFilter {
    static String password = null;
    static String user = null;
    static String host = null;
    static int port = -1;
    static String mbox = "INBOX";
    static String root = null;
    static boolean recursive = false;
    static String pattern = "*";
    static boolean verbose = false;
    static String namespace = null;

public static void main (String argv[]) throws Exception {
    for (int i = 0; i < argv.length; i++)
    {
    if (argv[i].equals("-U")) user = argv[++i];
    else if (argv[i].equals("-P")) password = argv[++i];
    else if (argv[i].equals("-D")) host = argv[++i];
    else if (argv[i].equals("-I")) port = Integer.parseInt(argv[++i]);
    else if (argv[i].equals("--"))
    {
        i++;
        break;
    }
    else if (argv[i].startsWith("-"))
    {
    System.out.println(
        "Usage: AddFilter [-D ldap_host] [-I ldap_port] [-U user] [-P password]");
    System.exit(1);
    }
```

```
    else
    {
    break;
    }
}

    Properties props = System.getProperties();
    Session session = Session.getDefaultInstance(props,null);
    session.setDebug(false);

    Store store = null;

    store = session.getStore("esmail");
    if (user != null && password != null && port != 0 && host != null)
    {
        store.connect(host, port, user, password);
        OracleEsProfile wp2 =
            new OracleEsProfile(store, "wp2", "Wireless Profile 2");
        wp2.create();

        OracleEsProfile[] profiles = ((OracleStore)store).listProfile();
        System.out.println("Number of profiles = " + profiles.length);

        for (int i = 0; i < profiles.length; i++)
        {
            System.out.println("PROFILE " + (i+1));
            System.out.println("profile name = " + profiles[i].getName());
            System.out.println("profile description = " +
            profiles[i].getDescription()+ "\n");
        }
        Folder folder = store.getFolder("xyz");
        OracleEsFilter filter =
        new OracleEsFilter
                (folder, "F2",
                 new FromTerm(new InternetAddress("tuser1@us.oracle.com")));
        profiles[0].addFilter(filter);
    }
    store.close();
}
}
```

External Condition

An external condition is a PL/SQL function that takes the following format:

```
function <func_name> (p_sessionid in integer,
                      p_msgobj in mail_message_obj) return integer;
```

Given the session ID and a message object, the function should return a number that indicates whether the condition is met.

If the return value is 0, the server regards it as a positive condition match, and if the return value is non-zero, it is regarded as a failed condition match. If a rule uses an external condition function, it must be manually loaded to the database server where the user resides before the condition can take effect.

To set a condition to be an external condition, use the ConditionType class method `addProcCall()` as follows:

```
ConditionType cond_t = new ConditionType();
cond_t.addProcCall("ext_func_name");
```

Invoking an External Action

An external action is a PL/SQL procedure that takes the following format:

```
procedure <proc_name>(p_event in number,
                      p_sessionid in number,
                      p_msgobj in mail_message_obj,
                      p_param1 in varchar2,
                      p_param2 in varchar2,
                      p_status out number);
```

The event ID parameter takes the following possible values:

```
es_rule.c_copy
es_rule.c_deliver
es_rule.c_expunge
es_rule.c_flagchange
```

The procedure also takes a session ID, current message object and two user-defined parameters set at rule creation time. After the procedure is completed, it should put a execution result value in the status parameter. A zero value in status indicates a normal execution, and a positive status indicates an abnormal execution.

To set an external action using server-side rules, use the ActionType class `addCommand()` method, then call `addParameter()` three times, with the first added parameter being the procedure name, and the second and third parameters being the user-defined parameters `p_param1` and `p_param2` above. The following example sets an external action using server-side rules:

```
ActionType action_t = new ActionType();
action_t.addCommand("call");
action_t.addParameter("ext_proc_name");
action_t.addParameter("param1");
action_t.addParameter("param2");
```

3

Custom Actions for Mail Rules

Oracle Collaboration suite provides the ability to customize Oracle Mail server-side rules to perform actions not provided by the Suite. These custom actions are program modules written in either PL/SQL, C, or Java. Regardless of the implementation language, developers must also write a PL/SQL "call specification" to provide the Oracle Mail server rules engine an interface to invoke the custom actions.

Implementing a Custom Rule Action in C

Rule action templates should be written as external C procedures, and stored in dynamic shared libraries (such as a Solaris .so library).

The steps for creating an external C procedure for use by a rule are:

- Set up the environment for calling external procedures by adding entries to the files `tnsnames.ora` and `listener.ora`, and then starting a Listener process exclusively for external procedures. By default, the agent that handles external procedures is named `extproc`. This agent runs on the same database instance as the main application. The database server, the agent process, and the listener process that spawns the agent must all reside on the same host.
- Identify the shared library, which is in this case a .so file (an operating system file that stores external procedures, and can be dynamically loaded). Using the CREATE LIBRARY statement, create a schema object called an *alias library* which represents the .so library. Specify the full path to the .so library in this object.
- Publish the external procedure by using the PL/SQL call specification mechanism. Oracle Mailcan only use external procedures that are published through a call specification that maps names, parameter types, and return types for C external procedures to their SQL counterparts.

After creating the action, create a rule using the custom action by creating an XML file for the rule, and save the rule using the `oesrl` command Alternately, use the Rules Java API to create a new rule.

Please refer to the *Email Java API Reference (Javadoc)* for more information on this process.

A Custom Action Written in C and PL/SQL

The following sample custom action writes the input parameters (`folder_id`, `msg_uid`, `param1`, and `param2`) to an OS file.

Step 1

Grant execute permissions to the user that will run the custom action, as follows:

```
<SQL> connect es_mail/<password>;
<SQL> grant execute on mail_message_obj to <username>;
```

Step 2

Supply the following custom procedure to SQL Plus:

```
CREATE OR REPLACE PROCEDURE procwritefile (p_event IN NUMBER,
                                           p_sessionid IN NUMBER,
                                           p_msgobj IN es_mail.mail_message_obj,
                                           p_sessionid IN NUMBER,
                                           p_param2 IN VARCHAR2,
                                           p_status OUT NUMBER) AS
BEGIN
    p_status: = cs_writefile (p_event, p_sessionid, p_msgobj.folder_id,
                              p_msgobj.msg_uid, p_param1,p_param2);
END;
```

Step 3

The above procedure calls a function `cs_writefile`, which is implemented in C below. The following code writes the parameters to a file.

```
#include <stdio.h>
#include <stdlib.h>

int writefile (int event, int session_id, int fid,
               int muid, char *param1, char *param2) {
    FILE *fp = NULL;
    char buf[1024];

    /* open file for writing */
    if ((fp = fopen("/tmp/ruleaction.txt", "w")) == NULL) {
        return (-1);
    }

    /* write parameters and close the file */
    sprintf(buf,
            "fid : %d, muid : %d, param1 : %s, param2 : %s",
            fid, muid, param1, param2);
    fputs(buf, fp);
    fclose(fp);
    return 0;
}
```

Step 4

Compile this C function and put it into a shared library called libesrule.so

```
% cc -G writefile.c -o libesrule.so
```

To load this shared library into the database, identify the shared library by using the CREATE library command in SQL Plus, and give the library a name (`libesruleplsql` in this case).

```
<SQL> connect system/<password>
<SQL> grant CREATE LIBRARY to <username>;
<SQL> connect <username>/<password>;
<SQL> CREATE OR REPLACE LIBRARY libesruleplsql AS '/absolute/path/libesrule.so';
```

Step 5

Publish the external C callout using a PL/SQL call specification. Create the specification in an application account, and issue the necessary grants issued for an es_mail database user to this account.

> **Note:** Call specifications for custom actions should not be created under the es_mail database user account itself. Instead, grant es_mail permissions to a regular user account.

```
CREATE OR REPLACE FUNCTION cs_writefile (p_event     IN BINARY_INTEGER,
                                         p_sessionid IN BINARY_INTEGER,
                                         p_folder_id IN BINARY_INTEGER,
                                         p_msgid     IN BINARY_INTEGER,
                                         p_param1    IN VARCHAR2,
                                         p_param2    IN VARCHAR2)
RETURN PLS_INTEGER
    AS LANGUAGE C
        NAME "writefile"
        LIBRARY libesruleplsql
        PARAMETERS (p_event      INT,
                    p_sessionid  INT,
                    p_folderid   INT,
                    p_msguid     INT,
                    p_param1     STRING,
                    p_param2     STRING,
                    RETURN INT);

SQL> connect <username>/<password>
SQL> grant execute on cs_writefile to es_mail
SQL> grant execute on procwritefile to es_mail
```

Custom actions for rules implemented as C callouts must support the following list of parameters:

```
p_event     IN number
p_sessionid IN number
p_folder_id IN number
p_msguid    IN number
p_param1    IN varchar2        /* custom parameter */
p_param2    IN varchar2        /* custom parameter */
```

Step 6

Now that a custom action has been created and implemented as a C callout, it can be called as part of a mail rule. To create a test rule for this action, use the following XML code.

```
<account qualifiedName="jdoe@orcltest.com">
  <rulelist event="deliver">
    <rule description="test">
      <action>
        <command tag="call">
        <parameter>username.procwritefile</parameter>
        <parameter>param1</parameter>
        <parameter>param2</parameter>
      </action>
    </rule>
```

```
  </rulelist>
</account>
```

Put the above XML into a file called rule.xml. Save the rule from the command line, using the `oesrl` command.

```
% oesrl -x rule.xml
```

Implementing a Custom Rule Action in Java

The process for implementing a rule in Java is similar to the C method, except instead of an external object, the Java code is placed in a Java stored procedure. The steps are as follows:

- Write the action template for rule as a Java stored procedure and load the Java classes into the Oracle database.
- Publish the Java classes. The methods of a Java class are not published automatically when they are loaded into the database. In order to be published, every method requires a call specification, which maps Java method names, parameter types, and return types to their SQL counterparts. You can declare a function or procedure call specification for a Java method using a SQL CREATE FUNCTION or CREATE PROCEDURE statement.

Once the classes have been published, you can create a rule that uses the custom action by creating a XML file for the rule. Save this rule using the `oesrl` command.

For further reading refer to the following Oracle Database documentation:

Database Application Developer's Guide - Fundamentals - Chapter 8: Calling External Procedures
Oracle Database Java Developer's Guide - Chapter 6: Publishing Java Classes With Call Specs

A Custom Action Written in Java

Example: A sample custom action for rule to write a set of parameters (`folder_id`, `msg_uid`, `param1`, and `param2`) to an OS file.

Step 1

Grant execute permissions to the user that will run the custom action, as follows:

```
<SQL> connect es_mail/<password>;
<SQL> grant execute on mail_message_obj to <username>;
```

Step 2

Supply the following custom procedure to SQL Plus:

```
CREATE OR REPLACE PROCEDURE procwritefile(p_event IN NUMBER,
    p_sessionid IN NUMBER,
    p_msgobj IN es_mail.mail_message_obj,
    p_param1 IN VARCHAR2,
    p_param2 IN VARCHAR2,
    p_status OUT NUMBER) AS
BEGIN
    p_status := cs_writefile (p_event, p_sessionid, p_msgobj.folder_id,
    p_msgobj.msg_uid, p_param1, param2);
END;
```

Step 3

The following Java code implements the action for the `cs_writefile` call in the above PL/SQL code. The code writes the contents of the `param1` parameter to a local file.

```
import java.io.FileWriter;
import java.lang.StringBuffer;

public class filewrite {

public static int writeFile(int event, int session_id, int folder_id,
                            int msg_uid, String param1, String param2) {
    int retVal = 0; //failure 1, success 0

    try {
        StringBuffer strBuf = new StringBuffer();
        File outFile = new File("/tmp/ruleaction.txt");
        FileWriter out = new FileWriter(outFile);

        strBuf.append(folder_id);
        strBuf.append(msg_uid);
        strBuf.append(param1);
        strBuf.append(param1);
        out.write(strBuf.toString(),0,strBuf.length());
        out.close();
    } (Exception e) {
        e.printStackTrace();
        retVal = 1;
    }
    return retVal;
}
```

Compile the above program using the command:

```
% javac filewrite.java
```

Step 4

Load the compiled class file into the Oracle database using the loadjava utility.

```
% loadjava -u username/userpw@<connect_string> -v -r filewrite.class
```

Step 5

This sample program involves writing data to an OS file, so the OS file must be granted permission by issuing the `dbms_java.grant_permission` command from SQL Plus as a SYS user. The command is:

```
SQL> exec dbms_java.grant_permission
('SYSUSERNAME,'SYS:java.io.FilePermission','/tmp/ruleaction.txt','write');
```

Step 6

Provide the following call specification to SQL Plus:

```
CREATE OR REPLACE FUNCTION cs_writefile (p_event     IN NUMBER,
                                        p_sessionid IN NUMBER,
                                        p_folderid  IN NUMBER,
                                        p_msguid    IN NUMBER,
                                        p_param1    IN VARCHAR2,
                                        p_param2    IN VARCHAR2)
RETURN NUMBER
```

```
AS LANGUAGE JAVA
NAME filewrite.writeFile (int, int, int, int,
                          java.lang.String, java.lang.String)
                          return int);
```

This call specification should be created in an application account with es_mail permissions. Grant these permissions as follows:

```
SQL> connect <username>/<password>
SQL> grant execute on cs_writefile to es_mail
SQL> grant execute on procwritefile to es_mail
```

> **Note:** Call specifications for custom actions should not be created under the es_mail account itself. Instead, create a regular application account for this purpose.

Custom actions for rules implemented as Java callouts should support the following list of parameters:

```
p_event     IN NUMBER
p_sessionid IN NUMBER
p_folder_id IN NUMBER
p_msguid    IN NUMBER
p_param1    IN VARCHAR2        /* custom parameter */
p_param2    IN VARCHAR2        /* custom parameter */
```

Step 7

To use the new custom action in a rule, an XML file corresponding to the user's rule profile must be created manually:

```
<account qualifiedName="jdoe@orcltest.com">
  <rulelist event="deliver">
    <rule description="test">
      <action>
        <command tag="call">
        <parameter>username.procwritefile</parameter>
        <parameter>param1</parameter>
        <parameter>param2</parameter>
      </action>
    </rule>
  </rulelist>
</account>
```

Step 8

Save the rule with the `oesrl` command from the command line, as follows:

```
% oesrl -x rule.xml
```

4

Mail Server Plug-In Framework

The Oracle Collaboration Suite Mail server plug-in interface provides a framework to pass mail and messaging information between existing Oracle Mail protocol servers and third-party vendor and partner products. This interface can be used to extend the functionality of mail servers in many ways - for example, to scan the mail system for viruses, provide efficient workflow integration, or track and archive mail messages that have entered or left the system.

Introduction to the Mail Server Framework Plug-In API

The Mail Server Framework allows two types of plug-ins. The first takes the form of a separate executable, written in any language, and spawned from the SMTP processes. The second is a shared library written in C and linked into the SMTP processes. Libraries are written and compiled like any non-Oracle Mail Server library, without added precompilation or preprocessing. However, plug-in libraries must be linked as shared libraries. The library plug-ins will be loaded into the Mail server address space during server startup. Configuration of both types of plug-ins is done through policy pages. For more information on configuring plug-ins, see Chapter 8, "Oracle Mail Policies"" of *Oracle Mail Administrator's Guide*.

If the plug-in is a separate application or executable, the appropriate mail process will spawn the executable for each mail message. The mail message is passed to the application through standard input. If the custom executable is called from a mail process that uses envelope information, then the envelope information will be passed on the command line as well.

The Mail Server processes can call the plug-ins at specific event control points, of which there are two types. There are *high-level* event control points, which specify when plug-ins are called, and *low-level* event control points, which define the information passed to the plug-in. A system administrator will set the high-level event control points through the administration pages. Low-level control points apply only to C plug-ins (shared libraries) and the plug-in specifies these control points as a list of services supported by the filter interface.

When to Call the Plug-in

Mail processes that route or send mail messages will eventually pass through high-level event control points. There are three such control points. The first point is called **Incoming**, which is reached as the message enters the mail process that routes or sends a message, regardless of the sender or recipient of the message. Note that the scanner plug-in interface is not called until after the message passes through the protocol server's mail relay control logic. That is, if the SMTP servers are not open relays, then all messages that pass the relay constraints will go through this event control point.

After the protocol server reviews the recipient list of the mail message, all messages that are destined for one or more recipients in one of the local domains will pass through the **Local** event. Afterwards, messages that are destined for one or more recipients in a mail domain that is not local will pass through the **Outgoing** event. Mail messages destined for recipients both a local domain and an external domain will pass through both Outgoing and Local event control points.

1. **INCOMING** - A plug-in configured to be called on the routing of all messages will be called for every message when a connection is made to the mail protocol service or when a message is submitted into a mail store.
2. **OUTGOING** - A plug-in configured to be called for outgoing will be called just before a connect request is made to the next message transfer agent.
3. **LOCAL** - A plug-in configured to be called for local will be called just before local delivery to the inbox or news store.

Each high-level event control point can call an array of one or more custom plug-ins. The order by which each plug-in is called can be set in the administration pages. If both types of plug-ins are configured for a Mail server, the framework will call C plug-ins in the specified order before external process plug-ins. Messages and connections will pass through the Incoming high-level event control point before passing through either the Outgoing or Local control points.

C Plug-In Service Support

Information passed to a plug-in depends upon the services supported by the plug-in. The information is passed at low-level control points, which include **Connect**, **Reset**, **Envelope**, **Message**, and **Authentication**.

- **Connect** - Processing a connection event is mandatory. For mail protocol servers that route mail or news, this event corresponds to the client connect or server connect request event. For the background tasks of the mail Information Store (virus scrubber), the event corresponds to the start of a new mail message. The host and IP address information from the connection are made available to the plug-in. When called by the virus scrubber, the host and IP address information will be those of the local host.
- **Reset** - Processing a reset event is mandatory. It is used to finish the processing at the end of a message, a message reset request from the client, or a connection close.
- **Envelope** - A plug-in can process envelope information prior to receiving the mail message. In addition to the information available at the Connect control point, the plug-in has access to fully resolved originator and recipient mail address information and authentication information. This information is not the same information provided at the Authentication event control point. This authentication information will only exist if there was a successful authentication of the connection. The information provided will be the fully qualified email address. The plug-in can trust the email address as the authenticated address of the user that connected. Supporting this control point is optional.
- **Message** - A plug-in can process message information. In addition to the information available at the Connect and Envelope control points, the plug-in has access to details of the message as well as the complete, unaltered message. Supporting this control point is optional.
- **Authentication** - This control point allows a plug-in to take control of authentication. A plug-in that publishes its support of the authentication event will be sent an email account and a password or key at this control point. At this

point, authentication has not yet taken place. The plug-in can then perform verification of the address and password/key pair, and return either a success or a failure of this verification. Supporting this control point is optional.

Table 4–1 Services Supported by Server Type

Server Type	CONNECT	AUTH	ENVELOPE	MESSAGE	RESET
SMTP_IN	MANDATORY	No	Yes	Yes	MANDATORY
SMPT_OUT	MANDATORY	No	Yes	Yes	MANDATORY
NNTP_IN	MANDATORY	Yes	Yes	Yes	MANDATORY
LS	MANDATORY	No	No	Yes	MANDATORY
VS	MANDATORY	No	No	Yes	MANDATORY

The above table shows the low-level services each of the six mail protocol server types are able to call. A plug-in must write a CONNECT and a RESET function. All protocol server types always call these two functions. Protocol servers optionally support AUTH, ENVELOPE, and MESSAGE functions. In the case of AUTH, only the NNTP_IN process supports this low-level event control point.

Calling the Plug-In From the Mail Protocol Server

All custom plug-ins must be placed in a specific directory, or the plug-in shared library must be in the LD_LIBRARY_PATH environment variable. Each middle-tier Oracle home has the directory structure `./oes/lib` below the Oracle home where the plug-ins should be located. If the shared library is not in this directory, then the LD_LIBRARY_PATH environment variable must include the directory where the library is located. The name (or complete path, if an executable) of the plug-in must be entered in the filter maintenance page when managing policies through the administration tools.

Furthermore, if the plug-in is a shared library, the plug-in must support a list of mandatory functions. The functions for the initialization, registration, and close actions are mandatory and must have specific names so that the mail protocol servers know how to call them. Within the initialization function, a plug-in will tell the protocol server which low-level event control points the plug-in can process. The registration function provides function pointers to the plug-in to request services of the mail protocol server.

Header File Inclusion for C Plug-ins

All custom plug-ins must include the C interface definition located in the `esefif.h` header file. This header file contains:

- platform-specific calling conventions;
- platform-specific export symbols for shared libraries
- platform independent oracle datatypes, and
- an `extern` wrapper for C++ code.

External Filter Process Plug-In

The Mail server framework provides a way to integrate external processes into a mail server. The Mail server invokes a configured filter at each of a series of high-level control points and communicates with the spawned process using standard input and

output. The external filter executable must have its permissions set to Execute. The Mail server will throw errors in its log if it fails to spawn the filter process.

Communication from Mail Server to External Filter Process

Both the message body and envelope are passed when the filter process is spawned by the Mail server. The call to the external process will pass the envelope information, as well as other information (such as message size) by command-line arguments in the syntax described below. The filter process reads the envelope information using `argv[]`. The message body is passed to the standard input (*stdin*) of the external process; the filter process must read its standard input until EOF (end of file) occurs, indicating the end of the passed message. The Mail server will return errors if the filter process exits or aborts before the complete message body is sent by the Mail server.

Communication from External Filter Process to Mail Server

The external process can apply its own filtering on the envelope and the received message body. The process must return a status code (success, failure or modify) to the Mail server on the standard output (*stdout*). If a success status is received, the Mail server proceeds with running the remaining filters (if any) or with delivery processing of the mail message. If a failure status is received from the external filter, the message will be rejected by the Mail server and will not be delivered. The external filter can also pass a modified mail message back to the server, which is then delivered to the recipients. If the external process aborts or exits during processing of a mail message, the Mail server will assume a temporary processing error. In this case, the server will re-queue the message and retry according to the Mail server's retry mechanism.

Command-Line Arguments

The call to the external process has the following syntax:

```
filter_process host=host mailfrom=mailfrom rcptto=rcptto msgsize=msg_size
```

where

- `filter_process` is the path of external filter executable as given in the administration pages,
- `host` is the host name of mail client,
- `mailfrom` is the address sent by the client during the Mail server protocol exchange,
- `rcptto` is the list of recipients from the Mail server protocol exchange. This is a list separated by a comma and is enclosed in brackets and,
- `msg_size` is the size of the mail message.

The external filter process should rely on the keywords in the arguments rather than the contents of the `argv` array. For example, host information will not necessarily be passed in `argv[1]`.

Format of Output from External Filter Process

After the filter process has finished processing the mail, it should return the status and the changed message (if any) back to the server. To do this, the process must write the following information to its standard output (*stdout*):

```
status_code [version_definition]
repaired_message
```

where

- possible value for `status_code` are
  - `0` - The message is clean and is to be sent to the recipient
  - `1` - The message is not clean and is to be rejected
  - `2` - The message was not clean but was repaired, or the message was modified for Mail policy reasons. The changed message will be sent to recipients.
- `version_definition` can be returned optionally by the filter process and is stored with the message. This can be used to store version information, such as a virus definition database identifier. Because it is stored with the message, it can be used later in the virus scrubber process to selectively re-scan messages.
- `repaired_message` is the modified message that should be sent to recipients.

C Plug-in Filter Process Flow

Initialization and Registration

The plug-in tells the protocol server which services it supports during the initialization phase so that the protocol server or virus scrubber knows how to call the plug-in. Through a series of callback registrations during the initialization phase, the plug-in learns how to communicate back to the server.

When a mail protocol server first loads, the server calls the plug-in's `esefifInit()` function, and then calls the `esefifRegister()` function repeatedly, once for each framework callback service supported by MTA. Each time the server calls `esefifRegister()`, it passes the plug-in a function pointer to one of the services it could provide. For example, it could tell the plug-in how to communicate with the framework and log a message in the mail protocol server log file.

The mail protocol servers are multi-threaded, and each thread can handle a connection. During the connection, the thread will call the plug-in at each supported low-level event control point.

Processing Low-Level Control Points

The five low-level event control points are passed through on a per-connection basis. The exception is the virus scrubber process, which runs against a mail store. For example, assume a plug-in written for the Oracle Message Transfer Agent (SMTP_IN process in this case) notifies the Oracle protocol server that it supports Connect, Message, and Reset servers (assume it does not support Envelope). Now suppose a remote MTA sends three messages over a single connection to the Oracle Message Transfer Agent. The protocol server would call `esefifConnect()` once after the remote MTA has opened a connection. Then it would call `esefifMessage()` three times; once after each message is received. Finally `esefifReset()` would be called when the connection is dropped or closed. If the plug-in published supported "ENVELOPE" as well, then `esefifEnvelope()` and `esefifMessage()` would be called three times each.

`esefifReset()` gets called at the end of the connection or if the client (or remote MTA) requests a temporary reset for the currently processed message. The plug-in can determine which type of reset is being called (i.e. the difference between End of Message/Connection and temporary message reset). The plug-in is responsible for any message level and connection-level cleanup of structures.

The plug-in framework allows a single plug-in (shared library) to be called by multiple protocol server types. The above plug-in could also apply to the virus scrubber. The virus scrubber would call `esefifConnect()`, then `esefifMessage()` and finally `esefifReset()` for each message that met the filter requirements.

Shutting Down

When a mail protocol server shuts down, the server calls the plug-in's `esefifClose()` function. The plug-in can then release resources allocated during `esefifInit()` as well as perform any other necessary cleanup.

Transferring Information Between a Plug-In and Mail Protocol Server

Information is passed between a protocol server and a plug-in through function pointers, which are registered with the plug-in when the protocol server starts. Each protocol server type will publish callback service functions to the plug-in, which can then call these functions as needed.

- **Framework Send function -** This function asks the framework to send the message to the plug-in. Since an email message can be quite large, calling this function causes the Oracle protocol server to, in turn, call the plug-in `esefifSend()` function in a loop. Each time `esefifSend()` is called, the protocol server sends a chunk of the message until the complete message is accepted.
- **Framework Receive function -** This function asks the framework to receive the message from the plug-in. This is called if the plug-in needs to modify the message before handing it back to the protocol server. For example, a virus scanning plug-in might "clean" a message and return it, or a disclaimer-adding plug-in might append a disclaimer and pass the message on. As an email message can be quite large, calling this function causes the Oracle protocol server to, in turn, call the plug-in `esefifRecv()` function in a loop. Each time `esefifRecv()` is called, the plug-in sends a chunk of the message until the complete message is accepted.
- **Framework Get Envelope function -** This function asks the framework to send the envelope information associated with this message. This information includes recipient list, sender, host, and authenticated user. When called at the Incoming event control point, the function returns a complete list of unresolved recipients and if available, a complete list of resolved recipients.. When called at Relay or Local control points, the recipient list is resolved but only contains relay or local domain recipients.
- **Framework Get Header Information function -** This function asks the framework to send the header information associated with this message. This will return the top-level header of the message in the format it was received from the client (or other MTA).
- **Framework Get Message Size function -** This function asks the framework to send the size of the message, in bytes.
- **Framework Get Message Identifier function -** Each mail protocol server has a default mail store where it performs certain tasks. This function asks the framework to send the mail store message identifier. Note that a single message stored in two different mail stores will have two different message identifiers.
- **Framework Allocate Memory function -** A plug-in can request the protocol server calling it to allocate memory on its behalf. Doing so uses the heap of the protocol server as opposed to the plug-in, which saves operating system calls.

- **Framework Free Memory function -** All memory explicitly requested by a plug-in from the framework must be freed with the framework's free memory function call.
- **Framework Set Version definition -** The Oracle mail store schema includes a history table (`es_scan_history`) of the messages that passed through the plug-in interface. The table consists of the mail store message identifier, scanner (plug-in) name, a definition, and a date stamp when the message passed through the plug-in. Each time a message passes through a plug-in, the plug-in can tell the protocol server to insert a record into the `es_scan_history` table.
- **Framework Add a Recipient function -** This function adds a "rcpt to:" entry to the message envelope.
- **Framework Delete a Recipient function -** This function removes a "rcpt to:" entry from the message envelope.
- **Framework Log function -** This function logs a message from a plug-in.
- **Framework Get Policy Identifier -** This function returns the archive policy ID (if any) of the calling plug-in. This function applies only to plug-ins used for archiving purposes.

Copying a Message Between the Server and Plug-in

The `esefifMessage()` is called by the protocol server when that protocol server has a message it wants the plug-in to process. Once the plug-in receives this notification, it would then call a framework function (for example, "Send Message" or "Get Message Size") within its `esefifMessage()` function.

Upon receiving a "Send Message" request from the plug-in, the protocol server will call the function `esefifSend()` in a loop, each time passing a chunk of the message, until the whole message has been passed. The plugin must collect these "chunks" and rebuild the message inside their `esefifSend()` function.

To return a modified message back to the protocol server, the plug-in would call the "Receive Message" framework function, and the protocol server would in turn call the plug-in's `esefifRecv()` function in a loop until the protocol server receives the complete modified message.

C Plug-In Functions

Plug-ins are written in C. The plug-in must be built as a shared library. A plug-in should contain the following functions:

Table 4–2 Mandatory Functions for Implementation by C Plug-Ins

Function	Mandatory?	Description
esefifInit()	Yes	The initialization function for the custom library. At a minimum, a global set of contexts must be initialized to pass between the Oracle mail protocol servers and the custom library.
esefifClose()	Yes	Performs any necessary tasks prior to shutdown. For example, freeing resources allocated in `esefifInit()`.
esefifRegister()	Yes	Registers, in the context of the plug-in, all the mail protocol server routines the framework provides.

Table 4–2 (Cont.) Mandatory Functions for Implementation by C Plug-Ins

Function	Mandatory?	Description
esefifConnect()	Yes	SERVICE: Processing at the end of the connect request.
esefifEnvelope()	No	SERVICE: Called when envelope information is received.
esefifMessage()	No	This function is called by the protocol server when that protocol server has a message it wants the plug-in to process.
esefifReset()	Yes	Processing at the end of each Message
esefifSend()	No	Receives the message sent from the mail protocol server to the plug-in.
esefifRecv()	No	Sends a modified (cleaned, appended, etc.) message received by the mail protocol server from the plug-in.

Return Codes

All plug-in functions must be written to return one of the following four return values. Note: some functions do not modify or reject mail messages and hence do not return those codes.

Table 4–3 Return Codes for C Plug-In Functions

Code	Description
ESEFIF_SUCCESS	Success.
ESEFIF_REJECT	Failure. A plug-in service function should return this code to the mail protocol server if it does not want the protocol server to allow the message to be sent or the connection to continue.
ESEFIF_ MODIFIED	Message Modified. The return code a plug-in service function returns in order to tell the mail protocol server that it needs to modify or clean the message before the message should be allowed to continue.
ESEFIF_ERROR	Error.

esefifInit()

```
int esefifInit(void **ifgctx,
               void *efgctx,
               void **services,
               char *scanflags,
               char *sysflags);
```

Purpose

- Initializes the plug-in.
- Processes parameters from plug-in administration screens.
- Allocates a global context for the plug-in.
- Tells the mail protocol server what low-level services it supports.

Parameters

ifgctx (OUT)
This is pointer to the global context of the plug-in. This context gets passed back to the plug-in by the Oracle mail protocol server every time a protocol server calls the plug-in and for every message.

efgctx (IN)
This is pointer to the global context for the framework. This pointer must be passed in each framework function call.

services (OUT)
This is a text string specifying the services supported by the plug-in and the name the function the protocol server must call to process the service. The syntax of the string is a single character ("E" or "F") followed by a colon followed by a list of services or service function names separated by commas. A string starting with "E:" is a variable number of elements listing the service names that the plug-in supports. A developer must write the plug-in using the expected function names. A string starting with "F:" is a fixed number of elements (eight) listing the function names that the plug-in developer expects the protocol server to call. When using the "F" option, the position 8 function (esefifReset) must be terminated with a comma. A developer will write up to nine functions. Any plug-in function can be renamed except the initialization function, which must be named "esefifInit".

Table 4–4

Service Syntax	Required	Position	Default Function Name
	Yes	1	EsefifClose
	Yes	2	esefifRegister
	No	3	esefifSend
	No	4	esefifRecv
CONNECT	Yes	5	esefifConnect
ENVELOPE	No	6	esefifEnvelope
MESSAGE	No	7	esefifMessage
RESET	Yes	8	esefifReset

Consider a plug-in written to look at the envelope and message of every message passed to it. The following two examples would be equivalent service strings for this plug-in:

```
"E:CONNECT,ENVELOPE,MESSAGE,RESET"
"F: , , , , esefifConnect, esefifEnvelope, esefifMessage,
esefifReset,"
```

Note the terminating comma after esefifReset.

The following example would tell the protocol server calling the plug-in that the plug-in function names are different from the default.

```
"F:pi_close, pi_reg, pi_send, pi_recv, pi_connect, pi_env, pi_
msg, pi_reset,"
```

scanflags (IN)
This is a text string retrieved from the scanner flags field for plug-in (filter) policy management pages. These flags provide a way to pick up necessary administration variables.

sysflags (IN)
This is a text string describing the system flags, or flags that are understood by the mail protocol server. The current release of the plug-in API only understands a single flag, `repairmsg`, with possible values of 1 and 0 (e.g. `repairmsg=1`). If the value is 1 then a protocol server will assume that it may get a request from the plug-in to modify a message. Otherwise it will ignore a request by the plug-in to receive a modified message.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

The initialization function is the one function written by the plug-in developer that must be called "esefifInit". This function must always be implemented.

esefifClose()

```
void esefifClose(void *ifgctx);
```

Purpose

- To shut down the plug-in.
- To free a global context for the plug-in.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

Returns

none

esefifRegister()

```
int esefifRegister(void *ifgctx,
                   int (*cb)(void*),
                   void*,void*),
                   unsigned int flags);
```

Purpose

- To register function pointers to framework services.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

cb (IN)
This is the function pointer to one of the twelve framework functions.

flags (IN)
An integer specifying which function pointer is being passed in.

Table 4–5 Flags for esefifRegister() function

Flag	Framework Function
ESEFIF_CALLBACK_SENDMSG	Send a message to the plug-in
ESEFIF_CALLBACK_RECVMSG	Receive modified message from the plug-in
ESEFIF_CALLBACK_ GETENVELOPE	Get envelope information
ESEFIF_CALLBACK_ GETMSGSIZE	Get the size of a message
ESEFIF_CALLBACK_GETMSGID	Get the mail store message ID of a message
ESEFIF_CALLBACK_FREE	Free memory that was allocated by the framework
ESEFIF_CALLBACK_ SETVERSION	Set the plug-in history definition text
ESEFIF_CALLBACK_ GETMSGHDR	Get the message's top-level header
ESEFIF_CALLBACK_WRITELOG	Log a message in the protocol server log file
ESEFIF_CALLBACK_ALLOC	Allocate memory
ESEFIF_CALLBACK_ADDRCPT	Add recipients to the recipient list
ESEFIF_CALLBACK_DELRCPT	Delete recipients from the recipient list
ESEFIF_CALLBACK_ GETPOLICYID	Get policy identifier of the archive plug-in, if any

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

The protocol server will call the registration function of a plug-in once for each framework function. The plug-in should update its global context with each function pointer.

esefifSend()

```
int esefifSend(void *ifgctx,
               char *buf,
               int buflen);
```

Purpose

- A wrapper function to copy a mail message sent from a protocol server to the plug-in.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

buf (IN)
This is a pointer to the block of memory containing a chunk of the message.

buflen (IN)
The size of each chunk of the message.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

This function is a wrapper used by the Oracle mail protocol server to send a message to the plug-in. In order to not have to build a complete message (which could be megabytes in size) and then copy it over to the library, the scanning interface uses this send function to retrieve the message in chunks. The buflen parameter specifies the size of each chunk.

This function must be written to receive the message from the sending protocol server. The protocol server will call this wrapper function within a loop, each time passing it "buflen" bytes of the mail message until the complete message has been sent. The protocol server will then stop calling this function and the framework function will return.

esefifRecv()

```
int esefifRecv(void *ifgctx,
               char *buf,
               int *buflen);
```

Purpose

- A wrapper function to copy a message received by a protocol server from a plug-in.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

buf (OUT)
This is a pointer to the block of memory containing a piece of the message.

buflen (IN/OUT)
The protocol server will set buflen to the maximum memory size it will accept, when it calls this function. The plug-in sets this value to the size of the buffer being returned. The protocol server will continue to call esefifRecv() until this function returns a NULL buf and a buflen of zero.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

This function is a wrapper function used by the Oracle mail protocol server to receive a message from the plug-in. In order to not have to allocate memory and move a complete message all at once (which could be megabytes in size), the interface uses this receive function to copy the message in chunks. This version of the interface sends in a maximum buffer size of 1024 bytes.

This function must be written to send a modified or cleaned message to the receiving protocol server. When a plug-in needs to return a modified message, it will call the framework function for sending the message. The protocol server will call this wrapper function within a loop, each time sending a piece of the mail message until the complete message has been sent. The protocol server will then stop calling this function and the framework function will return.

esefifConnect()

```
int esefifConnect(void *ifgctx,
                  void **iflctx,
                  void *efcbctx,
                  char *host,
                  char *ip,
                  int flags,
                  char *errmsg)
```

Purpose

- Allocate thread specific local context
- Process any connection information from the remote MTA or Client (SMTP specific)

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

iflctx (OUT)
This is a pointer to the local context of this connection, which will hold information to be passed between processing (CONNECTION, ENVELOPE, MESSAGE, RESET).

efcbctx (IN)
This is a pointer to the filter framework context. This context is only sent to the plug-in as a part of the mandatory CONNECT service. The context needs to be stored in the local context of this connection so that it can be made available to the other services. A service cannot call a framework function without the filter framework context.

host (IN)
This is a text string containing the host name of the client (or remote process) that is contacting the protocol server. If the mail protocol server calling the plug-in is the housekeeper, which does not accept connections from remote clients or processes, then this is filled the name of the local host running the housekeeper.

ip (IN)
This is a text string containing the decimal representation for the 32-bit IPv4 address ("nnn.nnn.nnn.nnn") of the client (or remote process) Internet protocol address that is contacting the protocol server. If the mail protocol server calling the plug-in is the housekeeper, which does not accept connections from remote clients or processes, then this is filled the name of the local host IP address.

flags (IN)
Unused. Will always be zero.

errmsg (IN/OUT)
The mail protocol server passes a pointer to a 512-byte buffer that holds error message text, should the plug-in need to reject the connection.

Returns

```
ESEFIF_SUCCESS     Success - Allow connection
ESEFIF_REJECT      Failure - Reject connection
```

```
ESEFIF_ERROR       Error
```

Comments

Information available to the plug-in consists only of the bind information for the connection.

The plug-in developer must implement this service. The protocol server calls this function first, once for each session connection from a client or other protocol server. In the case of the housekeeper protocol server, the connect function will be called once (and first) for each message.

The plug-in developer must allocate a context (iflctx), for the scope of the connection, which will be passed to all other plug-in processes when the mail protocol server calls them. It must also populate the context with the pointer to efcbctx so these subsequent services have access to all the framework functions.

Framework functions available to the plug-in developer:

ESEFIF_CALLBACK_FREE - Free Memory

ESEFIF_CALLBACK_WRITELOG - Log a message to the logfile

ESEFIF_CALLBACK_ALLOC - Allocate Memory

esefifEnvelope()

```
int esefifEnvelope(void *ifgctx,
                   char *envinfo,
                   unsigned int flags,
                   char *errmsg)
```

Purpose

- To read and process the envelope information.

Parameters

ifgctx (IN)
This is a pointer to the connection context of the plug-in. This is the same context allocated and populated in the esefifConnect() function.

envinfo (IN)
This is a pointer to the envelope information.

flags (IN)
These flags can be used to specify envelope properties, such as sender-only or rcpts-only.

```
 0 ALL env info (default).
 1 HOST (envinfo will be hostname)
 2 SENDER (envinfo will be sender)
 3 RECIPIENT (envinfo will be <rcpt1>,<rcpt2>,...
 4 AUTH (envinfo will be auth info)
 5 HELO/EHLO DOMAIN (envinfo will be domain).
```

errmsg (IN / OUT)
The mail protocol server passes a pointer to a buffer holding error message text that would be returned to the remote client or process in the case when the plug-in needs to reject the message based upon envelope information. The maximum buffer length is 512 bytes.

Returns

```
ESEFIF_SUCCESS     Success - Allow Login
ESEFIF_REJECT      Failure - Reject Login
ESEFIF_ERROR       Error
```

Comments

This is the service function that Oracle mail protocol servers use to send envelope information to the plug-in. Envelope information is only persistent for the life of the message as it passes through a message transfer agent (MTA). Only the Oracle SMTP and LIST SERVER processes would have this information or call this function. The envelope information is returned in a single buffer, which must be parsed and understood by this function.

The buffer contains up to five different pieces of information. Each piece of information is a keyword/value pair in the syntax "<keyword>=<value>". Pairs are separated by the pipe (|) character. One field, the recipient list, can contain multiple values. In this case, each value is separated by the comma (,) character.

```
host=<hostname>|mailfrom=<sender info>|rcptto=(<rcpt1>,<rcpt2>...)[|authinfo=<auth
info>]|domain=<helo domain>[|resrcpt=(<rrcpt1>,<rrrcpt2>...)]
```

Table 4–6 Description of Envelope Keywords

Keyword	Description
Host	The host name making the connection with the server and requesting the mail to be sent. An SMTP server configured to verify the connect request with a reverse DNS lookup will populate this field with the value returned by the DNS. Otherwise, this will contain the text for a decimal represented IPv4 address.
mailfrom	The sender's email address. This is the fully qualified email address from whom the message claims to be sent
rcptto	A list of comma delimited, fully qualified email addresses. The list in its entirety is enclosed in parentheses
authinfo	The fully qualified email address of the authenticated sender. This keyword/value pair is a part of the envelope only when the connection is authenticated. Optional.
domain	The helo or ehlo domain of the remote client or MTA.
resrcpt	A list of comma delimited, fully qualified resolved email addresses. The list in its entirety is enclosed in parentheses.

The header buffer will not include white space. The four keywords are in lower case; the order is always be the same (host, mailfrom, rcptto, [authinfo], domain). The "authinfo" field is optional, and is only provided if the remote system has authenticated the SMTP connection. The "rcptto=" list will always be enclosed in parentheses, even when the mail is only being delivered to a single recipient.

Note: When a plug-in reaches the INCOMING high-level event control point, the "rcptto" list will include all recipients. However, the list will be unresolved. The list will include email addresses, and public distribution lists. When the plug-in reaches either the OUTGOING or LOCAL high-level event control points, the "rcptto" list will be completely resolved but could be a subset of the complete recipient list. If OUTGOING is reached, only those recipients external to the system will be listed. If LOCAL is reached, only those recipients in local domains will be listed.

Note: Adding and deleting recipient addresses are only available at the INCOMING high-level event control point.

Framework functions available to plug-in developer:

ESEFIF_CALLBACK_FREE - Free Memory

ESEFIF_CALLBACK_WRITELOG - Log message to logfile

ESEFIF_CALLBACK_ALLOC - Allocate Memory

ESEFIF_CALLBACK_ADDRCPT - Add Recipient

ESEFIF_CALLBACK_DELRCPT - Delete Recipient

esefifMessage()

```
int esefifMessage(void *ifgctx,
                  unsigned int flags,
                  char *errmsg)
```

Purpose

- To read and process envelope, header, and message information.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

flags (IN)
Unused. Always zero.

errmsg (IN / OUT)
The mail protocol server passes a pointer to a 512-byte buffer holding error message text that would be returned to the remote client or process in the case when the plug-in needs to reject the message based upon envelope information.

Returns

```
ESEFIF_SUCCESS     Success - Allow Message
ESEFIF_REJECT      Failure - Reject Message
ESEFIF_MODIFIED    Allow the modified message
ESEFIF_ERROR       Error
```

esefifReset()

```
int esefifReset(void *ifgctx,
                unsigned int flags,
                char *errmsg)
```

Purpose

- To free the thread-specific local context if it is the end of the connection
- To clean up current message-specific data

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

flags (IN)
ESEFIF_RESET_MESSAGE The remote site has issued a temporary reset
ESEFIF_RESET_CLIENT This reset routine is being called at the end of a connection. The connection could be closed due to permanent network error (TCP/IP reset).

errmsg (IN / OUT)
The mail protocol server passes a pointer to a 512-byte buffer holding error message text that would be returned to the remote client or process in the case when the plug-in needs to reject the message.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

All framework functions are available to the plug-in developer at this service point.

Framework Functions

The plug-in framework provides various callback services to be used when developing plug-ins. Function pointers passed to the esefifRegistration() function are used to tell the plug-in code how to call these framework functions. Developers can make these functions globally available, either by defining global function pointers or by recording them in the global context for the plug-in. All framework functions share the same interface, each returns an integer, and each takes three void pointers as parameters.

```
int (*function)(void*, void*,void*);
```

ESEFIF_CALLBACK_SENDMSG - Send message

Purpose

- To tell the framework to send a copy of the message to the plug-in.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

iflctx (IN)
This is a pointer to the local context of the plug-in.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

ESEFIF_CALLBACK_RECVMSG - Receive message

Purpose

- To tell the framework to accept a copy of the message from the plug-in. Replaces a message with a cleaned or modified message.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

iflctx (IN)
This is a pointer to the local context of the plug-in.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

ESEFIF_CALLBACK_GETENVELOPE - Get envelope

Purpose

- To tell the framework to send the envelope information to the plug-in.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

envinfo (OUT)
This is a pointer to a buffer containing the envelope information. (See "envinfo" parameter in esefifEnvelope() for the buffer syntax).

eflctx (IN)
This is pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

This framework function implicitly asks the framework to allocate memory on behalf of the plug-in. The developer must free the memory "envinfo" points to if this function is called.

ESEFIF_CALLBACK_GETMSGID - Get mail store identifier of a message

Purpose

- To request from the framework the mail store identifier for the message.

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

msgid (OUT)
This is the mail store message identifier for the message.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

Message IDs are Oracle-specific and unique to the mail store. This ID is not the RFC822 GUID of the message. A message is retained only once in each mail store, along with instance records for all recipients of the message who use that mail store. If an email message is sent to two recipients, each on a different mail store, then two copies of the message are stored, one for each store. Each store would have a distinct and different message identifier for the message. Should a plug-in require a globally unique message ID, it should call the framework function for getting the message header and retrieve the RFC822 GUID from the header.

ESEFIF_CALLBACK_FREE - Free memory allocated by the framework

Purpose

- To request the framework to free any memory previously allocated by the framework for the plug-in

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

buffer (IN)
This is a pointer to the memory to be freed.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS      Success
ESEFIF_REJECT       Failure
```

Comments

ESEFIF_CALLBACK_SETVERSION - Set Version definition

Purpose

- To insert a record in the ES_SCAN_HISTORY table

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

definition (IN)
This is a character string that contains the definition of the knowledge base of virus definitions. Maximum length of 240 characters.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

Setting the history definition will create a record in the mail store schema. The record description is:

1. SCANNED - An Oracle date for when this framework function was called.
2. MSG_ID - The mail store message identifier for the message
3. SCANNER - The vendor name of the plug-in
4. DESCRIPTION - This text field

This record can be shared between plug-in instances on multiple mail protocol servers to ensure processing executes only once. For example, a plug-in that scans for viruses at the MTA as a message enters a mail store can create a record with definition text that describes the knowledge base of the virus definitions. The same vendor's plug-in could then scrub the mail store later on and bypass those messages which have already been checked with the same virus definition.

ESEFIF_CALLBACK_GETMSGHDR - Get message header

Purpose

- To request from the framework the top-level header information for the message

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

header (OUT)
This is a pointer to a buffer that contains the top-level header.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

Because the protocol server allocates the memory for this buffer, plug-ins that use this function must free the header buffer with the framework Free Memory function.

ESEFIF_CALLBACK_WRITELOG - Write entry in protocol server log file

Purpose

- To log a message in the mail protocol server log file

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

msg (IN)
This is a character string to be logged. The maximum length is 4000 bytes. The log string will have the filter library specified in the module parameter to identify which filter logged the message. The input string is logged as is.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

ESEFIF_CALLBACK_ALLOC - Allocate memory

Purpose

- To request the framework to allocate memory for the plug-in.

Parameters

buffer (OUT)
This is a pointer to the global context of the plug-in.

buflen (IN)
This is a pointer to the local context of the plug-in.

efgctx (IN)
This is a pointer to the global context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

ESEFIF_CALLBACK_ADDRCPT - Add address to recipient list

Purpose

- To add additional email addresses to the recipient list for a message (blind carbon copy)

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

addr (IN)
Fully qualified email address to be added to the recipient list.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

This function can be called multiple times. Each call will add an additional email address.

ESEFIF_CALLBACK_DELRCPT - Delete address from recipient list

Purpose

- To remove an email address from the recipient list

Parameters

ifgctx (IN)
This is a pointer to the global context of the plug-in.

ddr (IN)
This is a fully qualified email address to be removed from the recipient list.

eflctx (IN)
This is pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT      Failure
```

Comments

This function can be called multiple times, each call removing an additional email address.

This framework function only works on envelope information. An address removed from the envelope may still exist in the header of the mail message.

ESEFIF_CALLBACK_GETMSGSIZE - Get size of a mail message

Purpose

- To request from the framework the size in bytes of the mail message currently being processed.

Parameters

msgsize (OUT)
Message size in bytes.

eflctx (IN)
This is a pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT     Failure
```

Comments

The plug-in can determine the size of the message prior to obtaining the message from the framework. This size can be used or buffer allocation for the message. This function is applicable in the context of `esefifMessage()` and can be called multiple times.

ESEFIF_CALLBACK_GETPOLICYID - Get policy identifier of the archive plug-in

Purpose

- To request from the framework the policy identifier of the archive plug-in.

Parameters

ifgctx (IN)
A pointer to the global context of the plug-in.

policyid(OUT)
Archive policy identifier.

eflctx (IN)
A pointer to the local context of the framework.

Returns

```
ESEFIF_SUCCESS     Success
ESEFIF_REJECT     Failure
```

Comments

The policy identifier should be used to index built by plug-in from the list of policies passed in scanner flags during plug-in initialization to fetch current information related to this policy such as mailbox, xheader. If the plug-in is not associated with any archive policy, -1 is returned as the policy identifier.

Code Samples

Testing a Process Plug-In

To test a filter, run it as a standalone program and send sample message input using the terminal (*stdin*), with a ctrl-D character to end the input. Once testing is complete, you can configure and use this filter as an OCS filter. This method will reduce troubleshooting time if errors appear in the Mail Server logs due to problems such as incorrect permissions, incorrect reading of input parameters, or incorrect status line in the output from the filter program.

Code Listing for Sample Script

The following C-shell script logs the command line arguments, receives the message from Mail server, saves it to a file and accepts the message by passing a status code of 0. The log file and input message file should be different for different invocations of the executable and they will be created in the same directory as the executable. The directory must have write access for this executable to create these files.

```
#!/bin/csh
echo `date` >> $0.$$.log
echo "dummyscript started with arguments:" $* >> $0.$$.log
echo "Received Message on stdin:" >> $0.$$.msg

# save the message to a file
cat >> $0.$$.msg

# accept the message
echo "0"
echo `date` >> $0.$$.log
echo "dummyscript accepted the message" >> $0.$$.log

# send version info in the following way
# echo "0 [dummyscriptversion 1.0]"
exit 0
```

C Plug-in

The following code sample demonstrates how to construct a plug-in to be called by the Mail Protocol Server. The code provides a rudimentary implementation of each of the twelve framework callback functions.

```
/* Copyright (c) 2005, Oracle Corporation. All rights reserved. */
/*

NAME
    dummyif.c - An example dummy scanner/filter interface.

DESCRIPTION
    An example dummy scanner/filter interface to illustrate usage:
        - Implements services CONNECT, ENVELOPE, MESSAGE, RESET
        - Handles multiple messages in single connect.
        - Fetches envelope/header/message size/message id from MTA
        - Gets message from MTA.
        - Inserts a dummy header field; If repairmsg flag is on,
          posts this message back to MTA.
```

```
        - Sets Version definition.
        - Output is written to /tmp/dummyif.<pid>.out where pid is the
          MTA's process id.
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

/*-------------------------------------------------------------------------
   PRIVATE TYPES AND CONSTANTS
-------------------------------------------------------------------------*/
/* building shared library
** compile with -G option: cc -G dummyif.c -o dummyif.so
** use -g option for debugging: cc -g -G dummyif.c -o dummyif.so
*/

#define ESEF_VSCANIF_SUCCESS 0
#define ESEF_VSCANIF_FAIL 1
#define ESEF_VSCANIF_REPAIR 2
#define ESEF_VSCANIF_ERROR 3

typedef struct dummy_gcx
{
    /* MTA's call back functions */
    int (*frsend)(void*, void*,void*);
    int (*frrecv)(void*, void*,void*);
    int (*frgetenvl)(void*, char **,void*);
    int (*frgetmsgsize)(void*, int *,void*);
    int (*frgetmsgid)(void*, int *,void*);
    int (*frfreemem)(void*, void*,void*);
    int (*frsetversiondef)(void*, char *,void*);
    int (*frgethdr)(void*, char **,void*);
    int (*frlogmsg)(void*, char *,void*);
    int (*frallocmem)(void **, int ,void*);
    int (*fraddrcpt)(void*, char *,void*);
    int (*frdelrcpt)(void*, char *,void*);
    char services[1024];
    void *efgctx;
    char scanflags[256];
    char sysflags[256];
    unsigned int repair;
} dummy_gcx;

typedef struct dummy_lcx
{
    signed char *bufp;
    char *envbufp;
    int len;
    int pos;
    dummy_gcx *gctx;
    void *efcbctx;
} dummy_lcx;

FILE *fpout;


/*-------------------------------------------------------------------------
     STATIC FUNCTION DECLARATIONS
```

```
 ---------------------------------------------------------------------------*/


int esefifInit(
    void **ifgctx,
    void *efgctx,
    void **services,
    char *scanflags,
    char *sysflags)
{
    dummy_gcx *gctx;
    int pid;
    char outfile[256];

    pid = getpid();
    sprintf(outfile,"/tmp/dummyif.%d.out",pid);
    fpout = fopen(outfile,"w");
    if (!fpout) fpout=stdout;
    fprintf(fpout,"In esefifInit()\n");

    gctx = *ifgctx = (void *) calloc(1,sizeof(dummy_gcx));
    gctx->efgctx = efgctx;

    if (scanflags) strcpy(gctx->scanflags,scanflags);
    if (sysflags) strcpy(gctx->sysflags,sysflags);
    if (strstr(gctx->sysflags,"repairmsg=1")) gctx->repair = 1;
    fprintf(fpout,"scanflags=%s,sysflags=%s\n",gctx->scanflags,gctx->sysflags);

    sprintf(gctx->services,"E:CONNECT,ENVELOPE,MESSAGE,RESET");
    *services = gctx->services;

    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}

void esefifClose(void *ifgctx)
{
    fprintf(fpout, "In esefifClose()\n");
    fflush(fpout);

    free(ifgctx);

    fclose(fpout);
    return;
}


int esefifRegister(
    void *ifgctx,
    int (*cb)(void*, void*,void*),
    unsigned int flags)
{
    dummy_gcx *ifgctx1;

    fprintf(fpout,"In esefifRegister()\n");

    ifgctx1 = (dummy_gcx *) ifgctx;
    switch (flags)
    {
    case 0:
```

```
            ifgctx1->frsend = cb;
            break;
        case 1:
            ifgctx1->frrecv = cb;
            break;
        case 2:
            ifgctx1->frgetenvl = (int (*) (void*, char **,void*)) cb;
            break;
        case 3:
            ifgctx1->frgetmsgsize = (int (*) (void*, int *,void*)) cb;
            break;
        case 4:
            ifgctx1->frgetmsgid = (int (*) (void*, int *,void*)) cb;
            break;
        case 5:
            ifgctx1->frfreemem = cb;
            break;
        case 6:
            ifgctx1->frsetversiondef = (int (*) (void*, char *,void*)) cb;
            break;
        case 7:
            ifgctx1->frgethdr = (int (*) (void*, char **,void*)) cb;
            break;
        case 8:
            ifgctx1->frlogmsg = (int (*) (void*, char *,void*)) cb;
            break;
        case 9:
            ifgctx1->frallocmem = (int (*) (void**, int ,void*)) cb;
            break;
        case 10:
            ifgctx1->fraddrcpt = (int (*) (void*, char *, void*)) cb;
            break;
        case 11:
            ifgctx1->frdelrcpt = (int (*) (void*, char *, void*)) cb;
            break;
        default:
            return ESEF_VSCANIF_ERROR;
        }

        fflush(fpout);
        return ESEF_VSCANIF_SUCCESS;
    }

    int esefifConnect(
        void *ifgctx,
        void **iflctx,
        void *efcbctx,
        char *host,
        char *ip,
        int flags,
        char *errmsg)
    {
        dummy_lcx *iflctx1;
        int size=0;
        dummy_gcx *ifgctx1;

        fprintf(fpout,"In esefifConnect()\n");

        ifgctx1 = (dummy_gcx *) ifgctx;
        ifgctx1->frallocmem((void **)&(iflctx1),sizeof(dummy_lcx),ifgctx1->efgctx);
```

```
    memset(iflctx1,0, sizeof (dummy_lcx));
    iflctx1->gctx = ifgctx;;

    *iflctx = iflctx1;
    iflctx1->efcbctx = efcbctx;

    if (host)
        fprintf(fpout,"esefifConnect(): Host=%s\n",host);
    if (ip)
        fprintf(fpout,"esefifConnect(): IP=%s\n",ip);

    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}

int esefifEnvelope(
    void *iflctx,
    char *envinfo,
    unsigned int flags,
    char *errmsg)
{

    fprintf(fpout,"In esefifEnvelope()\n");
    if (envinfo) fprintf(fpout, "esefifEnvelope(): envinfo=%s\n",envinfo);

    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}

int esefifAuth(void *iflctx, char *authinfo, unsigned int flags, char *errmsg)
{

    fprintf(fpout,"In esefifAuth()\n");
    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}


int esefifMessage(void *iflctx, unsigned int flags, char *errmsg)
{
    int rc = ESEF_VSCANIF_SUCCESS;
    int msgsize, msgid, size=0;
    char versiondef[128] = "X-Dummy-VirusScan:01192004";
    char *hdr = NULL;
    dummy_gcx *gctx;
    dummy_lcx *lctx;


    fprintf(fpout,"In esefifMessage()\n");
    lctx = (dummy_lcx *)iflctx;
    gctx = lctx->gctx;

    /* SET DUMMY HEADER */
    gctx->frallocmem((void **)&(lctx->bufp), 128, gctx->efgctx);
    lctx->len = 0;
    lctx->pos = 0;
    sprintf((char *)&(lctx->bufp[lctx->len]), "X-Dummy-VirusScan: 1\n");
    lctx->len += 21;

    /* GET ENVELOPE */
```

```
fprintf(fpout, "esefifMessage(): calling MTA get envelope\n");
if (gctx->frgetenvl(gctx,&(lctx->envbufp),lctx->efcbctx))
{
    fprintf(fpout, "esefifMessage(): error in MTA get envelope\n");
    rc = ESEF_VSCANIF_ERROR;
    goto exit;
}
else
{
    fprintf(fpout, "esefifMessage(): Envelope Received=%s\n",lctx->envbufp);
    fflush(stdout);
    gctx->frfreemem(gctx,lctx->envbufp,gctx->efgctx);
    lctx->envbufp = NULL;
}

/* GET MESSAGE ID AND SIZE */
fprintf(fpout, "esefifMessage(): calling MTA msg size and msg Id\n");
if (gctx->frgetmsgsize(gctx,&msgsize,lctx->efcbctx))
{
    fprintf(fpout, "esefifMessage(): error in MTA get msg size\n");
    rc = ESEF_VSCANIF_ERROR;
    goto exit;
}
else
    fprintf(fpout, "esefifMessage(): MSG SIZE=%d\n",msgsize);

if (gctx->frgetmsgid(gctx,&msgid,lctx->efcbctx))
{
    fprintf(fpout, "esefifMessage(): error in MTA get msg id\n");
    rc = ESEF_VSCANIF_ERROR;
    goto exit;
}
else
    fprintf(fpout, "esefifMessage(): MSG ID=%d\n",msgid);

/* GET MESSAGE HEADER */
fprintf(fpout, "esefifMessage: calling MTA get header\n");
if (gctx->frgethdr(gctx, &hdr, lctx->efcbctx))
{
    fprintf(fpout, "esefifMessage(): error in MTA get header\n");
    rc = ESEF_VSCANIF_ERROR;
    goto exit;
}
if (hdr)
{
    fprintf(fpout, "esefifMessage(): Message Header=%s\n",hdr);
}

/* GET MESSAGE */
fprintf(fpout,"esefifMessage(): calling MTA send\n");
if (gctx->frsend(gctx, lctx, lctx->efcbctx))
{
    fprintf(fpout,"esefifMessage(): error in MTA send\n");
    rc = ESEF_VSCANIF_ERROR;
    goto exit;
}

if (gctx->repair)
/* if repair mode is on - post message with additional header field */
{
```

```
        /* MTA RECV */
        rc = 2;
        fprintf(fpout,"esefifMessage(): calling MTA recv\n");
        if (gctx->frrecv(gctx, lctx, lctx->efcbctx))
        {
            fprintf(fpout,"esefifMessage(): error in MTA recv\n");
            rc=ESEF_VSCANIF_ERROR;
            goto exit;
        }
    }

    /* set version def */
    if (gctx->frsetversiondef(gctx,versiondef,lctx->efcbctx))
    {
        fprintf(fpout,"esefifMessage(): error in MTA set version definition\n");
        rc=ESEF_VSCANIF_ERROR;
        goto exit;
    }

    exit:
        if (lctx->bufp)
        {
            gctx->frfreemem(gctx,lctx->bufp,gctx->efgctx);
            lctx->bufp = NULL;
        }
        if (lctx->envbufp) {
            gctx->frfreemem(gctx,lctx->envbufp,gctx->efgctx);
            lctx->envbufp = NULL;
        }

        /* if header contains "virus" treat it as one
           for negative testing */
        if (hdr)
        {
            if (strstr(hdr, "virus"))
            rc = ESEF_VSCANIF_FAIL;
            gctx->frfreemem(gctx,hdr,gctx->efgctx);
        }

        fflush(fpout);
        return rc;
}

int esefifSend(void *iflctx, char *buf, int buflen)
{
    dummy_lcx *lctx;
    dummy_gcx *gctx;
    signed char *bufp;

    fprintf(fpout,"In esefifSend()\n");

    lctx = (dummy_lcx *) iflctx;
    gctx = lctx->gctx;
    fprintf(fpout,"esefifSend(): buffer size=%d buffer=%s\n",buflen,buf);
    fflush(fpout);
    bufp = lctx->bufp;
    gctx->frallocmem((void **)&(lctx->bufp), (lctx->len+buflen),
        (lctx->gctx->efgctx));
    memcpy(lctx->bufp,bufp,lctx->len);
    gctx->frfreemem(gctx, bufp, gctx->efgctx);
```

```
    memcpy(&(lctx->bufp[lctx->len]), buf, buflen);

    lctx->len += buflen;

    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}


int esefifRecv(void *iflctx, char *buf, int *buflen)
{
    dummy_lcx *lctx;
    dummy_gcx *gctx;

    fprintf(fpout,"In esefifRecv()\n");

    lctx = (dummy_lcx *) iflctx;
    gctx = lctx->gctx;
    if (lctx->len > *buflen)
    {
        memcpy(buf, &(lctx->bufp[lctx->pos]), *buflen);
        lctx->len -= *buflen;
        lctx->pos += *buflen;
    }
    else if (lctx->len > 0)
    {
        memcpy(buf, &(lctx->bufp[lctx->pos]), lctx->len);
        *buflen = lctx->len;
        lctx->pos += lctx->len;
        lctx->len =0;
    }
    else
    {
        *buflen = 0;
        gctx->frfreemem(gctx,lctx->bufp,gctx->efgctx);
        lctx->bufp = NULL;
    }

    buf[*buflen] = '\0';
    fprintf(fpout,"esefifRecv(): buffer size=%d buffer=%s\n", *buflen, buf);

    fflush(fpout);
    return *buflen;
}


int esefifReset(void *iflctx, unsigned int flags, char *errmsg)
{
    dummy_lcx *lctx;
    dummy_gcx *gctx;

    fprintf(fpout,"In esefifReset()\n");
    lctx = (dummy_lcx *) iflctx;
    gctx = lctx->gctx;
    if (lctx->bufp)
    {
        gctx->frfreemem(gctx, lctx->bufp, gctx->efgctx);
        lctx->bufp = NULL;
    }
    if (lctx->envbufp)
```

```
    {
        gctx->frfreemem(gctx, lctx->envbufp, gctx->efgctx);
        lctx->envbufp = NULL;
    }
    lctx->len = 0;
    lctx->pos = 0;
    if (flags == 1)
    {
        gctx->frfreemem(gctx, lctx, gctx->efgctx);
    }
    fflush(fpout);
    return ESEF_VSCANIF_SUCCESS;
}

/* end of file dummyif.c */
```

Index

R

S

V

W

X