Oracle9*i*AS Unified Messaging

Application Developer's Guide

Release 9.0.2

January 2002

Part No. A95455-01

ORACLE

Oracle9*i*AS Unified Messaging Application Developer's Guide, Release 9.0.2

Part No. A95455-01

Primary Author: Ginger Tabora

Contributing Authors: Cynthia Kibbe

Contributors: Vicky Cao, Ashish Consul, Vikas Dhamija, Tanya Hitaisinee, Harvinder Walia, Anthony Ye

Contents

Send Us Your Comments

Oracle9*i*AS Unified Messaging Application Developer's Guide, Release 9.0.2

Part No. A95455-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev@us.oracle.com
- FAX: (650) 506-7228 Attn: Oracle9*i*AS Unified Messaging Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i*AS Unified Messaging Documentation Manager
  500 Oracle Parkway, 4OP11
  Redwood City, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Intended Audience

This *Application Developer's Guide* primarily addresses the application developer audience. It provides an introduction to Oracle9*i*AS Unified Messaging and describes the management tasks an Oracle9*i*AS Unified Messaging server administrator performs. PL/SQL programming knowledge is also helpful in implementing PL/SQL APIs and server-side rules.

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, 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, JAWS 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

Chapter 1, "PL/SQL API Reference"

This chapter contains the set of Oracle9*i*AS Unified Messaging PL/SQL APIs that can be used to access and manage e-mail on the Oracle9*i*AS Unified Messaging system.

Chapter 2, "JAVA API Reference"

This chapter contains the JAVA APIs provided for Oracle9*i*AS Unified Messaging.

Related Documents

Oracle9*i*AS Unified Messaging documentation is available in HTML and PDF.

The following document is available on the Oracle9*i*AS documentation library:

- *Oracle9iAS Unified Messaging Application Developer's Guide*

The following documents are available on http://otn.oracle.com

- *Oracle9iAS Unified Messaging Administrator's Guide*
- *Oracle9iAS Unified Messaging User's Guide*
- *Oracle9iAS Unified Messaging JAVA API Documentation*

Conventions

The following conventions are used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.

Convention	Meaning
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

PL/SQL API Reference

The Oracle9*i*AS Unified Messaging PL/SQL APIs are a set of application program interfaces that can be used to access and manage e-mail on the Oracle9*i*AS Unified Messaging system.

This chapter contains the following topics:

- Overview
- Concepts
- Mail Objects
- MAIL_SESSION Package
- MAIL_FOLDER Package
- MAIL_MESSAGE Package
- Exceptions
- Examples

Overview

The APIs described in this chapter expose Oracle9*i*AS Unified Messaging system functionality that can be customized to suit your business and application requirements.

This release includes the following PL/SQL packages:

- MAIL_SESSION
- MAIL_FOLDER
- MAIL_MESSAGE

MAIL_SESSION Package

The MAIL_SESSION package provides functions for authenticating and logging out users.

MAIL_FOLDER Package

The MAIL_FOLDER package provides access to the Oracle9*i*AS Unified Messaging server message store, enabling folder management and message list operations.

Folder management operations are:

- List folder
- Create folder
- Delete folder
- Rename folder
- Expunge folder

Message list operations are:

- Retreive message objects
- Copy messages
- Change message flags
- Delete messages
- Search messages
- Sort messages

MAIL_MESSAGE Package

The MAIL_MESSAGE package provides functions for accessing messages. It enables users to do the following:

- Retrieve message attributes, headers, structures and content
- Change message flags
- Compose new messages for sending and appending
- Retrieve message highlight, themes, markups, and filtered text

Concepts

This section discusses the following e-mail related concepts:

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

Folder UIDL

The folder UIDL is a numeric value that uniquely identifies a folder in a session. The value increments in the case where the messages belonging to this folder contain UIDs that must be compacted. A possible reason is the message UID has reached a number greater than 2^32. In this case, the folder UIDL changes to a larger value that is unique among all the user's folders. In addition, the message UIDs belonging to this folder are compacted starting from one.

Message UID

The message UID is a 32-bit value that identifies each message. This value is combined with the value of the folder UIDL to form a unique value that guarantees the correct identification of each message in the folder.

Message UIDs are assigned in ascending order in the folder. As each message is added to the folder, it is assigned a higher UID value. The value persists across sessions.

Message Flags

Message flags are attributes of a message. The following table lists the supported flags and describes their values when set:

MAIL_MESSAGE.GC_SEEN_FLAG	Indicates that the message has already been read
MAIL_MESSAGE.GC_FLAGGED_FLAG	Indicates that the message is flagged for urgent or special attention
MAIL_MESSAGE.GC_ANSWERED_FLAG	Indicates that the message has been replied to
MAIL_MESSAGE.GC_DELETED_FLAG	Indicates that the message is marked to be removed later with the expunge command
MAIL_MESSAGE.GC_DRAFT_FLAG	Indicates that the message has not completed composition

Recent Message

A message is considered recent if no mail client session has seen the message.

New Message

A message is considered new if the current session has not seen the message since the last message was retrieved from this folder.

When the folder is first opened, the last message considered retrieved is the last message seen by any client session. After that, the last message is the one retrieved by the previous GET_FOLDER_MESSAGES and GET_NEW_MESSAGES procedure calls in the current session.

MIME Level

A message MIME level is a string used to identify a specific part of a message. The procedure returns the message and body-part objects with the MIME level properly set.

Mail Objects

The PL/SQL API uses the following public mail objects:

- MAIL_FOLDER_OBJ
- MAIL_FOLDER_DETAIL

- MAIL_SORT_CRITERIA_ELEMENT
- MAIL_MESSAGE_OBJ
- MAIL_BODYPART_OBJ
- VARCHAR2_TABLE
- VARCHAR2_TABLE

Note: This SDK 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
);
```

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.

`content_type` contains the main Content-Type header value for this body-part.

`smime_ind` a value of 1, indicates that the body-part is from a decrypted message.

`mail_bodypart_list` is a nested table of `MAIL_BODYPART_OBJ` objects.

VARCHAR2_TABLE

`VARCHAR2_TABLE` defines a nested table of a variable-length character string that has a maximum length of 2000 as follows:

```
VARCHAR2_TABLE IS TABLE OF VARCHAR2(2000);
```

MAIL_HEADER_OBJ

The MAIL_HEADER_OBJ object is used to store a message header name and value pair. It is defined as follows:

```
MAIL_HEADER_OBJ (
 header_prompt     VARCHAR2(1000),
 header_value      VARCHAR2_TABLE
);
```

header_prompt is the name of the header.

header_value is the value of the header. It is a VARCHAR2_TABLE, which is a table of 2000 variable-length character strings. When the length of header value is greater than 2000, it is broken into multiple strings of 2000 length.

MAIL_HEADER_LIST is a nested table of MAIL_HEADER_OBJ objects:

```
MAIL_HEADER_LIST IS TABLE OF MAIL_HEADER_OBJ;
```

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 the user's user name and password.

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:

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 LDAP server is configured
ldap_port	The port LDAP server is listening to; the default is 389
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:

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

GET_CURRENT_USAGE Procedure

This procedure retrieves the user's current mail usage.

Throws Exceptions:

mail_errors.unauthenticated_err

Syntax:

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

Parameters:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
usage	The number of bytes used by the user

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
- 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_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
- 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
- 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 Exceptions:

```
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:

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

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:

Parameter	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:

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 Exceptions:

mail_errors.unauthenticated_err

Syntax:

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

Parameters:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
foldername_list	A list of top-level subscribed folders

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:

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:

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:

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:

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

Parameter	Description
folder_name	The full path of the folder name

HAS_FOLDER_CHILDREN Function

This function tests to see if any child folders exist.

Throws Exceptions:

mail_errors.unauthenticated_err

Syntax:

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

Parameters:

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

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:

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 with the given name, returning a folder object representing it. A FOLDER_TYPE_ERR is returned if the parent folder does not enable any subfolder creation.

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:

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 the DELETE_FOLDER procedure is called with a folder that is marked as not selectable and contains a subfolder, the MAIL_ERRORS.FOLDER_TYPE_ERR is 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:

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. Renaming the inbox moves all of the messages in the inbox to a new folder, leaving the inbox folder empty.

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:

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 opened successfully. If the folder is marked with a NOSELECT flag, a MAIL_ERRORS.FOLDER_TYPE_ERR is thrown. If a folder has already been opened prior

to this call, it is closed without being expunged (messages with the DELETED message flags are not removed from that folder).

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:

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

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:

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:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_uid	The message identifier

Parameter	Description
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:

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
);
```

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

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:

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 of flags.

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
);
PROCEDURE delete_messages (
session_id        IN  NUMBER,
message_list      IN  MAIL_MESSAGE_LIST
);
```

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
message_list	A list of message objects that belongs to 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:

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:

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:

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. 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

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 Exceptions:

```
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:

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

Parameter	Description
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, then 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:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_list	A list of new message objects

Parameter	Description
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:

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

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:

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:

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 criterion. 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.

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:

Parameter	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 message retreival 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_COTENTMD5 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_MSGS_FLAGS Procedure
- SET_MSGS_FLAGS Procedure

- GET_AUTH_INFO Procedure

The MAIL_MESSAGE package contains the following message composition procedures:

- COMPOSE_MESSAGE Procedure
- SET_MSGHEADER Procedure
- SET_BPHEADER Procedure
- SET_HEADER Procedure
- ADD_BODYPART Procedure
- ADD_INCLMSG_BODYPART Procedure
- SET_INCLMSG_BODYPART Procedure
- SET_CONTENT Procedure
- SEND_MESSAGE Procedure
- APPEND_MESSAGE Procedure
- DECRYPT_MESSAGE Procedure
- VERIFY_MESSAGE Procedure

The MAIL_MESSAGE package contains the following Oracle Text procedures:

- GET_THEMES Procedure
- GET_THEMES Procedure
- GET_HIGHLIGHT Procedure
- GET_MARKUPTEXT Procedure
- GET_FILTERED_TEXT Procedure
- GET_TOKENS 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:

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 Content-Type; otherwise the MAIL_ERRORS.PARAM_PARSE_ERR exception is thrown.

Throws Exceptions:

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

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 2000 in length if the length of the value is longer than 2000. If there are multiple headers with the same prompt, the returned value is the first header. When the returnedvalue is a MAIL_HEADER_LIST object list, all headers with the specified prompt name are returned. Each name and value pair is represented by a MAIL_HEADER_OBJ object.

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:

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_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 Exceptions:

```
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:

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

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 Exceptions:

```
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:

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:

Parameter	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:

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:

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 Exceptions:

```
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:

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:

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:

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_COTENTMD5 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:

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:

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

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:

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:

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:

Parameter	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:

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:

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:

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:

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:

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

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:

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_MSGS_FLAGS Procedure

This procedure returns the message's flags.

Throws Exceptions:

```
mail_errors.unauthenticated_err
mail_errors.folder_closed_err
```

Syntax:

```
PROCEDURE get_msgs_flags (
session_id    IN  NUMBER,
message_obj   IN  MAIL_MESSAGE_OBJ,
message_flags OUT NUMBER);
```

Parameters:

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

SET_MSGS_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_msgs_flags (
session_id    IN  NUMBER,
message_obj   IN  MAIL_MESSAGE_OBJ,
```

```
message_flags IN  NUMBER,
set_flag      IN  BOOLEAN);
```

Parameters:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
message_obj	A message object
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
set_flag	If true, set the specified flags ; otherwise, unset 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:

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

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:

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:

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-Encodin 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:

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
filname	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:

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 is not the same type of multipart message, 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:

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 is not the same type of message, 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:

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 is not the same type of message, 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:

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:

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:

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
sign_algorithm	The algorithm for signing. It is not used if the message is only being encrypted. Values are: ■ MAIL_MESSAGE.GC_RSA ■ MAIL_MESSAGE.GC_DSA
encrypt_ algorithm	The algorithm for encryption. It is not used if the message only being signed. Values are: ■ MAIL_MESSAGE.GC_DES_EDE3_CBC ■ MAIL_MESSAGE.GC_DES_CBC ■ MAIL_MESSAGE.GC_RC2_CBC_128 ■ MAIL_MESSAGE.GC_RC2_CBC_40

Parameter	Description
send_option	Indicates whether to send the messagesigned or encrypted or both. Values are: ■ MAIL_MESSAGE.GC_SEND_SIGNED ■ MAIL_MESSAGE.GC_ENCRYPTED ■ MAIL_MESSAGE.GC_SEND_SIGNED | MAIL_MESSAGE.GC_ENCRYPTED

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.unathenticated_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:

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

Parameter	Description
folder_obj	The folder to which the message tis 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

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:

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,
retruned_content OUT BLOB
);
```

Parameters:

Parameter	Description
session_id	An identifier that represents a user's authenticated session
certificate	The user's certificate
private_key	The user's private key

Parameter	Description
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_erros.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:

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

Parameter	Description
bodypart_obj	The body-part object
flags	Currently not used
incl_binary_parts	If false, 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:

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, 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:

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, non-text part is ignored
tag_set	Refer to Oracle Text documentation
star_ttag	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_erros.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:

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

Parameter	Description
flags	The returned buffer format. Values are: ■ MAIL_MESSAGE.GC_TEXT_FORMAT ■ MAIL_MESSAGE_GC_HTML_FORMAT
incl_binary_parts	If false, 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_erros.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:

Parameter	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. Refer to NLS documentation for all possible languages.
incl_binary_parts	If false, non-text part is ignored
token_buffer	The token buffer

Exceptions

This section describes the following exceptions for the PL/SQL APIs:

- external_rule_err EXCEPTION
- external_cond_err EXCEPTION
- too_many_rules EXCEPTION
- sql_err EXCEPTION
- imt_err EXCEPTION
- bad_message_var EXCEPTION
- bad_msgpart_var EXCEPTION
- no_binary_err EXCEPTION
- unauthenticated_err EXCEPTION
- folder_closed_err EXCEPTION
- msg_compose_limit_err EXCEPTION
- folder_not_found_err EXCEPTION
- folder_already_exists_err EXCEPTION
- operation_not_allowed EXCEPTION
- param_parse_err EXCEPTION
- internal_err EXCEPTION

- folder_name_err EXCEPTION
- login_err EXCEPTION
- folder_type_err EXCEPTION
- smime_err EXCEPTION

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 of binary type but the option given to the API specifies with binary to be false.
Action:	Set the `withbinary` option to be true for Oracle Text information on binary type message part

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 to 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 (eg. x rename to x/y)
Action:	User should not try to re-arrange a folder hierachy with the rename operation. User should use the create and delete operations to achieve the desired folder hierachy.

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 ■ Unknown search criteria ■ The header value exceeds the 2000 length limit ■ Trying to create a folder with null foldername
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:	LDAP Login Error: %s
Cause:	Exception is raised when an invalid username or password is specified
Action:	Check your spelling and try again

folder_type_err EXCEPTION

Error No:	20212
Message:	Folder type voilation: %s
Cause:	Possible cause for this error includes: ■ 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

Examples

This section gives examples of how to log in and use the create, list, search, fetch, compose, send functions on folders. It also has an example of how to use the `GetTheme` API.

This section contains the following topics:

- Login, Create, List, and Search Example
- Login and Fetch All Example
- Compose and Send Example
- GetTheme Example

Login, Create, List, and Search Example

This example shows how to log in, create, list, and search on a folder.

```
set serveroutput on size 1000000;

DECLARE
   user_name            VARCHAR2(50) := 'testuser1';
   user_pswd            VARCHAR2(50):= 'welcome';
   user_domain          VARCHAR2(50) := 'oracle.com';
   ldap_server          VARCHAR2(50) := 'test-sun.us.oracle.com';
   ldap_port            NUMBER := 389;

   sessionID            NUMBER;
   my_folder_obj        MAIL_FOLDER_OBJ;
   my_folder_list       MAIL_FOLDER_LIST;
   inbox_obj            MAIL_FOLDER_OBJ;
   search_string        VARCHAR2(500);
   message_list         MAIL_MESSAGE_LIST;

   -- declare and define two procedures to recursively print out all folders
   PROCEDURE print_folder(session_id     IN  NUMBER,
                          folder_obj     IN  MAIL_FOLDER_OBJ);
   PROCEDURE print_folderlist(session_id IN  NUMBER,
                              flist      IN  MAIL_FOLDER_LIST);

   PROCEDURE print_folder(session_id     IN  NUMBER,
                          folder_obj     IN  MAIL_FOLDER_OBJ) IS
        flist   MAIL_FOLDER_LIST;
   BEGIN
      -- print out the folder name
      dbms_output.put_line(folder_obj.name);

      -- if there are child-folder, print them too
      if mail_folder.has_folder_children(session_id, folder_obj) then
         -- get all the child-folders and prin them
         mail_folder.list_folders(session_id, folder_obj, flist);
         print_folderlist(flist);
```

```
      end if;
   END;

   PROCEDURE print_folderlist(session_id IN  NUMBER,
                              flist      IN  MAIL_FOLDER_LIST) IS
   BEGIN
      for i in 1..flist.count loop
         print_folder(session_id, flist(i));
      end loop;
   END;

BEGIN
   -- login
   mail_session.login(user_name, user_pswd, user_domain,
                      ldap_server, sessionID, ldap_port);
   dbms_output.put_line(user_name || ' logged-in');

   -- create a new folder called my_folder
   mail_folder.create_folder(sessionID, 'my_folder', my_folder_obj);
   dbms_output.put_line('Created my_folder folder');

   -- list all folders
   mail_folder.list_toplevel_folders(sessionID, my_folder_list);
   dbms_output.put_line('My folder list:');
   print_folderlist(sessionID, my_folder_list);

   -- open INBOX
   mail_folder.open_folder(sessionID, 'INBOX', inbox_obj);
   dbms_output.put_line('Opened INBOX');

   -- search on INBOX for mails from Crosby with subject containing the word
   -- "White Christmas"
   search_string := 'from Crosby subject "White Christmas"';
   mail_folder.search_folder(sessionID, inbox_obj, search_string, message_list);
   dbms_output.put_line('Search for ' || search_string || ' returns:');
   FOR i IN 1..message_list.COUNT LOOP
      dbms_output.put(message_list(i).msg_uid || ' ');
   END LOOP;
   dbms_output.new_line;

   -- logout
   mail_session.logout(sessionID);
   dbms_output.put_line('Elvis has left the building!');

   -- commit
```

```
   commit;
EXCEPTION
   WHEN mail_errors.login_err THEN
      dbms_output.put_line('Failed to log-in!!!');
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
   WHEN mail_errors.folder_already_exists_err OR
        mail_errors.folder_type_err THEN
      dbms_output.put_line('Failed to create folder!!!');
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
   WHEN OTHERS THEN
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
END;
/
```

Login and Fetch All Example

This example shows how to log in and fetch all messages from a folder.

```
set serveroutput on size 1000000;

DECLARE
   user_name            VARCHAR2(50) := 'testuser1';
   user_pswd            VARCHAR2(50):= 'welcome';
   user_domain          VARCHAR2(50) := 'oracle.com';
   ldap_server          VARCHAR2(50) := 'test-sun.us.oracle.com';
   ldap_port            NUMBER := 389;

   sessionID            NUMBER;
   inbox_obj            MAIL_FOLDER_OBJ;
   message_list         MAIL_MESSAGE_LIST;


   -- declare and define some sub-programs to print out a message
   PROCEDURE print_message (sessionId   IN  NUMBER,
                            p_msg_obj   IN  MAIL_MESSAGE_OBJ);

   PROCEDURE print_bodypart (sessionId  IN  NUMBER,
                             p_bp_obj   IN  MAIL_BODYPART_OBJ);
```

```
PROCEDURE print_header (sessionId    IN  NUMBER,
                        p_msg_obj    IN  MAIL_MESSAGE_OBJ);

PROCEDURE print_header (sessionId    IN  NUMBER,
                        p_bp_obj     IN  MAIL_BODYPART_OBJ);

PROCEDURE print_content (sessionId          IN  NUMBER,
                         p_msg_obj          IN  MAIL_MESSAGE_OBJ,
                         p_content_type     IN  varchar2);

PROCEDURE print_content (sessionId          IN  NUMBER,
                         p_bp_obj           IN  MAIL_BODYPART_OBJ,
                         p_content_type     IN  varchar2);


PROCEDURE print_message (sessionId   IN  NUMBER,
                         p_msg_obj   IN  MAIL_MESSAGE_OBJ) IS
   bp_list           MAIL_BODYPART_LIST;
   content_type      varchar2(500);
   incl_msg          MAIL_MESSAGE_OBJ;
BEGIN
   -- print out the message header
   print_header(sessionId, p_msg_obj);

   -- get the message's content-type
   mail_message.get_content_type(sessionId, p_msg_obj, content_type);
   content_type := upper(content_type);

   -- if it is a multipart type, get each of its body-parts and print
   -- it out one by one
   if content_type like 'MULTIPART/%' then
      mail_message.get_multipart_bodyparts(sessionId, p_msg_obj, bp_list);
      for i in 1..bp_list.count loop
         print_bodypart(sessionId, bp_list(i));
      end loop;

   -- if it is a message type, get the included message and print
   -- the included message
   elsif content_type like 'MESSAGE/%' then
      mail_message.get_included_message(sessionId, p_msg_obj, incl_msg);
      print_message(sessionId, incl_msg);

   -- if it is a simple type, print the content
   else
      print_content(sessionId, p_msg_obj, content_type);
```

```
   end if;
END;

PROCEDURE print_bodypart (sessionId  IN  NUMBER,
                          p_bp_obj   IN  MAIL_BODYPART_OBJ) IS
   bp_list           MAIL_BODYPART_LIST;
   content_type      varchar2(500);
   incl_msg          MAIL_MESSAGE_OBJ;
BEGIN
   -- print out the body-part header
   print_header(sessionId, p_bp_obj);

   -- get the body-part's content-type
   mail_message.get_content_type(sessionId, p_bp_obj, content_type);
   content_type := upper(content_type);

   -- if it is a multipart type, get each of its body-parts and print
   -- it out one by one
   if content_type like 'MULTIPART/%' then
      mail_message.get_multipart_bodyparts(sessionId, p_bp_obj, bp_list);
      for i in 1..bp_list.count loop
         print_bodypart(sessionId, bp_list(i));
      end loop;

   -- if it is a message type, get the included message and print
   -- the included message
   elsif content_type like 'MESSAGE/%' then
      mail_message.get_included_message(sessionId, p_bp_obj, incl_msg);
      print_message(sessionId, incl_msg);

   -- if it is a simple type, print the content
   else
      print_content(sessionId, p_bp_obj, content_type);
   end if;
END;

--
-- private procedure to print header given a MAIL_HEADER_LIST object
--
PROCEDURE print_hdrlist (hdr_list    IN  MAIL_HEADER_LIST)
BEGIN
   dbms_output.put_line('[# message hdr: ' || hdr_list.count || ']');
   for i in 1..hdr_list.count loop
      dbms_output.put(hdr_list(i).header_prompt || ': ');
      for j in 1..hdr_list(i).header_value.count loop
```

```
         dbms_output.put_line(hdr_list(i).header_value(j));
      end loop;
   end loop;
END;

PROCEDURE print_header (sessionId    IN  NUMBER,
                        p_msg_obj    IN  MAIL_MESSAGE_OBJ) IS
   hdr_list          MAIL_HEADER_LIST;
   msg_size          NUMBER;
   msg_flags         NUMBER;
   msg_rcvd_date     VARCHAR2(50);
BEGIN
   if p_msg_obj.mime_level = '0' then        -- top level message
      mail_message.get_msg_size(sessionId, p_msg_obj, msg_size);
      mail_message.get_msg_flags(sessionId, p_msg_obj, msg_flags);
      mail_message.get_received_date(sessionId, p_msg_obj,
                                     'DD-Mon-YYYY HH24:MI:SS',
                                     msg_rcvd_date);

      dbms_output.put_line('[MSG_SIZE:' || msg_size || ']' ||
                           ' [MSG_FLAG:' || msg_flags || ']');
      dbms_output.put_line('[message header]:');
   else
      dbms_output.put_line('[included message header]:');
   end if;

   -- get all the headers for this message and print them
   mail_message.get_headers(sessionId, p_msg_obj, hdr_list);
   print_hdrlist(hdr_list);

   if p_msg_obj.mime_level = '0' then        -- top level message
     dbms_output.put_line('Received: '|| msg_rcvd_date);
   end if;

   dbms_output.new_line;
END;

PROCEDURE print_header (sessionId    IN  NUMBER,
                        p_bp_obj     IN  MAIL_BODYPART_OBJ) IS
   hdr_list          MAIL_HEADER_LIST;
BEGIN
   -- get all the headers for this body-part and print them
   dbms_output.put_line('[bodypart header]:');
   mail_message.get_headers(sessionId, p_bp_obj, hdr_list);
   print_hdrlist(hdr_list);
```

```
   dbms_output.new_line;
END;

PROCEDURE print_content (sessionId           IN  NUMBER,
                         p_msg_obj           IN  MAIL_MESSAGE_OBJ,
                         p_content_type      IN  varchar2) IS
   msg_data          BLOB;
   l_data            RAW(255);
   l_length          NUMBER;
   l_offset          NUMBER;
   l_size            NUMBER;
BEGIN
   -- create a temporary lob to hold the message body
   dbms_lob.createtemporary(msg_data, TRUE, dbms_lob.session);

   -- get the decoded message body
   mail_message.get_msg_body(sessionId, p_msg_obj, msg_data);
   l_length := dbms_lob.getlength(msg_data);
   dbms_output.put_line('[read ' || l_length || ']');
   dbms_output.new_line;

   -- print the decoded message body if it is a text type
   if p_content_type like 'TEXT/%' then
      l_offset := 1;
      while l_offset <= l_length loop
         -- sqlplus has a limitation of 255 characters per line.
         -- break the content into 250-character chunks and print.
         if (l_length - l_offset + 1) <= 250 then
            l_size := l_length - l_offset + 1;
         else
            l_size := 250;
         end if;
         l_data := dbms_lob.substr(msg_data, l_size, l_offset);
         l_offset := l_offset + l_size;
         dbms_output.put_line(utl_raw.cast_to_varchar2(l_data));
      end loop;
   end if;

   -- free the temporary lob allocated
   dbms_lob.freetemporary(msg_data);
END;

PROCEDURE print_content (sessionId           IN  NUMBER,
                         p_bp_obj            IN  MAIL_BODYPART_OBJ,
                         p_content_type      IN  varchar2) IS
```

```
   bp_data           BLOB;
   l_data            RAW(255);
   l_length          NUMBER;
   l_offset          NUMBER;
   l_size            NUMBER;
BEGIN
   -- create a temporary lob to hold the body-part content
   dbms_lob.createtemporary(bp_data, TRUE, dbms_lob.session);

   -- get the decoded body-part content
   mail_message.get_bodypart_content(sessionId, p_bp_obj, bp_data);
   l_length := dbms_lob.getlength(bp_data);
   dbms_output.put_line('[read ' || l_length || ']');
   dbms_output.new_line;

   -- print the decoded data if it is a text type
   if p_content_type like 'TEXT/%' then
      l_offset := 1;
      while l_offset <= l_length loop
         -- sqlplus has a limitation of 255 characters per line.
         -- break the content into 250-character chunks and print.
         if (l_length - l_offset + 1) <= 250 then
            l_size := l_length - l_offset + 1;
         else
            l_size := 250;
         end if;
         l_data := dbms_lob.substr(bp_data, l_size, l_offset);
         l_offset := l_offset + l_size;
         dbms_output.put_line(utl_raw.cast_to_varchar2(l_data));
      end loop;
   end if;

   -- free the temporary lob allocated
   dbms_lob.freetemporary(bp_data);
END;

-- define a bit_on function
FUNCTION bit_on (
     p_flag  IN  INTEGER,
     p_bit   IN  INTEGER
) RETURN INTEGER IS
BEGIN
   IF bitand(p_flag, p_bit) = 0 THEN         -- bit not turned on yet
      RETURN p_flag + p_bit;
   ELSE                                      -- bit already on
```

```
         RETURN p_flag;
      END IF;
   END;


BEGIN
   -- login
   mail_session.login(user_name, user_pswd, user_domain,
                      ldap_server, sessionID, ldap_port);
   dbms_output.put_line(user_name || ' logged-in');

   -- open INBOX
   mail_folder.open_folder(sessionID, 'INBOX', inbox_obj);
   dbms_output.put_line('Opened INBOX');

   -- get all the messages from INBOX
   mail_folder.get_folder_messages(sessionId, message_list);
   dbms_output.put_line('Got ' || message_list.COUNT || ' messages');
   FOR i IN 1..message_list.COUNT LOOP
      dbms_output.put_line('***** Message # ' || i || ' *****');
      print_message(sessionId, message_list(i));

   END LOOP;

   -- set all the messages flags to be seen and deleted
   mail_folder.set_msg_flags(sessionId, message_list,
             bit_on(MAIL_MESSAGE.GC_SEEN_FLAG, MAIL_MESSAGE.GC_DELETED_FLAG),
             true);

   -- expunge INBOX to remove all the messages we have fetched
   mail_folder.expunge_folder(sessionId);

   -- logout
   mail_session.logout(sessionID);
   dbms_output.put_line('Elvis has left the building!');

   -- commit
   commit;
EXCEPTION
   WHEN mail_errors.login_err THEN
      dbms_output.put_line('Failed to log-in!!!');
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
   WHEN OTHERS THEN
```

```
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
END;
/
```

Compose and Send Example

This example shows how to compose and send a multipart/alternative message.

```
set serveroutput on size 1000000;

DECLARE
   to_addr              VARCHAR2(100) := 'testuser2@oracle.com';
   from_addr            VARCHAR2(100) := 'testuser1@oracle.com';
   subject_str          VARCHAR2(100) :=
        'Example: send a multipart/alternative message';

   msg_obj              mail_message_obj;
   a_bodypart           mail_bodypart_obj;
   b_bodypart           mail_bodypart_obj;
   msg_data             VARCHAR2(500);

BEGIN
   -- start composing a message
   mail_message.compose_message(msg_obj);
   dbms_output.put_line('start composing...');

   -- set the message's headers
   mail_message.set_msgheader(
        msg_obj, to_addr, from_addr, null, from_addr, sysdate,
        subject_str, '1.0', 'multipart/alternative', null, null
   );
   dbms_output.put_line('set message header');

   -- add the text/plain body-part
   mail_message.add_bodypart(msg_obj, a_bodypart);
   mail_message.set_bpheader(
        a_bodypart, 'text/plain', 'us-ascii', 'quoted-printable',
        null, null, null, null, null, null
   );
   msg_data := 'text/plain data';
   mail_message.set_content(a_bodypart, utl_raw.cast_to_raw(msg_data));
   dbms_output.put_line('set text/plain body');
```

```
   -- add the text/html body-part
   mail_message.add_bodypart(msg_obj, b_bodypart);
   mail_message.set_bpheader(
        b_bodypart, 'text/html', 'us-ascii', 'quoted-printable',
        null, null, null, null, null, null
   );
   msg_data := '<html>text/html data</html>';
   mail_message.set_content(b_bodypart, utl_raw.cast_to_raw(msg_data));
   dbms_output.put_line('set text/html body');

   -- now send this message
   mail_message.send_message(msg_obj);
   dbms_output.put_line('send message');

   -- commit
   commit;
EXCEPTION
   when OTHERS then
      dbms_output.put_line ('send failed!');
      dbms_output.put_line('SQLcode: ' || sqlcode);
      dbms_output.put_line('SQLerrm: ' || sqlerrm);
      rollback;
END;
/
```

GetTheme Example

This example shows how to use the GetTheme API.

```
ACCEPT msgid NUMBER DEFAULT 1 PROMPT 'Input message id: ';

set serveroutput on
DECLARE
  themebuf CTXSYS.CTX_DOC.THEME_TAB;
  errmsg   VARCHAR2(254);
  res      INTEGER;
  i        INTEGER;
BEGIN

  res := ES_OT_API.GetThemes(&msgid, '0' , 1,false, themebuf, errmsg);
  IF res != ES_OT_API.ESOTAPI_OK THEN
     DBMS_OUTPUT.PUT_LINE('GetTheme error no: ' || res);
     DBMS_OUTPUT.PUT_LINE(errmsg);
  ELSE
     dbms_output.put_line('found '|| themebuf.count ||' themes');
```

```
    FOR i in 1..themebuf.count LOOP
      DBMS_OUTPUT.PUT_LINE(' (' || themebuf(i).theme || ', ' ||
                           themebuf(i).weight || ') ');
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Get ' || themebuf.count || ' themes succeed');
  END IF;

EXCEPTION
  WHEN no_data_found THEN
    DBMS_OUTPUT.PUT_LINE('ERROR:' || errmsg);
END;
/
```

2

JAVA API Reference

This chapter describes the JAVA APIs provided for Oracle9*i*AS Unified Messaging.

This chapter contains the following topics:

- JavaMail API
- Directory Management API
- Rule Management API

> **See Also:** The *Oracle9i*AS Unified Messaging JAVA API Documentation on http://otn.oracle.com/products/ias, for information about the JAVA APIs

JavaMail API

The Oracle Javamail Service Provider (OJMA) implements the standard interfaces available with the Sun Javamail API(s) version 1.2. In addition, OJMA provides integration with Oracle Text, integration with Oracle S/MIME toolkit, support for Shared Folders, and support for Server Side Rules. The Javamail APIs can be used to authenticate the user, and perform folder and message operations. OJMA is integrated with Oracle Internet Directory (OID) and provides customized methods in the OracleStore class to authenticate mail users.

> **See Also:** The *Oracle9i*AS Unified Messaging JAVA API Documentation on http://otn.oracle.com, for information about the OJMA Extensions

> **See Also:** The Javamail Web site for Javamail documentation

This section provides the following examples for using the JavaMail API:

- Reading a User's Messages
- Creating a Shared Folder and Granting User Permissions
- Appending Simple Messages
- Basic Folder Operations
- Shared Folder and Message Fetching

Reading a User's Messages

The following is an example of how to read all the messages for a particular user's inbox:

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;
import oracle.mail.ldap.ESDSConstants;
import oracle.mail.sdk.esmail.OracleAuthenticator;

public class ReadMessage
{
        static String password = null;
        static String user = null;
        static String ldapHost = null;
```

```
        static int ldapPort = -1;
        static String mbox = "INBOX";

        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"))
                                ldapHost = argv[++i];
                        else if (argv[i].equals("-I"))
                                ldapPort = Integer.parseInt(argv[++i]);
                        else if (argv[i].equals("--")) {
                                i++;
                                break;
                        }
                        else if (argv[i].startsWith("-")) {
                                System.out.println(
                                "Usage: ReadMessage [-D ldapHost] [-I ldapPort]
[-U user] [-P password]");
                                System.exit(1);
                        }
                        else {
                                break;
                        }
                }


                Properties props = System.getProperties();

                // Set system properties - it is necessary to set up these
                // properties to obtain the database connect information
                // from oid. The database connect information is available only
                // to privileged users.
                // NOTE: The password may umadmin password may be different
                // for your installation.
                props.setProperty("oracle.mail.ldap.admin_
dn","cn=umadmin,cn=emailservercontainer,cn=products,cn=oraclecontext");
                props.setProperty("oracle.mail.ldap.admin_password","umadmin");

                Session session = Session.getDefaultInstance(props,new
OracleAuthenticator());
```

```
                    session.setDebug(true);

                    Store store = null;

                    store = session.getStore("esmail");

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

                    // Open the folder - after store has connected
                    Folder folder = store.getDefaultFolder();

                    folder = folder.getFolder(mbox);

                    if (folder == null) {
                            System.out.println("Invalid folder");
                            System.exit(1);
                    }
                    else {
                            System.out.println("Folder name : " + folder.getName());
                    }

                    // try to open read/write and if that fails try read-only
                    try {
                       folder.open(Folder.READ_WRITE);
                    } catch (MessagingException ex) {
                       folder.open(Folder.READ_ONLY);
                    }

                    int totalMessages = folder.getMessageCount();

                    if (totalMessages == 0) {
                       System.out.println("Empty folder");
                       folder.close(false);
                       store.close();
                       System.exit(1);
                    }
                    else
                    {
                       System.out.println("Total messages: " + totalMessages);

                       Message[] msgs = folder.getMessages();
                       if (msgs != null)
                            System.out.println("ReadMessage: Number of messages: " +
```

```
msgs.length);
                    int my_uid = 0;
                    for (int i = 0; i < msgs.length; i++)
                    {
                         System.out.println("----------------------------");
                         System.out.println("This is the message uid# : " +
(int)msgs[i].getMessageNumber());
                         my_uid = msgs[i].getMessageNumber();

                         if (my_uid > 0)
                         {
                            System.out.println("Retrieving msg_uid : " + my_uid);
                            Message msg = folder.getMessage(my_uid);
                            System.out.println("After getMessage");
                            //String[] aHdrArray =
(String[])msg.getHeader("X-Mailer");
                            //System.out.println("X-Mailer => " +
aHdrArray.length);
                            //System.out.println("X-Mailer => " + aHdrArray[0]);
                            System.out.println("Sent Date: " +
msg.getSentDate());
                            System.out.println("Msg Size: " + msg.getSize());
                            System.out.println("Received Date: " +
msg.getReceivedDate());
                            System.out.println("Before dumpPart");
                            dumpPart(msg);
                         }
                         else
                         {
                                System.out.println("No messages returned");
                         }
                      }  //end for


                }
                }

            }

            public static void dumpPart(Part p) throws Exception
            {
                if (p instanceof Message)
                   dumpEnvelope((Message)p);

                System.out.println("----------------------------");
```

```
                    pr("CONTENT-TYPE: " + p.getContentType());

                    if (p.isMimeType("text/plain")) {
                            pr("This is plain text");
                            pr("---------------------------");
                            System.out.println((String)p.getContent());
                    } else if (p.isMimeType("multipart/*")) {
                            pr("This is a Multipart");
                            pr("---------------------------");
                            Multipart mp = (Multipart)p.getContent();
                            int count = mp.getCount();
                            for (int i = 0; i < count; i++)
                                  dumpPart(mp.getBodyPart(i));
                    } else if (p.isMimeType("message/rfc822")) {
                            pr("This is a Nested Message");
                            pr("---------------------------");
                            dumpPart((Part)p.getContent());
                    }
                    else if (p.isMimeType("image/jpeg")) {
                        pr("--------> image/jpeg");
                        Object o = p.getContent();
                        if (o instanceof InputStream) {
                           System.out.println("o.length = " +
((InputStream)o).available());
                        }
                        InputStream x = (InputStream)o;
                        // Construct the required byte array
                        System.out.println("x.length = " + x.available());
                        int i = 0;
                        byte[] bArray = new byte[x.available()];

                        while ((i = (int)((InputStream)x).available()) > 0)
                        {
                            int result = (int)(((InputStream)x).read(bArray));
                            if (result == -1) break;
                        }
                        System.out.println("B ARRAY SIZE : " + bArray.length);
                        FileOutputStream f2 = new
FileOutputStream("/tmp/image.jpg");
                        f2.write(bArray);
                    }
                    else {
                            Object o = p.getContent();
                            if (o instanceof String) {
                                    pr("This is a string");
```

```
                pr("--------------------------");
                System.out.println((String)o);
            } else if (o instanceof InputStream) {
                pr("This is just an input stream");
                pr("--------------------------");
                InputStream is = (InputStream)o;
                is = (InputStream)o;
                int c;
                while ((c = is.read()) != -1)
                    System.out.write(c);
            } else {
                pr("This is an unknown type");
                pr("--------------------------");
                pr(o.toString());
            }
        }

    }

    public static void dumpEnvelope(Message m) throws Exception {
        pr("This is the message envelope");
        pr("--------------------------");
        Address[] a;

        // FROM
        if ((a = m.getFrom()) != null) {
            for (int j = 0; j < a.length; j++)
                pr("FROM: " + a[j].toString());
        }

        // TO
        if ((a = m.getRecipients(Message.RecipientType.TO)) != null) {
            for (int j = 0; j < a.length; j++)
                pr("TO: " + a[j].toString());
        }

        // SUBJECT
        if (m.getSubject() != null)
            pr("SUBJECT: " + m.getSubject());

        // DATE
        Date d = m.getSentDate();
        pr("SendDate: " + (d != null ? d.toString() : "UNKNOWN"));
    }
```

```
            public static void pr(String s){
                System.out.println(s);
            }
}
```

Creating a Shared Folder and Granting User Permissions

The following is an example of creating a shared folder and granting permissions to a user. The shared folder name and the user(grantee) are taken in as parameters in addition to the shared folder owner and the owner password.

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;
import oracle.mail.sdk.esmail.*;

public class SharedFolderCreate
{
        static String password = null;
        static String user = null;
        static String aSharedFolderUser = null;
        static String aSharedFolderName = null;
        static String ldapHost = null;
        static int ldapPort = -1;
        static String mbox = "INBOX";

        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("-S"))
                                aSharedFolderUser = argv[++i];
                        else if (argv[i].equals("-N"))
                                aSharedFolderName = argv[++i];
                        else if (argv[i].equals("--")) {
                                i++;
                                break;
                        }
                        else if (argv[i].startsWith("-")) {
                                System.out.println(
```

```
                                "Usage: SharedFolderCreate [-U user] [-P
password] [-S sharedFolderUser] [-N sharedFolderName]");
                                System.exit(1);
                        }
                        else {
                                break;
                        }
                }


                // This property is used to set the default oracle home
                // in the iasv2 environment. The default ldap host and
                // ldap port information is picked up from the
                // $ORACLE_HOME/config/ias.properties config file
                // In the store.connect method, the ldapHost is set to
                // null and the ldapPort is set to -1 to make sure that
                // the default values are used. In this example, the oracle
                // home is retrieved from a system level ORACLE_HOME
                // property.
                Properties props = System.getProperties();
                String orclHome = System.getProperty("ORACLE_HOME");
                props.setProperty("oracle.mail.ldap.oracle_home",orclHome);

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

                Store store = null;

                store = session.getStore("esmail");

                if (user != null && password != null){
                   store.connect(null, -1, user, password);

                   // Open the folder - after store has connected
                   Folder folder = store.getFolder(aSharedFolderName);

                   if (!folder.exists()) {
                        System.out.println("Folder does not exist");
                        folder.create(Folder.HOLDS_MESSAGES);
                   }
                   System.out.println("Folder name : " + folder.getName());
                   System.out.println("Creating Shared Folder");
                   ((OracleFolder)folder).addACI(aSharedFolderUser,
OracleACI.READACI);
                   //((OracleFolder)folder).modifyACI(aSharedFolderUser,
```

```
OracleACI.READACI + OracleACI.WRITEACI);
                  System.out.println("Done Creating Shared Folder");
               }
            }
}
```

Appending Simple Messages

The following is an example of how to append a simple message to a user's inbox. The message is appended to a user's INBOX.

```
import java.io.*;
import java.net.InetAddress;
import java.util.Properties;
import java.util.Date;

import javax.mail.*;
import javax.mail.internet.*;

public class TestAppendMime
{

        static String password = null;
        static String user = null;
        static String host = null;
        static String ldapHost = null;
        static int ldapPort = -1;
        static String mbox = "INBOX";

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

          BufferedReader in =
                      new BufferedReader(new InputStreamReader(System.in));

               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"))
                               ldapHost = argv[++i];
                       else if (argv[i].equals("-I"))
                               ldapPort = Integer.parseInt(argv[++i]);
```

```
                        else if (argv[i].equals("--")) {
                                i++;
                                break;
                        }
                        else if (argv[i].startsWith("-")) {
                                System.out.println(
                                "Usage: TestAppendMime [-D ldapHost] [-I
ldapPort] [-U user] [-P password]");
                                System.exit(1);
                        }
                        else {
                                break;
                        }
                    }
            Properties props = System.getProperties();

            // Set system properties - it is necessary to set up these
            // properties to obtain the database connect information
            // from oid. The database connect information is available only
            // to privileged users.
            // NOTE: The password may umadmin password may be different
            // for your installation.
            props.setProperty("oracle.mail.ldap.admin_
dn","cn=umadmin,cn=emailservercontainer,cn=products,cn=oraclecontext");
            props.setProperty("oracle.mail.ldap.admin_password","umadmin");

            // Get a Session object
            Session session = Session.getDefaultInstance(props, null);
            session.setDebug(true);


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

        System.out.println(ldapHost + "\n" + ldapPort + "\n" + user + "\n" +
                 password);
        store.connect(ldapHost, ldapPort, user, password);

        // Get record Folder.  Create if it does not exist.
        Folder folder = store.getFolder("INBOX");
        Message msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress("oracle@oracle.com"));
        msg.setRecipients(Message.RecipientType.TO,
InternetAddress.parse("testuser1@umdev.us.oracle.com", false));
        msg.setSubject("Welcome!!!");
```

```
        //collect(in, msg);
        msg.setText("Hello welcome\n");
        msg.setSentDate(new Date());

        System.out.println("Total Number of messages : " +
folder.getMessageCount());

        Message[] msgs = new Message[1];
        msgs[0] = msg;
        folder.appendMessages(msgs);

        System.out.println("Total Number of messages : " +
folder.getMessageCount());
        System.out.println("Mail was recorded successfully.");
   }

     public static void collect(BufferedReader in, Message msg)
                         throws MessagingException, IOException {
        String line;
        StringBuffer sb = new StringBuffer();
        while ((line = in.readLine()) != null) {
            sb.append(line);
            sb.append("\n");
        }

        // If the desired charset is known, you can use
        // setText(text, charset)
        msg.setText(sb.toString());
    }
}
```

Basic Folder Operations

The following example demonstrates folder operations - the basic functionality of create folder, copy messages to folder, list folders, rename folders, delete msgs, list folders, delete folders is demonstrated through this script.

```
/
public class TestFolder
{
 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: TestFolder [-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);

              namespace = user.substring(0, user.indexOf('@'));
```

```
  // Open the folder - after store has connected
  Folder folder = store.getDefaultFolder();


  if (folder == null) {
   System.out.println("Invalid folder");
   System.exit(1);
  }
  else {
                        /*** List folders ***************/
                        Folder[] f = folder.list(pattern);
                        for (int i = 0; i < f.length; i++)
                        {
                           System.out.println("Name: " + f[i].getName());
                           System.out.println("Full Name: " +
f[i].getFullName());
                        }
                        /*** Create folder ***************/
   System.out.println("Creating folder xyz");
   Folder aFolder = store.getFolder("/" + namespace + "/xyz");

   if (!aFolder.exists()) //create
                        {
                                System.out.println("Creating folder....");
    aFolder.create(Folder.HOLDS_MESSAGES);
                        }
                        Folder aNewFolder = store.getFolder("/" + namespace +
"/temp");
                        /****** Renaming Folder ***************/
                        System.out.println("Renaming Folder to temp");
                        aFolder.renameTo(aNewFolder);
                        /***** List folder again **************/
                        f = folder.list(pattern);
                        for (int i = 0; i < f.length; i++)
                        {
                           System.out.println("Name: " + f[i].getName());
                           System.out.println("Full Name: " +
f[i].getFullName());
                        }
                        /***** Copy messages into folder ****/
                        Folder mbox = store.getFolder("/" + namespace +
"/INBOX");
                        mbox.open(Folder.READ_WRITE);
                        Message[] msgArray = new Message[3];
                        System.out.println("********" + mbox.getMessageCount());
```

```
                        //Message[] mboxArray = new
Message[mbox.getMessageCount()];
                        for (int j = 0; j < 3; j++)
                          msgArray[j] = mbox.getMessage(j+1);

                        //msgArray = mbox.getMessages(1,3);
                        mbox.copyMessages(msgArray, aFolder);
                        /***** Get message count **********/
                        System.out.println("Number of messages in folder  " +
                            aFolder.getFullName() + " is " +
                            aFolder.getMessageCount());

                        /***** delete messages in folder ****/
                        System.out.println("DELETING two MESSAGES ");
                        System.out.println("---> Number of messages in folder
after delete " + aFolder.getFullName() + " is " +
aFolder.getMessageCount());
                        aFolder.open(Folder.READ_WRITE);
                        Message aMessage = aFolder.getMessage(1);
                        Message aMessage2 = aFolder.getMessage(2);
                        aMessage.setFlag(Flags.Flag.DELETED,true);
                        aMessage2.setFlag(Flags.Flag.DELETED,true);
                        /******* expunge folder *************/
                        System.out.println("EXPUNGING");
                        aFolder.expunge();
                        aFolder.close(true);
                        System.out.println("---> Number of messages in folder
after expunge " + aFolder.getFullName() + " is " +
aFolder.getMessageCount());
                        /******* msg count of folder ********/


                        /***** List folder again **************/
                        System.out.println("************LIST ***********");
                        f = folder.list(pattern);
                        for (int i = 0; i < f.length; i++)
                        {
                           System.out.println("Name: " + f[i].getName());
                           System.out.println("Full Name: " +
f[i].getFullName());
                        }
                        /****** Delete Folder *******/
                        System.out.println("*********** DELETE ***********");
                        aFolder.delete(true);
                        /***** List folder again **************/
```

```
                        f = folder.list(pattern);
                        for (int i = 0; i < f.length; i++)
                        {
                           System.out.println("Name: " + f[i].getName());
                           System.out.println("Full Name: " +
f[i].getFullName());
                        }

  }
      }
         store.close();
            }

     public static void dumpFolder(Folder folder, boolean recurse, String tab)
throws Exception
     {
  System.out.println(tab + "Name:      " + folder.getName());
         System.out.println(tab + "Full Name: " + folder.getFullName());
         System.out.println(tab + "URL:       " + folder.getURLName());

         if (verbose) {
              if (!folder.isSubscribed())
                  System.out.println(tab + "Not Subscribed");

              if ((folder.getType() & Folder.HOLDS_MESSAGES) != 0) {
                  if (folder.hasNewMessages())
                       System.out.println(tab + "Has New Messages");
                   System.out.println(tab + "Total Messages:  " +
                                              folder.getMessageCount());
                   System.out.println(tab + "New Messages:    " +
                                              folder.getNewMessageCount());
                   System.out.println(tab + "Unread Messages: " +
                                              folder.getUnreadMessageCount());
               }
              if ((folder.getType() & Folder.HOLDS_FOLDERS) != 0)
                  System.out.println(tab + "Is Directory");
         }

         System.out.println();

         if ((folder.getType() & Folder.HOLDS_FOLDERS) != 0) {
              if (recurse) {
                  Folder[] f = folder.list();
                  for (int i = 0; i < f.length; i++)
                       dumpFolder(f[i], recurse, tab + "    ");
```

```
            }
        }
    }
}

**************************************************************************
SharedFolderRead.java

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

Shared Folder and Message Fetching

The following is an example demonstrates basic shared folder and fetch message fetching functionality:

```
public class SharedFolderRead
{
        static String user = null;
        static String password = null;
        static String ldapHost = null;
        static int ldapPort = -1;
        static String mbox = "INBOX";
        static String pattern = "*";

        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"))
                                ldapHost = argv[++i];
                        else if (argv[i].equals("-I"))
                                ldapPort = Integer.parseInt(argv[++i]);
                        else if (argv[i].equals("--")) {
                                i++;
                                break;
                        }
                        else if (argv[i].startsWith("-")) {
                                System.out.println(
```

```
                                "Usage: SharedFolderRead [-D ldap_host] [-I
ldap_port] [-U user] [-P password]");
                                System.exit(1);
                        }
                        else {
                                break;
                        }
                }

                Properties props = System.getProperties();

                // Set system properties - it is necessary to set up these
                // properties to obtain the database connect information
                // from oid. The database connect information is available only
                // to privileged users.
                // NOTE: The password may umadmin password may be different
                // for your installation.
                props.setProperty("oracle.mail.ldap.admin_
dn","cn=umadmin,cn=emailservercontainer,cn=products,cn=oraclecontext");
                props.setProperty("oracle.mail.ldap.admin_password","umadmin");

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

                Store store = null;

                store = session.getStore("esmail");

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

                        Folder aLocalFolder = store.getFolder(mbox);
                if (aLocalFolder == null) {
                        System.out.println("NULL FOLDER " + mbox);
                        System.exit(1);
                }
                else {
                        System.out.println("Folder Name: " +
aLocalFolder.getFullName());
                }

                try {
                        aLocalFolder.open(Folder.READ_WRITE);
                } catch (MessagingException ex) {
```

```
                        aLocalFolder.open(Folder.READ_ONLY);
                }

                int totalMessages = aLocalFolder.getMessageCount();

                System.out.println("Total Number of messages in folder " +
                        aLocalFolder.getFullName() + " is " + totalMessages);

                Folder[] aSharedNSArray = store.getSharedNamespaces();

                if (aSharedNSArray == null) {
                        System.out.println("No shared namespaces");
                        System.exit(1);
                }
                else {
                        //For each namespace, list
                        System.out.println("Number of shared namespace : " +
aSharedNSArray.length);

                        Folder folder = null;
                        for (int i = 0; i < aSharedNSArray.length; i++)
                        {
                           folder = aSharedNSArray[i];
                           System.out.println("Folder fullname : " +
folder.getFullName());
                        }
                                Folder[] f = folder.list(pattern);
                                int aNum =0;
                                for (int j = 0; j < f.length; j++)
                                {
                                        System.out.println("Name:      " +
f[j].getName());
                                        System.out.println("Full Name: " +
f[j].getFullName());
                                        aNum = j;

                                try {
                                        f[aNum].open(Folder.READ_WRITE);
                                } catch (MessagingException ex) {
                                        f[aNum].open(Folder.READ_ONLY);
                                }

                         System.out.println("Shared Folder fullname : " +
f[aNum].getFullName());
```

```
                        // Select and fetch messages from Shared Folder
                        int total_messages = f[aNum].getMessageCount();
                        System.out.println("Number of messages in " +
f[aNum].getFullName() + " is " + f[aNum].getMessageCount());

                        if (total_messages == 0) {
                           System.out.println("Empty folder " +
f[aNum].getFullName());
                           f[aNum].close(false);
                        }
                        else {
                              int my_uid = 0;
                              Message[] msgs = f[aNum].getMessages();
                              if (msgs != null)
                                    System.out.println("Number of messages :
" + msgs.length);

                              for (int i = 0; i < msgs.length; i++)
                              {
                                my_uid = msgs[i].getMessageNumber();
                                System.out.println("MSG_NUMBER : " + my_uid);
                                if (my_uid > 0)
                                {
                                 System.out.println("Retrieving msg_num : " +
my_uid);

                                  Message msg = f[aNum].getMessage(my_uid);
                                  dumpPart(msg);
                                 }
                              }

                             // Copy message from shared folder to local
folder
                             //System.out.println("----- BEFORE COPY
-------");
                             //f[aNum].copyMessages(msgs, aLocalFolder);
                             //System.out.println("----- AFTER COPY -------");
                        }
                     }
                  }
               }
               store.close();
    }

    //Fetch message part by part
    public static void dumpPart(Part p) throws Exception
    {
```

```
if (p instanceof Message)
  //pr("Envelope out of order------------>");
  dumpEnvelope((Message)p);

//InputStream is = p.getInputStream();
//System.out.println("SIZE of INPUT STREAM : " + is.available());
System.out.println("----------------------------");
pr("CONTENT-TYPE: " + p.getContentType());

if (p.isMimeType("text/plain")) {
    pr("This is plain text");
    pr("--------------------------");
    System.out.println((String)p.getContent());
} else if (p.isMimeType("multipart/*")) {
    pr("This is a Multipart");
    pr("--------------------------");
    Multipart mp = (Multipart)p.getContent();
    int count = mp.getCount();
    for (int i = 0; i < count; i++)
       dumpPart(mp.getBodyPart(i));
    } else if (p.isMimeType("message/rfc822")) {
       pr("This is a Nested Message");
       pr("--------------------------");
       dumpPart((Part)p.getContent());
    }
    else {
      Object o = p.getContent();
      if (o instanceof String) {
       pr("This is a string");
       pr("--------------------------");
       System.out.println((String)o);
      }
     else if (o instanceof InputStream) {
          pr("This is just an input stream");
          pr("--------------------------");
          InputStream is = (InputStream)o;
          //is = (InputStream)o;
          int c;
          while ((c = is.read()) != -1)
             System.out.write(c);
      } else {
          pr("This is an unknown type");
          pr("--------------------------");
          pr(o.toString());
      }
```

```
            }
        }

    public static void dumpEnvelope(Message m) throws Exception {
        pr("This is the message envelope");
        pr("--------------------------");
        Address[] a;

        // FROM
        if ((a = m.getFrom()) != null) {
            for (int j = 0; j < a.length; j++)
                pr("FROM: " + a[j].toString());
        }

        // TO
        if ((a = m.getRecipients(Message.RecipientType.TO)) != null) {
                for (int j = 0; j < a.length; j++)
                        pr("TO: " + a[j].toString());
        }

        // SUBJECT
                pr("SUBJECT: " + m.getSubject());

        // DATE
        Date d = m.getSentDate();
        pr("SendDate: " +
                        (d != null ? d.toString() : "UNKNOWN"));
    }

    public static void pr(String s){
        System.out.println(s);
    }
    ////--------------
}
```

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 the LDAP directory for a given domain.

This section contains the following topics:

- Directory Components
- Authentication
- Retrieving the MetaData and Validation
- Directory Management Code Examples

> **See Also:** The *Oracle9i*AS Unified Messaging JAVA API Documentation on http://otn.oracle.com, for information about Directory Management APIs

Directory Components

In Oracle9*i*AS Unified Messaging, 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 lists is a mailing list that has an 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 the Thin Client 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, the caller must authenticate with the LDAP directory using the oracle.mail.OESContext class. Once

authenticated, the instance of oracle.mail.OESContext representing a trusted session must be passed to all of the directory APIs.

Authentication Example:

```
ESDSDirectoryAccess dirAccess = oesctx.getDSAccess();
dirAccess.CreateMailUser(oesctx, ……
where, oesctx is an instance of OESContext.
```

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.

Example:

```
ESDSDirectoryAccess dirAccess = oesctx.getDSAccess();
//This call retrieves metadata associated with a mailuser for the domain
oracle.com
ESDSLdapObject ldapobj = access. GetMailUserMetaData(oesctx,"oracle.com ");
//Getting the mandatory attributes for a mailuser 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
ESDSAttribMetaData 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 ESDSLdapObject class.

Example:

This example sets the value for the `telephonenumber` attribute.

```
//This sets the value to the default values provided mdata.getDefaultValues() is
not null
ldapobj.setAttributeValue("telephonenumber", mdata.getDefaultValues());
```

This example sets a new set of values for the `telephonenumber` attribute.

```
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 ESDSLdapObject class. The caller needs to specify the type of modification.

The list of allowed modifications are as follows.

- ESDSConstants.DS_MODIFY_ADD: This adds the given set of values to the existing values .
- ESDSConstants.DS_MODIFY_DELETE: This deletes the given set of values from the existing values.
- ESDSConstants.DS_MODIFY_REPLACE: This replaces the existing values with a new set of values.

Example:

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, ESDSConstants.DS_
MODIFY_ADD);
```

Directory Management Code Examples

In the following examples, it is assumed that `oesctx` is an instance of the OESContext class that the caller obtains during login. The instance of `ESDSDirectoryAccess` is obtained as follows:

```
ESDSDirectoryAccess dirAccess = oesctx.getDSAccess();
```

Note: To run these examples, the CLASSPATH environment variable must include `jndi.jar`, `ldap.jar`, `providerutil.jar`, `classes12.zip`, `repository.jar`, `esldap.jar`, and `escommon.jar`

Mail User Examples

The following examples show how to use the mail user APIs to create, modify, look up, and delete a mail user.

```
/**
Retrieving Mail User Metadata and Creating a New Mail User

**/

ESDSLdapObject ldapobj = dirAccess.GetMailUserMetaData(oesctx,"oracle.com ");

//This example sets the mail id of the new user, all other attributes it sets to
the default values.

if (ldapobj.getMandatoryAttribs() != null) {

Enumeration enum = ldapobj.getMandatoryAttribs().elements();

while (enum.hasMoreElements()) {

String attr = enum.nextElement().toString();

if (attr.equalsIgnoreCase("mail")) {

Vector vals = new Vector();

vals.add("newuser@oracle.com");

ldapobj.setAttributeValue(attr, vals);
```

```
}
else {
//Set everything else same as default values
ESDSAttribMetaData mdata = ldapobj.getMetaData(attr);
if (mdata.getDefaultValues() != null)
ldapobj.setAttributeValue(attr, mdata.getDefaultValues());
}
}
}
//Set the default values for the optional attributes
enum = ldapobj.getOptionalAttribs().elements();
while (enum.hasMoreElements()) {
String attr = enum.nextElement().toString();
//Set everything else same as default values
ESDSAttribMetaData mdata = ldapobj.getMetaData(attr);
if (mdata.getDefaultValues() != null)
ldapobj.setAttributeValue(attr, mdata.getDefaultValues());
}
// Now Create The Mailuser
dirAccess.CreateMailUser(oesctx,"cn=newuser,dc=oracle,dc=com",ldapobj);
// Here ,"cn=newuser,dc=oracle,dc=com" is the dn of the public user
/**
Looking Up a Mail User
```

```
**/

ldapobj = dirAccess.LookupMailUser(oesctx,"newuser@oracle.com");

//Get the telephonenumber of this mailuser

ESDSAttribMetaData mdata = ldapobj.getMetaData("telephonenumber");

Vector values = mdata.getDefaultValues();

/***
Modifying a Mail User

**/

//Modifying the orclmailquota of the user

Vector modify = new Vector();

modify.add ("50000000");

// If you don't want to use the existing ldapobj, you can use as follows

//ldapobj = new ESDSLdapObject(ESDSConstants.DS_USER);

ldapobj.modifyAttributeValue("orclmailquota",

modify,

ESDSConstants.DS_MODIFY_REPLACE);

dirAccess.ModifyMailUser(oesctx, "newuser@oracle.com",ldapobj);

/***

Delete A User

***/

dirAccess.DeleteMailUser(oesctx,"user1@oracle.com");
```

Distribution List Example

The following code describes how to use the distribution list APIs to create, modify, look up, resolve, and delete a distribution list.

```
/**

Getting DL Metadata And Creating A New DL

**/

ESDSLdapObject ldapobj =
dirAccess.GetDistributionListMetaData(oesctx,"oracle.com ");

//This example sets the mail id of the new DL, all other attributes it sets to
the default values.

if (ldapobj.getMandatoryAttribs() != null) {

Enumeration enum = ldapobj.getMandatoryAttribs().elements();

while (enum.hasMoreElements()) {

String attr = enum.nextElement().toString();

if (attr.equalsIgnoreCase("mail")) {

Vector vals = new Vector();

vals.add("newlist@oracle.com");

ldapobj.setAttributeValue(attr, vals);

}

else {

//Set everything else same as default values

ESDSAttribMetaData mdata = ldapobj.getMetaData(attr);

if (mdata.getDefaultValues() != null)

ldapobj.setAttributeValue(attr, mdata.getDefaultValues());

}
```

```
}

}

//Set the default values for the optional attributes

enum = ldapobj.getOptionalAttribs().elements();

while (enum.hasMoreElements()) {

String attr = enum.nextElement().toString();

//Set everything else same as default values

ESDSAttribMetaData mdata = ldapobj.getMetaData(attr);

if (mdata.getDefaultValues() != null)

ldapobj.setAttributeValue(attr, mdata.getDefaultValues());

}

//Add some members to the Dl, if you don't want to add members then the members
hashtable can be null.

Hashtable members = new Hashtable();

/**USERS***/

Vector v = new Vector();

v.add ("htanaya@oracle.com");

v.add ("newuser@oracle.com");

members.put(ESDSConstants.DS_USER,v);

/**OTHER DL's***/

v = new Vector();

v.add("anotherdl@dmv.gov");

members.put(ESDSConstants.DS_LIST,v);
```

```
// Now Create The DL

dirAccess.CreateDistributionList(oesctx, ldapobj, members);

/**
Looking up a DL

**/

ldapobj = dirAccess.LookupDistributionList(oesctx,"newlist@oracle.com");

/***

Modifying a DL

**/

//Modifying the orclmailquota of the user

Vector modify = new Vector();

modify.add ("newhost");

// If you don’t want to use the existing ldapobj, you can use as follows

//ldapobj = new ESDSLdapObject(ESDSConstants.DS_LIST);

ldapobj.modifyAttributeValue("mailhost",

modify,

ESDSConstants.DS_MODIFY_REPLACE);

dirAccess.ModifyDistributionList(oesctx, "dl1@oracle.com",ldapobj);

/***

Delete A DL

***/

dirAccess.DeleteDistributionList(oesctx,"dl1@oracle.com");

/**
```

```
Resolving a DL

**/

//Resolving dl2@oracle.com

Hashtable result = dirAccess.ResolveDistributionList(oesctx, "l2@oracle.com");

//Get the users, other dls and foreign members out of it

Vector users = (Vector) result.get(ESDSConstants.DS_USER);

Vector lists = (Vector) result.get(ESDSConstants.DS_LIST);

Vector foreign = (Vector) result.get(ESDSConstants.DS_FOREIGN);

/***
Modifying DL Members
**/

//Adding two new foreign members to the list

Vector newmembers = new Vector();

newmembers.add("usr@mail.com");

newmembers.add ("user1@netscape.net");

dirAccess.ModifyDistributionListMembers(oesctx, "dl1@oracle.com",
ESDSConstants.DS_MODIFY_ADD,
ESDSConstants.DS_FOREIGN, newmembers);
```

Private Address Book Contact Information Examples

The following code describes how to use the private address book contact information APIs to create, modify, look up, resolve, and search contacts.

```
/**

Getting Contact Info Metadata And Creating A New Contact Info

**/

ESDSLdapObject ldapobj = dirAccess.GetContactInfoMetaData(oesctx);
```

```
//This example sets the name and email id of the new contact. Other attributes
can be set in a similar way.

ldapobj.setAttributeValue("name","myfriend1");

ldapobj.setAttributeValue("orclmailemail","test@hotmail.com");

// Now Create The Contact

dirAccess. CreateContactInfo(oesctx, ldapobj);

/**

Looking up a Contact Info

**/

ldapobj = dirAccess.LookupContactInfo(oesctx,"myfriend1");

/***

Modifying a Contact

**/

//Modifying the given name of the contact

Vector modify = new Vector();

modify.add ("myfriend_name");

ldapobj.modifyAttributeValue("givenname",

modify,

ESDSConstants.DS_MODIFY_ADD);

dirAccess.ModifyContactInfo(oesctx, "myfriend1",ldapobj);

/***

Delete A Contact Info

***/
```

```
dirAccess.DeleteContactInfo(oesctx,"friend ");

/**

Get All Contacts

**/

String[] contacts = dirAccess.GetAllContacts(oesctx);

/**

Get All Contacts For a Given Filter

**/

contacts = dirAccess.SearchContacts(oesctx,"name=t*");
```

Private Distribution List Example

The following code describes how to use the private distribution list APIs to create, modify, look up, resolve, and search contacts.

```
/**

Getting Private List Metadata And Creating A New Private List

**/

ESDSLdapObject ldapobj = dirAccess.GetPrivateListMetaData(oesctx);

//This example sets the name and email id of the new contact. Other attributes
can be set in a similar way.

ldapobj.setAttributeValue("name","myfriends");

ldapobj.setAttributeValue("orclmailemail","test@hotmail.com");

ldapobj.setAttributeValue("orclmailemail","test2@hotmail.com");

// Now Create The Contact

dirAccess.CreatePrivateList(oesctx, ldapobj);
```

```
/**
Looking up Private List
**/
ldapobj = dirAccess.LookupPrivateList(oesctx,"myfriends");
/***
Modifying a private list
**/
//Modifying the given name of the contact
Vector modify = new Vector();
modify.add ("myfriend_name");
ldapobj.modifyAttributeValue("givenname",
modify,
ESDSConstants.DS_MODIFY_ADD);
dirAccess.ModifyContactInfo(oesctx, "myfriends_new",ldapobj);
/***
Delete A Private List
***/
dirAccess.DeletePrivateList(oesctx,"friends ");
/**
Get All Private Lists
**/
String[] lists= dirAccess.GetAllPrivateLists((oesctx);
/**
```

```
Get All Lists For a Given Filter

**/

lists = dirAccess.SearchPrivateLists(oesctx,"name=t*");

/**

Private list resolution

**/

Vector resolved = dirAccess.ResolvePrivateList(oesctx, "friends" ,
"orclmailemail");

// This call resolves the given entry to the attribute "orclmailemail". It
resolved the other private dl's and contacts also to this type.
```

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 LDAP directory as an attribute of a user.

This section contains the following topics:

- What Are Server Side Rules?
- Rule Components
- Authentication
- Rule Visibility, Activeness, and Group Affiliation
- External Condition
- Message Templates
- Auto-Reply Effective Duration
- XML Representation

> **See Also:** The *Oracle9i*AS Unified Messaging JAVA API Documentation on http://otn.oracle.com, for information about Rule Management APIs

What Are 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 on 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.

Authentication

Before a caller can access a user's rules, it must be authorized. The caller must authenticate with the LDAP directory using the oracle.mail.OESContext class. Once authenticated, the instance of the oracle.mail.OESContext class representing a trusted session must be passed into the RuleParser rule management class using the `setAuthContext()` method.

Example:

```
RuleParser parser = new RuleParser();
parser.setAuthContext(oesctx);
```

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 run during runtime.

Validation is disabled as follows:

```
parser.setValidation(false);
```

Rule Visibility, Activeness, and Group Affiliation

A rule has several attributes that are classified as follows:

- Visibility
- Activeness
- Appendix

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.

RuleType Example:

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

External Condition

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

```
function <func_name> (p_folderid in integer,
p_msguid in integer,
p_msgid in integer) return integer;
```

Given the current folder ID, the current message's UID, and the current message ID, 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, the condition function 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()`.

External Condition Example:

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

External Action

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

```
procedure <proc_name>(p_event in number,
p_userid in number,
p_folder in varchar2,
p_msguid in number,
p_msgid in numner,
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 the user ID, current folder name in its full path, the current message UID and ID, 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 in rules, use the Action Type 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.

External Action Example:

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

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 variables %rfc822subject% and %rfc822date% are replaced by the real subject and date sent information obtained from the incoming message. The set of supported substitutable parameters is the same as the set of supported message attributes.

> **See Also:** The Javadoc for Unified Messaging for information on the AttributeType class on ***OTN.

Message template text is used as parameter values for rule actions such as Notify, Reply, Replyall, and Forward.

Message Template Example:

```
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 a concept of 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.

Effective Duration Example:

```
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

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 LDAP directory. To flatten rule Java objects into XML text, use the `print()` method from the Account class.

XML Example:

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

The following code demonstrates using the API to compose 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 {
        // login to LDAP using Directory APIs
        OESContext appCtx = new OESContext();
        OESLoginContext appLogin =
            appCtx.createDSLoginContext(DirectoryConstants.DS_CALLERTYPE_APP);
        appLogin.authenticate(null, "/your/local/oracle/home");
        // authenticate as a rule owner
        OESContext clientCtx = new OESContext();
        OESLoginContext clientLoginCtx =
            clientCtx.createDSLoginContext(ESDSConstants.DS_CALLERTYPE_
MAILUSER);
        clientLoginCtx.authenticate("testuser1@oracle.com", null, appLogin);
        // set authentication context in RuleParser
        parser = new RuleParser();
        parser.setAuthContext(clientCtx);
        // 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 installation name in LDAP,
// such as "install1", 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.setVisible("yes"); // this is the default
rule_t.setActive("yes"); // this is the default
rule_t.setDescription("a new rule");
rule_t.setGroup("group1");
// create a condition type object
ConditionType cond_t = new ConditionType();
cond_t.setJunction("and"); // this is the default
cond_t.setNegation("no"); // this is the default
// 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"); // this is optional
// cond_t.addAttribute(attr_t);
// create a simple operator:
cond_t.addOperator("contains");
cond_t.addOperand("Hello");
// create another condition with an external call
ConditionType cond_t2 = new ConditionType();
cond_t2.addProcCall("extproc");
// create a third condition with sectional match
ConditionType cond_t3 = new ConditionType();
// create an "InSection" object searching for word "Oracle" in
// message body
InSectionType sect_t = new InSectionType("Oracle");
sect_t.setTag("body");
cond_t3.addInSection(sect_t);
// create a negation of disjunction of above three conditions
ConditionType cond_t4 = new ConditionType();
cond_t4.setJunction("or");
cond_t4.setNegation("yes");
cond_t4.addCondition(cond_t);
cond_t4.addCondition(cond_t2);
cond_t4.addCondition(cond_t3);
```

```
            // add the condition object to the rule type object
            rule_t.addCondition(cond_t4);
            // create an action type object
            ActionType action_t = new ActionType();
            action_t.addCommand("pass");
            // 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("discard");
            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.setValidation(true); // default
            parser.setRuleObjects(acnt);
        }
    }
```

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V