PSP_Mail – Dynamic PSP™ Add-On
User's Guide and Reference

Version 1.4.5

Document ID: 001-200101-075
Last Revision: August 24, 2004

COPYRIGHT INFORMATION AND ACKNOWLEDGEMENTS

INTRODUCTION

PSP_Mail is a Dynamic PSP™ add-on package that allows DPSP and PL/SQL developers to send various types of e-mail from DPSP-based applications or any other PL/SQL applications. PSP_Mail can be used in conjunction with Dynamic PSP or standalone. PSP_Mail uses the Oracle8*i*/9*i*/10*g* standard `UTL_SMTP` package as well as several custom Java™ classes to implement its functionality. This document describes the package installation procedure and all subprograms defined in the package along with usage examples.

The reader is assumed to have some knowledge of the SMTP protocol (defined in IETF RFC-2822), related standards (SMTP standard extensions, etc.) and MIME (Multipurpose Internet Mail Extensions) standards (RFC-2045 to 2047). Some most common questions are answered in the product FAQ document accompanying the product (`FAQ.TXT` file.)

PART I. PACKAGE INSTALLATION.

Package distribution archive contains several code files and two installation scripts – one for Windows-based systems (`loadmail{9i}.bat`) and one for Unix and Unix-like systems (`loadmail{9i}.sh`). Both scripts accept one command-line parameter: Oracle `SYS` user password followed by optional @ character and target database TNS alias (configured in the `TNSNAMES.ORA` file or available through Oracle Names server):

```
(Windows) > loadmail{9i} syspassword[@TNSName]
(Unix) $ /bin/sh loadmail{9i}.sh syspassword[@TNSName]
```

If the script is invoked on the host where Oracle RDBMS is installed, only one instance is configured and running on this host and `ORACLE_SID` environment variable is properly set, `@TNSName` is not required, otherwise it should be TNS alias for the Oracle instance you want to connect to (including remote Oracle hosts).
There are also versions of the scripts for Oracle9*i* and later databases, which require `SYS` connections to be `AS SYSDBA`. These scripts are called `loadmail9i.bat` and `loadmail9i.sh`, for Windows and UNIX, respectively. Use these scripts when installing PSP_Mail into 9*i*+ databases. The same invocation rules apply to these scripts.

PACKAGE CONFIGURATION PARAMETERS (CONSTANTS).

Several parameters need to be adjusted in the package header (`PSP_MAIL.PLH` file) according to your network configuration. Provided installation script prompts for values for these parameters during initial installation, but if you need to change them in the future, you will need to edit the package header and recompile it with new values. Parameters are:

`PRIMARY_DNS` - This is the address of your DNS server. You can use either IP address or DNS alias (symbolic name, like ns.yourdomain.com).
Default is `192.9.9.3` (`ns.sun.com`)
All DNS lookups will be performed against this server, so it must be able to do recursive lookups (that is, forward query to other DNS servers if its own database doesn't have mappings for requested domain). Consult with your network administrator to obtain correct address.

`SITE_NAME` - This is the name of your site or Oracle host - this name is sent to the receiving SMTP server to identify the mail originator host. Some SMTP servers are set up to verify the name of host they receive with the DNS alias assigned to the IP address of originator via DNS and deny connection if they do not match - so be sure to set this parameter to correct name.

`DEFAULT_SMTP` - This is the address of default SMTP gateway (relay server) that will be used for mail delivery in case DNS lookup fails to locate mail server accepting mail for particular recipient. This server must be able to relay mail originating from senders you will specify or mail coming from the IP address of your Oracle server (consult with your network or mail administrator to obtain this address and ensure it is configured to allow mail relaying from your Oracle host).
Set this parameter to `NULL` to disable sending mail through the mail relay

server if it can't be delivered directly. SendMail functions will return 510 error code if DNS lookup fails and this parameter is `NULL`.

SMTP_PORT - This is `DEFAULT_SMTP` server TCP port. Default value is `25`, standard SMTP port. This value should not be changed unless instructed to do so by network administrator (for example, if the relaying SMTP server was set up to listen on different port to prevent spamming through it).

ALWAYS_RELAY - This Boolean parameter defines whether PSP_Mail should not perform DNS lookups and send all outgoing mail through mail relay host defined with `DEFAULT_SMTP`. Default value for this parameter is `False`. Setting it to `True` will cause all mail to be sent through `DEFAULT_SMTP` relay server. If `DEFAULT_SMTP` is `NULL` (not defined), setting this parameter to `True` will cause package initialization error.

GMT_DIFF - This numeric parameter defines offset in hours from GMT (Greenwich Mean Time), currently known as UTC (Universal Coordinated Time), and is used for correct `Date` header generation. Default value for this parameter is `0`, meaning no offset.

On Oracle9*i* and later, this parameter has no effect as the `Date` header is generated according to the session time zone information.

PART II. PACKAGE DESCRIPTION AND REFERENCE.

PACKAGE DESCRIPTION.

The **PSP_Mail** package implements a set of subprograms allowing Oracle PL/SQL developers to send e-mail from DPSP objects or any PL/SQL stored procedures without any external scripts/programs. Package subprograms automatically retrieve DNS information to determine SMTP server, which accepts e-mail for particular recipient and then uses UTL_SMTP standard package to connect to the server and send e-mail. Subprograms defined in the package include versions allowing sending mail to several recipients with one subprogram call, attaching LOBs to the message (LOBs are automatically encoded using Base64 encoding standard and MIME-standard multipart/mixed message is automatically built.) Package subprograms automatically fix common problems in messages that can prevent them from being delivered due to receiving SMTP server's strict SMTP implementation (like lone or out-of-order CR or LF characters, non-canonic email addresses, etc.). The package extensively uses PL/SQL overloading feature – there are several versions of SendMailEx and SendMailEx2 functions, each accepting different set of parameters, which simplifies package use.
Currently, there is no way to create multipart/alternate MIME messages (this format is used to combine several versions of the same message, for example plain text and HTML, into single message so that mail reader can select the most complex version it is able to render) automatically, but you can use the low-level SendMail interface to send any correctly formed MIME messages – it is up to the developer how to generate such messages in this case.

REQUIREMENTS.

The PSP_Mail package requires the following to operate:

- Oracle8*i* Release 2 (8.1.6) or later RDBMS on any Oracle-supported platform;
- Oracle JServer/Oracle VM installed (built-in Java Virtual Machine);
- Unrestricted access to the recursive DNS server for DNS lookups from the Oracle host where PSP_Mail will be installed (consult with your network administrator for DNS server address,) if local mail relay server is not [always] used;
- PSP_Mail packages must be installed into SYS schema.

PACKAGE-DEFINED TYPES, VARIABLES AND CONSTANTS.

The **PSP_Mail** package defines some types for its input and output values:

```
TYPE Strings IS TABLE OF VARCHAR2(32000) INDEX BY BINARY_INTEGER;
```

This type defines an indexed collection (array) of character strings.

```
TYPE TBAttachment IS RECORD(
                            content      BLOB,
                            name         VARCHAR2(100),
                            content_type VARCHAR2(100),
                            content_id   VARCHAR2(100) );
```

This type defines a record describing an e-mail attachment. **CONTENT** is a binary LOB (BLOB) descriptor, **NAME** is the name of the attachment (usually its file name), **CONTENT_TYPE** is a string defining attachment's MIME content type and **CONTENT_ID** is an optional content ID string that can be used to identify the attachment and refer to it from the message body (useful, for example, when sending HTML message with images and attaching image files to the HTML – in this case you could reference them from HTML message body with `cid:` prefix in SRC attribute <img src="**cid:**image_content_id">). This format is supported by Microsoft Outlook [Express] and, maybe, by other email clients.

```
TYPE TBAttachments IS TABLE OF TBAttachment INDEX BY BINARY_INTEGER;
```

This type defines an indexed collection (array) of **TBAttachment** attachment descriptor records.

```
TYPE TCAttachment IS Record(
                            content      CLOB,
                            name         VARCHAR2(100),
                            content_type VARCHAR2(100),
                            content_id   VARCHAR2(100));

TYPE TCAttachments IS TABLE OF TCAttachment INDEX BY BINARY_INTEGER;
```

These two types are character LOB (CLOB) versions of attachment descriptor record and descriptors collection.

```
TBAtt_Empty TBAttachments;
TCAtt_Empty TCAttachments;
```

These two variables represent empty collections of TxAttachment. You can use them when you do not want to attach any LOBs to the message.

```
CrLf CONSTANT VarChar2(2) := CHR(13)||CHR(10);
```

This constant is an abbreviation to CR/LF (ASCII codes 13 and 10, respectively) sequence of characters, used in MIME standard for line termination. You should use this constant or its equivalent for terminating lines in your messages, or your messages may be rejected by strict SMTP hosts (QMail is one known SMTP daemon to reject such messages).

```
NLT CONSTANT VarChar2(3) := CrLf||CHR(9);
```

This constant is an abbreviation to CR/LF/Tab sequence of characters, used in MIME for splitting parameters in headers. It is also used as continuation mark (folding white space, FWS) for very long headers (according to the standard, individual header lines should not exceed 80 characters, thus overly long headers should be "folded" so that each line does not exceed 80 characters.)

```
RELAY_IN_SINGLETRANS Boolean := TRUE;
```

This variable affects the way messages targeting multiple recipients are sent, when ALWAYS_RELAY parameter is set to TRUE. When RELAY_IN_SINGLETRANS is TRUE (the default), the message is sent in single SMTP transaction (that is, all recipients are specified to the relay server and single copy of the

message is sent) if possible, or in as few transactions as possible if the relay server imposes some limit on the number of recipients in single mail transaction, otherwise a separate SMTP transaction for each recipient takes place (that is, the same message is sent once for each recipient in separate mail transaction.) Sending the message in single SMTP transaction saves bandwidth and is obviously much faster than sending the same message in separate transactions for each recipient, especially when the message contains attachments and targets a lot of recipients.
This variable does not affect package behavior when `ALWAYS_RELAY` is `FALSE`, because the package has to send the message in separate transaction for each recipient (currently, the package doesn't attempt to group the recipients by their resolved MX.)
Note: this variable and the feature it controls are new in Version 1.4.0 and later. If you want the package to behave as previous versions, change the default value of this variable to `FALSE` in the package header and recompile the package, or set it to `FALSE` explicitly before invoking `SendMailEx` functions.

```
AUTH_USERNAME VARCHAR2(100);
AUTH_PASSWORD VARCHAR2(100);
```

These variables are used for SMTP Authentication (RFC-2554) if *both* are not `NULL`. PSP_Mail automatically checks for supported authentication methods and attempts to authenticate with the server using the username and password supplied in these variables when new mail transaction is initiated by a `SendMail`*XXX* function. Supported authentication methods are `CRAM-MD5` (RFC-2195) and `LOGIN` (`CRAM-MD5` is used if supported, otherwise less secure `LOGIN` is used.) These variables are not automatically reset when mail transaction is complete, it is up to the caller to initialize and reset them as needed.
Important note: improper/invalid credentials supplied in these variables will cause PSP_Mail mail transaction to fail with `520` error code (internal error,) the package will not proceed unauthenticated if these variables are set.
You will rarely need to assign these variables – possible case may be the relay SMTP server that only relays mail for authenticated senders. Be sure to reset them to `NULL` before sending mail directly to the recipient's MX server, otherwise PSP_Mail will attempt to authenticate with that server using supplied credentials and will most probably fail.

PACKAGE-DEFINED FUNCTIONS.

The PSP_Mail package defines several functions that developers can invoke for sending e-mail and various other tasks.

MXLOOKUP FUNCTION.

```
FUNCTION MXLookup(host VARCHAR2,
                  recNo PLS_INTEGER DEFAULT -1) RETURN VARCHAR2;
```

This function performs a DNS query to the PRIMARY_DNS DNS server to find MX (Mail eXchanger) server for **host** host or domain. If no MX records are found, A (Address) lookup for **host** is performed to verify if this is a standalone host rather than domain name (it is assumed that the host receives and processes its own mail locally in this case.) If either DNS query succeeds, returned value contains the DNS name of the SMTP server that accepts mail for given host or domain, otherwise DEFAULT_SMTP is returned, indicating that DEFAULT_SMTP mail relay should be used. If you set DEFAULT_SMTP to NULL, then NULL will be returned for failed DNS lookups. **recNo** is an optional parameter indicating which MX record to return from DNS reply (if there is only one MX record in response, it will always be returned regardless the **recNo** value). Default of -1 for **recNo** parameter requests to return preferred MX record (the record with the least *preference* attribute), 0 or positive value specifies zero-based index into the list of MX records returned by the DNS server.
This function is used internally in other package functions for determining SMTP server(s) to use for mail delivery.
This function can be used in SQL (SELECT) statements.

Exceptions: None.

Example call:

```
SELECT PSP_Mail.MXLookup('oracle.com') "e-mail server"
  FROM sys.dual;
```


GETMXFOREMAIL FUNCTION.

```
FUNCTION getMXForEmail( recipient VARCHAR2,
                        recNo PLS_INTEGER DEFAULT -1 ) RETURN VARCHAR2;
```

This function performs a DNS query to the PRIMARY_DNS DNS server to find MX (Mail eXchanger) server for **recipient**. **recipient** parameter is recipient's e-mail address. Domain name is automatically extracted from this address and passed to MXLookup function. If ALWAYS_RELAY is set to TRUE, DEFAULT_SMTP is always returned and no DNS query is performed, otherwise MXLookup is invoked to look up the MX for recipient's domain. If DNS query succeeds, returned value contains name of the SMTP server that accepts mail for given recipient, otherwise DEFAULT_SMTP is returned, indicating that DEFAULT_SMTP mail relay should be used. If you set DEFAULT_SMTP to NULL, then NULL will be returned for failed DNS lookups. **recNo** is an optional parameter indicating which MX record to return from the DNS reply (if there is only one MX record in the response, it will always be returned regardless the **recNo** value). It generally should not be specified unless you want to implement some level of load-balancing so that mail to the same domain will be sent through different MX host each time.
Default of –1 for **recNo** parameter requests to return preferred MX record (the record with the least *preference* attribute), 0 or positive value specifies zero-based index into the list of MX records returned by the DNS server.
This function is used internally in other package functions for determining SMTP server(s) to use for mail delivery.
This function can be used in SQL (SELECT) statements.

Exceptions: None.

Example call:

```
SELECT PSP_Mail.getMXForEmail('Larry.Ellison@oracle.com') "e-mail server"
  FROM sys.dual;
```


SENDMAIL FUNCTION.

```
FUNCTION SendMail (sender    VARCHAR2,
                   recipient VARCHAR2,
                   message   Strings
                  ) RETURN VARCHAR2;
```

SendMail is a very generic version of e-mail sender function. It accepts two email addresses – **sender** and **recipient**, and a fully formatted MIME-compliant message text in **message**. No format checking or processing is performed, message is sent blindly to the recipient. If any error occurs during send, returned value will contain the error description (it will either be SMTP error reply from the server, or PL/SQL exception description if any exception was thrown by UTL_SMTP or underlying packages/Java classes). If no error occurred, return value will be NULL.

SMTP authentication takes place automatically on handshake with the SMTP server if both AUTH_USERNAME and AUTH_PASSWORD global variables are not NULL.

Exceptions: None.

Example call:

```
DECLARE
        res VarChar2(32000);
        msg PSP_Mail.Strings;
BEGIN
 msg(1) := 'From: "Some Name" <someone@yourcompany.com>
To: "Some Other Name" <someone@theircompany.com>
Content-Type: text/plain
Subject: hello

Hello!
This is a test message.
';
 res := PSP_Mail.SendMail('someone@yourcompany.com',
                          'someone@theircompany.com',
                          msg);
 IF res IS NOT NULL THEN
   dbms_output.put_line('Error during SendMail: ', res);
 END IF;
END;
```

Note how we put line breaks in the message here. PL/SQL compiler correctly leaves line breaks intact between starting and ending single quotes of constant character string, so we do not need to concatenate each line with CHR(13)||CHR(10) sequence. PSP_Mail will automatically fix up any 'wrong' line breaks.
You will rarely need to use this function as **SendMailEx** and **SendMailEx2** provide much easier interfaces for building message body and add all necessary headers automatically.

SENDMAILEX FUNCTION.

SendMailEx is the main function of the package. It has several overloaded variants with different sets of parameters, allowing you to send almost any type of mail. Below are interfaces of all versions, along with brief descriptions of parameters for each version and examples of invocation.

All versions return NULL if operation was successful, or error message text, just like SendMail function. Multi-recipient versions return Strings collection with either NULL or error message for each recipient. No exceptions should be raised during execution, so it is generally not needed to wrap calls to these functions into BEGIN…EXCEPTION…END; blocks.

SendMailEx assumes the content transfer encoding to be 8bit unless Content-Transfer-Encoding header with different encoding is issued in other_headers. If no Content-Transfer-Encoding is issued in other_headers, default Content-Transfer-Encoding: 8bit header is added automatically. **SendMailEx** then attempts to negotiate 8-bit MIME content transfer through the EHLO command. However, if the remote server does not support EHLO command, or does not support 8BITMIME extension, the message is still delivered in 8bit encoding. With older servers this may cause message headers or body to be corrupted if they contain non-ASCII characters, because the standard explicitly requires that all messages should use 7-bit encoding and do not contain any non-ASCII characters, and that the server may alter the message to comply with this requirement by zeroing the highest bit of each message byte. For 100% guaranteed delivery, it is recommended that you encode non-ASCII message bodies and headers using Base64 encoding and specify Content-Transfer-Encoding: base64 in the other_headers argument.

Common parameters (present in all versions):

sender	E-mail address of the sender. VARCHAR2 string. **Mandatory** parameter. All functions will return 505 error if this parameter is NULL.
sender_name	Name of the sender. VARCHAR2 string. May be blank.
subject	Subject of the email. VARCHAR2 strings. May be blank.
message_body	Body of the message (essentially, message text). PSP_Mail.Strings collection, allowing sending messages of virtually any length.
other_headers	Any additional MIME headers you want to send with the message. VARCHAR2 string. Headers should be separated by CR/LF character sequences, no blank lines allowed. Consult IETF RFC-2822 (SMTP protocol) and MIME standards (RFCs 2045 through 2047) for full reference on Internet message headers/MIME headers at http://www.ietf.org/rfc. The most common use for this argument is to specify message content transfer encoding (Content-Transfer-Encoding header) if it differs from default 8bit encoding (Base64, for example). It may also be used to specify alternate reply address (Reply-To header) or delivery failure notification return path (Return-Path header), to specify message priority and importance (X-Priority, X-MSMail-Priority and Importance headers) and to add any other standard or experimental/proprietary message headers to the message.
content_type	MIME content type of the message body. VARCHAR2 string. Default is 'text/plain; charset="iso-8859-1" '.

SMTP authentication takes place automatically on handshake with the SMTP server if both AUTH_USERNAME and AUTH_PASSWORD global variables are not NULL.

Exceptions: None.

Following are the prototypes and examples for each version of **SendMailEx** function.

- **Single recipient, no attachments.**

```
FUNCTION SendMailEx( sender         VARCHAR2,
                     sender_name    VARCHAR2,
                     recipient      VARCHAR2,
                     recipient_name VARCHAR2,
                     subject        VARCHAR2,
                     message_body   Strings,
                     other_headers  VARCHAR2 DEFAULT NULL,
                     content_type   VARCHAR2 DEFAULT 'text/plain;'
                                    ||nlt||'charset="iso-8859-1"'
                   ) RETURN VARCHAR2;
```

recipient is the email address of the recipient, **recipient_name** is name of the recipient or any other string.

Example call:

```
DECLARE
   res  VARCHAR2(32000);
   msg  PSP_Mail.Strings;
BEGIN
  msg(1) := 'Hello there,

This is a test message, please do not reply.

Best regards,
   Me.';
  -- authenticate as user jdoe using password 'xxxxxx'
  PSP_Mail.AUTH_USERNAME := 'jdoe';
  PSP_Mail.AUTH_PASSWORD := 'xxxxxx';

  res := PSP_Mail.SendMailEx(
            'me@yourcompany.com', 'Your Name',
            'you@theircompany.com', 'His Name',
            'Some subject',
            msg,
            -- add a header requesting confirmation
            -- of message receipt and specify where
            -- to return undeliverable message.
            'Disposition-Notification-To: <me@yourcompany.com>'||PSP_Mail.CrLf||
            'Return-Path: <errors@mycompany.com>'
           );
 IF res IS NOT NULL THEN
   dbms_output.put_line('Error during SendMailEx: '||res);
 END IF;
END;
```


- **Multiple recipients, no attachments.**

```
FUNCTION SendMailEx(
                    sender          VARCHAR2,
                    sender_name     VARCHAR2,
                    recipients      Strings,
                    recipient_names Strings,
                    subject         VARCHAR2,
                    message_body    Strings,
                    other_headers   VARCHAR2 DEFAULT NULL,
                    content_type    VARCHAR2 DEFAULT 'text/plain;'||
                                    nlt||'charset="iso-8859-1"'
                    ) RETURN Strings;
```

recipients and **recipient_names** are Strings collections. Each `recipients` entry should have either a string or a `NULL` in corresponding `recipient_names` entry. "520 Internal Error" will be returned for particular `recipients` element if corresponding element in `recipient_names` is undefined for it.

`RELAY_IN_SINGLETRANS` flag affects how the message is sent in `ALWAYS_RELAY` mode: if this flag is `TRUE`, single copy of the message is sent to the relay server for all recipients in one SMTP transaction, otherwise separate copy of the message is sent for each recipient.

Example call:

```
DECLARE
   res PSP_Mail.Strings;
   rc  PSP_Mail.Strings;
   rn  PSP_Mail.Strings;
   msg PSP_Mail.Strings;
BEGIN
  rc(1) := 'he@theircompany.com';
  rn(1) := 'His Name';
  rc(2) := 'she@theircompany.com';
  rn(2) := 'Her Name';
  msg(1) := 'Hello there,

This is a test message, please do not reply.

Best regards,
   Me.';
  res := PSP_Mail.SendMailEx('me@yourcompany.com', 'Your Name',
                             rc, rn,
                             'Some subject',
                              msg);
 IF res.count > 0 THEN
  FOR i IN res.first..res.last LOOP
   IF res(i) IS NOT NULL THEN
    dbms_output.put_line('Error delivering to '|| rc(i) || ': '|| res(i));
   END IF;
  END LOOP;
 END IF;
END;
```


- **Single recipient, single attachment.**

```
FUNCTION SendMailEx(sender         VARCHAR2,
                    sender_name    VARCHAR2,
                    recipient      VARCHAR2,
                    recipient_name VARCHAR2,
                    subject        VARCHAR2,
                    message_body   Strings,
                    attachment     {C|B}LOB,
                    att_file_name  VARCHAR2,
                  att_content_type VARCHAR2 DEFAULT 'application/octet-stream',
                    att_content_id VARCHAR2 DEFAULT NULL,
                    other_headers  VARCHAR2 DEFAULT NULL,
                    content_type   VARCHAR2 DEFAULT 'text/plain;'
                                   ||nlt||'charset="iso-8859-1"'
                   ) RETURN varchar2;
```

attachment is either a CLOB or a BLOB to be attached to the message, **att_file_name** is the file name to be associated with the attachment, **att_content_type** is MIME content type of the attachment, default is 'application/octet-stream', that is, untyped binary file. **att_content_id** is an optional content ID string.

Example call:

```
DECLARE
   res  VARCHAR2(32000);
   att  BLOB;
   msg  PSP_Mail.Strings;
BEGIN
  msg(1) := 'Hello there,

Please find enclosed a document you requested.

Best regards,
   Me.';
  -- get LOB content into variable (table and column are fictious)
  SELECT LOB_COLUMN INTO att FROM LOB_TABLE;
  -- send it, it will automatically be base64-encoded
  res := PSP_Mail.SendMailEx('me@yourcompany.com', 'Your Name',
                             'you@theircompany.com', 'His name',
                             'Some subject',
                              msg,
                              att, 'document.doc',
                              'application/vnd.msword');
  IF res IS NOT NULL THEN
   dbms_output.put_line('Error while SendMailEx: '|| res);
  END IF;
END;
```


- **Multiple recipients, single attachment.**

```
FUNCTION SendMailEx( sender           VARCHAR2,
                     sender_name      VARCHAR2,
                     recipients       Strings,
                     recipient_names  Strings,
                     subject          VARCHAR2,
                     message_body     Strings,
                     attachment       {C|B}LOB,
                     att_file_name    VARCHAR2,
                   att_content_type VARCHAR2 DEFAULT 'application/octet-stream',
                     att_content_id VARCHAR2 DEFAULT NULL,
                     other_headers    VARCHAR2 DEFAULT NULL,
                     content_type     VARCHAR2 DEFAULT 'text/plain;'
                                      ||nlt||'charset="iso-8859-1"'
                    ) RETURN Strings;
```

attachment is either a CLOB or a BLOB to be attached to the message, **att_file_name** is the file name to be associated with the attachment, **att_content_type** is MIME content type of the attachment, default is 'application/octet-stream', that is, untyped binary file. **att_content_id** is an optional content ID string. **recipients** and **recipient_names** are Strings collections. Each recipients entry should have corresponding initialized recipient_names entry (VARCHAR2 string or NULL). "520 Internal error" will be returned for each recipients element that does not have corresponding element in recipient_names.

RELAY_IN_SINGLETRANS flag affects how the message is sent in ALWAYS_RELAY mode: if this flag is TRUE, single copy of the message is sent to the relay server for all recipients in one SMTP transaction, otherwise separate copy of the message is sent for each recipient.

Example call:

```
DECLARE
   res  PSP_Mail.Strings;
   att  BLOB;
   rc   PSP_Mail.Strings;
   rn   PSP_Mail.Strings;
   msg  PSP_Mail.Strings;
BEGIN
  msg(1) := 'Hello there,

Please find enclosed a document you requested.

Best regards,
   Me.';
  -- get LOB content into variable (table and column are fictious)
  SELECT LOB_COLUMN INTO att FROM LOB_TABLE;
  -- send it, it will automatically be base64-encoded
  rc(1) := 'he@theircompany.com';
  rn(1) := 'His Name';
  rc(2) := 'she@theircompany.com';
  rn(2) := 'Her Name';
  res := PSP_Mail.SendMailEx('me@yourcompany.com', 'Your Name',
                             rc, rn,
                             'Some subject',
                             msg,
                             att, 'document.doc',
                             'application/vnd.msword');
 IF res.count > 0 THEN
  FOR i IN res.first..res.last LOOP
   IF res(i) IS NOT NULL THEN
    dbms_output.put_line('Error delivering to '|| rc(i) || ': '|| res(i));
   END IF;
  END LOOP;
 END IF;
END;
```


- **Single recipient, multiple attachments.**

```
FUNCTION SendMailEx(sender         VARCHAR2,
                    sender_name    VARCHAR2,
                    recipient      VARCHAR2,
                    recipient_name VARCHAR2,
                    subject        VARCHAR2,
                    message_body   Strings,
                    attachments    T{B|C}Attachments,
                    other_headers  VARCHAR2 DEFAULT NULL,
                    content_type   VARCHAR2 DEFAULT 'text/plain;'
                                   ||nlt||'charset="iso-8859-1"'
                   ) RETURN VARCHAR2;
```

attachments is either TBAttachments (BLOB version) or TCAttachments (CLOB version) collection. All other parameters are the same as with other versions of SendMailEx.

Example call:

```
DECLARE
   res   VARCHAR2(32000);
   atts  PSP_Mail.TBAttachments;
   msg   PSP_Mail.Strings;
BEGIN
  msg(1) := 'Hello there,

Please find enclosed documents you requested.

Best regards,
   Me.';
  SELECT LOB_COLUMN INTO atts(1).content
    FROM LOB_TABLE WHERE NAME='aaaaa';
  atts(1).name         := 'aaaaa.doc';
  atts(1).content_type := 'application/vnd.msword';
  SELECT LOB_COLUMN INTO atts(2).content
    FROM LOB_TABLE WHERE NAME='bbbbb';
  atts(2).name         := 'bbbbb.doc';
  atts(2).content_type := 'application/vnd.msword';
  res := PSP_Mail.SendMailEx('me@yourcompany.com', 'Your Name',
                             'you@theircompany.com', 'His name',
                             'Some subject',
                             msg,
                             atts);
  IF res IS NOT NULL THEN
   dbms_output.put_line('Error while SendMailEx: '|| res);
  END IF;
END;
```


- **Multiple recipients, multiple attachments.**

```
FUNCTION SendMailEx(sender          VARCHAR2,
                    sender_name     VARCHAR2,
                    recipients      Strings,
                    recipient_names Strings,
                    subject         VARCHAR2,
                    message_body    Strings,
                    attachments     T{C|B}Attachments,
                    other_headers   VARCHAR2 DEFAULT NULL,
                    content_type    VARCHAR2 DEFAULT 'text/plain;'
                                    ||nlt||'charset="iso-8859-1"'
                   ) RETURN Strings;
```

Now, finally, the most complex version of SendMailEx: **recipients** and **recipient_names** are `Strings` collections of recipients addresses and name respectively, **attachments** is either `TBAttachments` (`BLOB` version) or `TCAttachments` (`CLOB` version) collection.

`RELAY_IN_SINGLETRANS` flag affects how the message is sent in `ALWAYS_RELAY` mode: if this flag is `TRUE`, single copy of the message is sent to the relay server for all recipients in one SMTP transaction, otherwise separate copy of the message is sent for each recipient.

Example call:

```
DECLARE
   res   PSP_Mail.Strings;
   rc    PSP_Mail.Strings;
   rn    PSP_Mail.Strings;
   atts  PSP_Mail.TBAttachments;
   msg   PSP_Mail.Strings;
BEGIN
  rc(1) := 'he@theircompany.com';
  rn(1) := 'His Name';
  rc(2) := 'she@theircompany.com';
  rn(2) := 'Her Name';
  SELECT LOB_COLUMN INTO atts(1).content
    FROM LOB_TABLE WHERE NAME='aaaaa.doc';
  atts(1).name         := 'aaaaa.doc';
  atts(1).content_type := 'application/vnd.msword';
  SELECT LOB_COLUMN INTO atts(2).content
    FROM LOB_TABLE WHERE NAME='bbbbb.xls';
  atts(2).name         := 'bbbbb.xls';
  atts(2).content_type := 'application/vnd.msexcel';
  msg(1) := 'Hello there,

Please find enclosed documents you requested.

Best regards,
   Me.';
  res := PSP_Mail.SendMailEx('me@yourcompany.com', 'Your Name',
                             rc, rn,
                             'Some subject',
                             msg, atts);

 IF res.count > 0 THEN
  FOR i IN res.first..res.last LOOP
   IF res(i) IS NOT NULL THEN
    dbms_output.put_line('Error delivering to '|| rc(i) || ': '|| res(i));
   END IF;
  END LOOP;
 END IF;
END;
```


SENDMAILEX2 FUNCTION.

SendMailEx2 was introduced in PSP_Mail Version 1.1 along with some supplemental functions and procedures for RFC-2822 email address format handling. The function allows you to define recipient addresses in a more convenient manner, as people usually do in their e-mail clients. This function accepts three lists of RFC-2822 formatted addresses, for `To:`, `Cc:` and `Bcc:` headers respectively, internally processes them and sends the mail to all recipients specified in the lists according to general rules (for example, Bcc: targets are not appearing on recipient lists, but instead blindly sent a copy without any mention of their address in the email). **SendMailEx2** is an all-in-one function that also allows you to attach LOBs to the email, specify additional headers for the message, etc.

```
FUNCTION SendMailEx2( v_from           VARCHAR2,
                      v_to             VARCHAR2,
                      v_cc             VARCHAR2 DEFAULT NULL,
                      v_bcc            VARCHAR2 DEFAULT NULL,
                      v_subject        VARCHAR2 DEFAULT 'No Subject',
                      v_body           Strings,
                      attachments      T{C|B}Attachments,
                      other_headers    VARCHAR2 DEFAULT NULL,
                      content_type     VARCHAR2 DEFAULT 'text/plain;'
                                       ||nlt||'charset="iso-8859-1"',
                      list_delimiter   VARCHAR2 DEFAULT ','
                    ) RETURN Strings;
```

All addresses in **v_from**, **v_to**, **v_cc** and **v_bcc** parameters should follow RFC-2822 formatting:
["literal string"] **<user@host>** [, …]
where elements in square brackets are optional, and **bold** elements are required. Example of a valid address list:

```
"John Doe" <jdoe@host1.com>, <rroe@host1.com>, "Jane" <janedoe@mailhost.com>
```

attachments may be either `TBAttachments` or `TCAttachments`. If you do not want to attach anything to the message, use empty array (**TxAtt_Empty**) as the value of this parameter. You cannot use `NULL` value for this parameter, as this will confuse Oracle giving more than one function matching the list of actual parameters.
v_to is not required to contain a valid address list, it can be any string. This may be useful for blind-mailing to multiple recipients using **v_bcc** and putting something like 'Undisclosed Recipients' in the `To:` header. However, if **v_to** contains at least one @ character, it is assumed to be an address list and will be processed accordingly.
list_delimiter is an optional parameter allowing you to specify the list delimiter character used in **v_to**, **v_cc** and **v_bcc** parameters, it defaults to comma as per RFC-2822. Some clients also allow this delimiter character to be semicolon although this is not allowed in the RFC.
The function returns delivery status line for each recipient in target lists. If some of the addresses were invalid, message will be not delivered to them, but it will still be delivered to all valid addresses. Resulting `Strings` array will hold error message in form 'user@host: error description' for unsuccessful deliveries, and `NULL` for successful ones. Number of entries in returned array will match total number of addresses specified in all three address lists, and order of status lines in result array will match order of addresses in **v_to || v_cc || v_bcc**.

SMTP authentication takes place automatically on handshake with the SMTP server if both `AUTH_USERNAME` and `AUTH_PASSWORD` global variables are not `NULL`.

Exceptions: None.

`RELAY_IN_SINGLETRANS` flag affects how the message is sent in `ALWAYS_RELAY` mode: if this flag is `TRUE`, single copy of the message is sent to the relay server for all recipients in one SMTP transaction, otherwise separate copy of the message is sent for each recipient.

SendMailEx2 assumes the content transfer encoding to be `8bit` unless `Content-Transfer-Encoding` header with different encoding is issued in `other_headers`. If no `Content-Transfer-Encoding` is issued in `other_headers`, default `Content-Transfer-Encoding: 8bit` header is added automatically. **SendMailEx2** then attempts to negotiate 8-bit MIME content transfer through the `EHLO` command. However, if the remote server does not support `EHLO` command, or does not support `8BITMIME` extension, the message is still delivered in 8bit encoding. With older servers this may cause message headers or body to be corrupted if they contain non-ASCII characters, because the standard explicitly requires that all messages should use 7-bit encoding and do not contain any non-ASCII characters, and that the server may alter the message to comply with this requirement by zeroing the highest bit of each message byte. For 100% guaranteed delivery, it is recommended that you encode non-ASCII message bodies and headers using Base64 encoding and specify `Content-Transfer-Encoding: base64` in the `other_headers` argument.

Example call (no attachments, uses `PSP_Mail.TBAtt_Empty`):

```
DECLARE
        msg PSP_Mail.Strings;
        res PSP_Mail.Strings;
BEGIN
 msg(1) := 'Test Message';
 res := PSP_Mail.SendMailEx2(v_from => '"Me" <mymail@mycompany.com>',
                             v_to => '"You" <yourmail@yourcompany.com>',
                             v_subject => 'Test message',
                             v_body => msg,
                             attachments => PSP_Mail.TBAtt_Empty);
 FOR i IN res.first..res.last LOOP
  IF res(i) IS NOT NULL THEN
   -- print out error message along with failed email address
    dbms_output.put_line(res(i));
  END IF;
 END LOOP;
END;
```

Example with attachments (this example assumes you have a table `T` with the following layout:

```
TABLE T (
           NAME          VARCHAR2(100),
           CONTENT_TYPE  VARCHAR2(100),
           CONTENT       BLOB
         )
```

which stores files you wish to attach to the message.

```
DECLARE
   msg    PSP_Mail.Strings;
   res    PSP_Mail.Strings;
   ATT    PSP_Mail.TBAttachments; -- attachments array
   attIdx pls_integer := 1;       -- index to attachments array

PROCEDURE attachFile( att_name VARCHAR2, cid VARCHAR2 DEFAULT NULL )
 -- we create this procedure to simplify ATT array population.
 -- Optional cid parameter may be used to specify unique content ID
 -- for attached file in single call to the procedure.
 IS
 BEGIN
   -- populate array element
   SELECT NAME, CONTENT, CONTENT_TYPE, cid
     INTO ATT(attIdx).name,
          ATT(attIdx).content,
          ATT(attIdx).content_type,
          ATT(attIdx).content_id
     FROM T WHERE NAME = att_name;
   -- advance to the next array element
   attIdx := attIdx + 1;
```



```
 EXCEPTION
   -- ignore errors on SELECT, attIdx will not be incremented if this
   -- happens and ATT array won't be altered
   WHEN NO_DATA_FOUND THEN NULL;
 END;
BEGIN
   attachFile('file1.doc');
   attachFile('file2.doc');
   -- let's print some debug info to be sure we are attaching anything
   dbms_output.put_line('Attaching '||ATT.Count||' files to the message...');
   msg(1) := 'Test Message';
   -- send the message. Note how v_to and v_bcc are used together to hide
   -- the actual recipients list. Recipient will see 'Undisclosed Recipients'
   -- as To: header value and actual recipient list will not appear anywhere
   res := PSP_Mail.SendMailEx2(
                  v_from => '"Me" <mymail@mycompany.com>',
                  v_to => 'Undisclosed Recipients',
                  v_bcc => '<you@yourcompany.com>,<them@theircompany.com>'
                  v_subject => 'Test message',
                  v_body => msg,
                  attachments => ATT);
   FOR i IN res.first..res.last LOOP
      IF res(i) IS NOT NULL THEN
         -- print out error message along with failed email address
         dbms_output.put_line(res(i));
      END IF;
   END LOOP;
END;
```


VERSION FUNCTION.

Version function will return current version of the package as VARCHAR2 string. In previous versions, there was public package variable with the same name, but it was redefined into function to avoid losing variable value due to package state resets using DBMS_SESSION.RESET_PACKAGE.

```
FUNCTION Version RETURN VARCHAR2;
```


CLOB_TO_STRINGS FUNCTION.

This function allows passing `CLOB` as message body to SendMailXXX routines:

```
FUNCTION CLOB_To_Strings( c CLOB ) RETURN Strings;
```

The function takes a `CLOB c` and breaks it down into `PSP_Mail.Strings` collection, which then may be passed as message body to SendMail, SendMailEx or SendMailEx2.

Exceptions: None.

Example of use:

```
DECLARE
        c   CLOB;
        res PSP_Mail.Strings;
BEGIN
 -- create and populate a temporary CLOB with CALL duration
 DBMS_LOB.createTemporary(c, false, dbms_lob.call);
 DBMS_LOB.open(c, dbms_lob.lob_readwrite);
 DBMS_LOB.writeAppend(c, 12, 'Test message');
 res := PSP_Mail.SendMailEx2(v_from => '"Me" <mymail@mycompany.com>',
                             v_to => '"You" <yourmail@yourcompany.com>',
                             v_subject => 'Test message',
                             v_body => PSP_Mail.CLOB_To_Strings(c),
                             attachments => PSP_Mail.TBAtt_Empty);
 FOR i IN res.first..res.last LOOP
  IF res(i) IS NOT NULL THEN
   -- print out error message along with failed email address
   dbms_output.put_line(res(i));
  END IF;
 END LOOP;
 -- dispose the temporary CLOB (it will be done automatically if we
 -- omit this call, but just to be sure it won't consume our resources
 -- for long we do it manually here.)
 DBMS_LOB.freeTemporary(c);
END;
```


ERROR CODES RETURNED BY SENDMAILXXX FUNCTIONS.

SendMailXXX functions report errors via return values. If there was no error, return value is NULL; otherwise it is an error message in special format:

```
[recipient: ] nnn Error message text
```

where nnn is error code and [recipient:] is optional prefix (recipient's email address without square brackets, which are used here to denote that this part of the return value is optional) returned by functions that target multiple recipients which helps identifying the recipient for which the error is reported. Below is the list of all error codes that may be returned by the functions, their meaning and recommended error resolution actions:

505 No recipient specified
Cause: There are no recipients for the message specified.
Action: Specify at least one recipient for the message.

506 No sender specified
Cause: Sender of the message is not specified
Action: Always specify message sender, otherwise the message will not be delivered as most servers are configured to consider messages with NULL sender as spam.

510 MX DNS lookup failed for <recipient address>
Cause: Could not find MX (mail exchanger) record for <recipient address>. May be result of improper PRIMARY_DNS value or non-existent domain in recipient's address.
Will never appear if ALWAYS_RELAY = TRUE.
Action: Verify that PRIMARY_DNS is set correctly and the DNS server is accessible from the Oracle host.
Verify that the recipient address is correct.

511 Recipient rejected, reason: <SMTP reply>
Cause: The SMTP server rejected recipient address. <SMTP reply> is the SMTP server reply string (code and text) describing the reason for rejecting the address.
Action: Check the <SMTP reply> and correct the problem accordingly.

520 Internal error: <SQLERRM>
Cause: Unexpected exception in internal routine.
Should not normally appear. Most probably is due to a bug in the PSP_Mail or one of Oracle-supplied packages.
Action: Apply the latest patch set for your Oracle release and platform and retry the operation that caused error. Ensure that DEFAULT_SMTP and SMTP_PORT together point to correct relay server. If the problem persists, contact N-Networks Support and quote the error message you received, your Oracle software version (including patch set), relay SMTP server brand and version (if used) and provide steps to reproduce the error.

521 Sorry, this TRIAL version is limited to 1 attachment
Cause: Trial version is limited to one attachment per message.
Action: Obtain proper license for the product from N-Networks or its representatives.

550 Sorry, your license has expired.
Cause: Licensing error - license expired.
Action: Obtain proper license for the product from N-Networks or its representatives.

551 License is invalid or not found.
Cause: Licensing error - license key was not found or is invalid (altered.)
Action: Obtain proper license for the product from N-Networks or its representatives.
Install the product through installation script.
Do not alter the license information view.

SUPPLEMENTAL FUNCTIONS AND PROCEDURES FOR RFC-2822 E-MAIL ADDRESS PROCESSING

There are several supplemental functions and procedures that are used internally by SendMailEx2, but may be of help to developers and thus are published in the package specification:

```
FUNCTION extractName( addr VARCHAR2 ) RETURN VARCHAR2;

FUNCTION extractEmail( addr VARCHAR2 ) RETURN VARCHAR2;

PROCEDURE decodeAddressLine(addr VARCHAR2,
                            names IN OUT Strings,
                            emails IN OUT Strings,
                            delimiter VARCHAR2 DEFAULT ',');
```

extractName and **extractEmail** functions extract name and e-mail address parts from an RFC-2822 address respectively. If no literal string is specified in the **addr**, **extractName** returns same value as **extractEmail**, otherwise it returns that literal string, which is usually name of recipient. Both functions may throw VALUE_ERROR exception if result of the function exceeds 10000 characters.

decodeAddressLine procedure takes **addr** parameter which is a list of RFC-2822 addresses, and breaks it down into arrays of **names** and **emails**. You can specify a delimiter character used to delimit addresses in the list, default value for the delimiter is comma character. The procedure uses **extractName** and **extractEmail** functions internally.

Examples of output:

```
PSP_Mail.extractName('"John Doe" <jdoe@somewhere.com>') == 'John Doe'

PSP_Mail.extractEmail('"John Doe" <jdoe@somewhere.com>') == 'jdoe@somewhere.com'


PSP_Mail.decodeAddressLine(
           '"John Doe" <jdoe@somewhere.com>, <rroe@somewhereelse.com>',
           names, emails)
```

will return

```
names(1) == 'John Doe', names(2) == 'rroe@somewhereelse.com'
emails(1) == 'jdoe@somewhere.com', emails(2) == 'rroe@somewhereelse.com'
```


SUPPLEMENTAL PACKAGES.

This chapter describes supplemental packages supplied with PSP_Mail, which are either used by PSP_Mail or provided for user's convenience. These packages may be used standalone and does not require PSP_Mail for their functionality.

UTL_B64 PACKAGE

UTL_B64 package is a supplemental package which implements Base64 encoding and decoding of various Oracle types, including VARCHAR2, RAW and LOBs. This package is used by PSP_Mail for encoding of attachments and may also be used by end users for encoding of the message body and MIME headers.

This document describes package version 1.2.0.

PACKAGE SPECIFICATIONS.

Below you can find specifications and descriptions of subprograms defined in the package, as well as some examples of use.

Subprograms.

encode (RAW)

```
FUNCTION encode (encodeThis IN RAW) RETURN VARCHAR2;
```

encode function is a generic wrapper around Base64 Java class method. It takes a RAW value to be encoded and returns its Base64 representation as VARCHAR2. Note that due to the nature of Base64 encoding and PL/SQL VARCHAR2 type size limit, encodeThis may be up to 24kb in size – this is the maximum input size for which Base64 output will fit 32kb VARCHAR2 limit. For larger inputs, VALUE_ERROR exception will be raised.

encodev

```
FUNCTION encodev (encodeThis IN VARCHAR2) RETURN VARCHAR2;
```

encodev function is similar to encode but takes a VARCHAR2 argument. The same restriction on input size applies to this function.

encodeva

```
FUNCTION encodeva (encodeThis IN vc_arr) RETURN vc_arr;
```

encodeva is yet another version of encode which takes an array of VARCHAR2 (vc_arr, subtype of the PSP_Mail.Strings type) and returns another array with encoded representation of the source. There are no size restrictions on individual array elements, except natural VARCHAR2 limit of 32k.

encode (LOB)

```
FUNCTION encode (encodeThis IN BLOB) RETURN BLOB;
FUNCTION encode (encodeThis IN CLOB) RETURN CLOB;
```

These are two overloaded versions of encode which deal with LOBs. Each takes a source LOB and returns its encoded representation. Encoded LOB contains Base64 encoding of the source LOB with 76 Base64 characters per line followed by CR/LF. This LOB is TEMPORARY and should be freed with DBMS_LOB.FREETEMPORARY when no longer needed (or Oracle will automatically dispose this LOB at the end of the session at expense of extra LOB handle and temporary space being used while session is still active).

encode_mime_header

```
FUNCTION encode_mime_header( encodeThis     IN VARCHAR2,
                             encoding       IN VARCHAR2,
                             auto_translate Boolean := False )
                           RETURN VARCHAR2;
```

This function encodes `encodeThis` string so that it can be issued in MIME header (according to RFC-2047). English strings need not be encoded this way, but strings with national characters should be encoded. Function correctly breaks long strings so that resulting encoded string does not have lines longer than 76 characters. The `encoding` should be valid IANA character set identifier.
When `auto_translate` is set to `TRUE`, the function will attempt to obtain Oracle character set matching the `encoding` and automatically convert `encodeThis` to this character set before encoding it. This feature is only supported on Oracle releases exposing `UTL_GDK` (Oracle8i Version 8.1.7.4 and Oracle9i) or `UTL_I18N` (Oracle10g and later) packages. If the function is unable to obtain proper Oracle character set name that matches `encoding`, no translation is performed. You should test this feature on your Oracle release before using it in production environments.

decode (RAW)

```
FUNCTION decode (decodeThis IN VARCHAR2) RETURN RAW;
```

`decode` function is a generic wrapper around Base64 Java class method. It takes a `VARCHAR2` string encoded with Base64 encoding and returns `RAW` decoded data.

decodev

```
FUNCTION decodev (decodeThis IN VARCHAR2) RETURN VARCHAR2;
```

`decodev` is similar to `decode` but returns decoded data as `VARCHAR2`.

decode (LOB)

```
FUNCTION decode (decodeThis IN CLOB) RETURN CLOB;
FUNCTION decode (decodeThis IN BLOB) RETURN BLOB;
```

These are overloaded LOB versions of `decode`. They decode `decodeThis` LOB and returns LOB with decoded data. Function expects that `decodeThis` LOB contains valid Base64-encoded data (with no more than 76 Base64 characters per line and CR/LF as line terminator). Returned LOB is TEMPORARY and should be freed with `DBMS_LOB.FREETEMPORARY` when no longer needed (or Oracle will automatically dispose this LOB at the end of the session at expense of extra LOB handle and temporary space being used while session is still active).

getVersion

```
FUNCTION getVersion RETURN VARCHAR2;
```

This function returns current version of the `UTL_B64` package.

EXAMPLES OF USE.

Below are some examples of UTL_B64 use. Please note that "iana-cs" character set used in examples is fictious and should be replaced with valid IANA-registered character set name in real applications. You will probably use the IANA character set corresponding to the Oracle database character set.

Encoding Message Body and Sending Email With Encoded Body.

```
DECLARE
  Msg PSP_Mail.Strings;
  ret PSP_Mail.Strings;
BEGIN
  Msg(1) := 'Some text';
  Msg := utl_b64.encodeva(Msg);
  ret := PSP_Mail.SendMailEx2(
            v_from => '"Me" me@here',
            v_to => 'someone@somewhere',
            v_subject => 'some subject',
            v_body => Msg,
            other_headers => 'Content-Transfer-Encoding: base64',
            content_type => 'text/plain; charset="iana-cs"'
            attachments => PSP_Mail.tbatt_empty);
  FOR i IN ret.First..ret.Last LOOP
   IF ret(i) IS NOT NULL THEN
    -- report an error here
   END IF;
  END LOOP;
END;
```

In the above example, the message was encoded using Base64 encoding and sent in encoded form. Content-Type header indicates that the message body is in 'iana-cs' character set. If you do not specify content_type, your message body is likely to be converted to 8bit encoding by the SMTP server since default character set does not need to be encoded.

Encoding Message Body and Subject.

```
DECLARE
  Msg PSP_Mail.Strings;
  ret PSP_Mail.Strings;
BEGIN
  Msg(1) := 'Some text';
  Msg := utl_b64.encodeva(Msg);
  ret := PSP_Mail.SendMailEx2(
            v_from => '"Me" me@here',
            v_to => 'someone@somewhere',
            v_subject => utl_b64.encode_mime_header('some subject','iana-cs');
            v_body => Msg,
            other_headers => 'Content-Transfer-Encoding: base64',
            attachments => PSP_Mail.tbatt_empty);
  FOR i IN ret.First..ret.Last LOOP
   IF ret(i) IS NOT NULL THEN
    -- report an error here
   END IF;
  END LOOP;
END;
```

In the above example, subject of the message was also encoded and 'iana-cs' was assigned as character set of the Subject header value.

UTL_BINFILE PACKAGE

Version 1.0.7, March 2004

UTL_BinFile package allows PL/SQL developers to read and write binary OS files and files in ZIP/JAR archives into/from BLOBs, delete and rename files and directories, create new directories, compress/decompress BLOBs using popular 'deflate' and 'gzip' compression methods, and create and modify ZIP archives. The package relies on com.nnetworks.FileOperations Java class. Correct java.io.FilePermission permissions should be granted to the package caller using DBMS_JAVA.GRANT_PERMISSION() to allow access to the OS files and directories. Without proper permissions the package will not be able to read or write OS files/directories. Note that UTL_BinFile package and com.nnetworks.FileOperations class are loaded with INVOKER (AUTHID CURRENT_USER) rights, public synonyms are created and execute permissions are granted to PUBLIC, which means the package will run with the privileges of the calling user and all users are allowed to call it. This allows fine-tuning access permissions for each Oracle account. For example, user SCOTT may be granted certain java.io.FilePermission for some directories or files and will be able to read or write these files, while user BLAKE may not be granted any permissions and will be unable to read any files while still being able to invoke the package – the package will throw an exception if correct permissions for the requested file or directory are not granted to the calling user. Refer to Oracle Java Developer's Guide for more information on DBMS_JAVA package and Java2 security. Note that Java security policy changes do not take effect in active sessions; only new sessions will be affected as the policy is loaded once at session startup and is not re-read afterwards.

Be sure to check your security policy settings thoroughly – failure to do so (for example, granting too wide permissions to an unprivileged Oracle account) may open a serious security hole in your system. Never load the UTL_BinFile package with AUTHID DEFINER (or no AUTHID) – invoking the package with definer rights may allow invokers to access the file system objects they normally denied access to (especially if you load the package into SYS schema – in this case all file system objects accessible to Oracle software owner will also be accessible to any unprivileged Oracle user). Also note that packages loaded into SYS schema are normally protected from web access through the PL/SQL gateway (mod_plsql denies access to SYS.* by default), but your DAD (Data Access Descriptor) exclusion_list parameter, which controls which schemas and packages are inaccessible from the web, may be overridden – in this case default mod_plsql protection may not be in effect and SYS.* may need to be added to the list of excluded packages, otherwise malicious user could remotely execute PL/SQL and/or Java code on your system by means of simple HTTP calls.

PACKAGE SPECIFICATIONS.

Below you can find specifications and descriptions of subprograms defined in the package and supporting database structures, as well as some examples of their use.

Tables.

NN$FOPS$DIR_LIST

```
GLOBAL TEMPORARY TABLE NN$FOPS$DIR_LIST ( FILENAME VARCHAR2(1024),
                                          FILETYPE VARCHAR2(4),
                                          FILESIZE NUMBER(12,0),
                                          MTIME    DATE )
ON COMMIT DELETE ROWS
```

This global temporary table is used to return directory and archive listings. See description of the List_Directory procedure below for details on this table and its use. PUBLIC is granted all DML rights to this table.

Exceptions.

JAVA_ERROR

```
EXCEPTION JAVA_ERROR;
PRAGMA EXCEPTION_INIT(JAVA_ERROR, -29532);
```


The JAVA_ERROR exception is raised when any error in the underlying Java code occurs. Oracle does not distinguish between different exceptions in the server-side Java code and always raises this exception when something goes wrong. The exception message usually contains detailed error description. This exception maps to the ORA-29532: Java call terminated by uncaught Java exception: string error.

Subprograms.

Load_BLOB_From_File

Prototype:

```
FUNCTION Load_BLOB_From_File ( filename IN VARCHAR2 ) RETURN BLOB;
```

This function creates a temporary BLOB and loads OS binary file fully qualified by filename parameter into it. The returned BLOB should be freed using DBMS_LOB.FREETEMPORARY when no longer needed (or it will be disposed automatically when session ends). The filename parameter is case-sensitive. If the file does not exist, JAVA_ERROR exception is raised.

Save_BLOB_To_File

Prototype:

```
PROCEDURE Save_BLOB_To_File ( b IN BLOB, filename IN VARCHAR2 );
```

This procedure saves BLOB b into the OS file fully qualified by filename parameter. filename parameter is case-sensitive. If the file cannot be written due to security violation or space issues, JAVA_ERROR exception is raised. If the file does not exist, it is automatically created, and if it already exists, it is overwritten with the b BLOB content.

List_Directory

Prototype:

```
PROCEDURE List_Directory( v_directory IN VARCHAR2,
                          list_hidden IN BOOLEAN DEFAULT False );
```

This procedure attempts to list files in the OS directory specified by v_directory. If correct permissions are set in the Java policy table using the DBMS_JAVA.GRANT_PERMISSION() procedure and file system permissions allow the account running Oracle software to read the directory, names of its files and subdirectories are inserted into the global temporary table **NN$FOPS$DIR_LIST** where they persist until the transaction is committed or until session ends. The columns of NN$FOPS$DIR_LIST table are **FILENAME**, **FILETYPE**, **FILESIZE** and **MTIME**. FILENAME is the name of the file (without leading path, which is v_directory, for regular directories, or with full path from archive root for ZIP/JAR archives), FILETYPE is either 'FILE' for plain files or 'DIR' for subdirectories, FILESIZE is the size of the file in bytes (0 for directories, -1 if not known), and MTIME is the file modification timestamp. Hidden files and directories are not listed by default unless list_hidden parameter overrides this behavior. Retrieved file names are case-sensitive. The v_directory parameter should contain only the directory name, additional file masks are not supported. You can apply a filter mask on file names and other file attributes later when retrieving file names from the NN$FOPS$DIR_LIST table, for example:

```
SELECT FILENAME
  FROM NN$FOPS$DIR_LIST
WHERE UPPER(FILENAME) LIKE '%.DOC'
      AND MTIME > TO_DATE('01 Jan, 2000','DD Mon, YYYY')
```


Load_BLOB_From_ZIP

Prototypes:

```
FUNCTION Load_BLOB_From_ZIP( zip_file IN VARCHAR2,
                             filename IN VARCHAR2) RETURN BLOB;
```

This function loads a file identified by `filename` from a ZIP or JAR OS file `zip_file` into a temporary `BLOB`. The returned `BLOB` should be freed using `DBMS_LOB.FREETEMPORARY()` when no longer needed (or it will be destroyed automatically when session ends). The `filename` parameter is case-sensitive. If the file is not found in the archive, the function returns `NULL`. If the ZIP/JAR archive is inaccessible or does not exist, `JAVA_ERROR` exception is raised.

```
FUNCTION Load_BLOB_From_ZIP( zip_file IN BLOB,
                             filename IN VARCHAR2) RETURN BLOB;
```

This function loads a file identified by `filename` from a ZIP or JAR archive `zip_file` stored as internal `BLOB` into a temporary `BLOB`. The returned temporary `BLOB` should be freed using `DBMS_LOB.FREETEMPORARY()` when no longer needed (or it will be destroyed automatically when session ends). The `filename` parameter is case-sensitive. If the file is not found in the archive, the function returns `NULL`.

List_ZIP

Prototypes:

```
PROCEDURE List_ZIP(zip_file IN VARCHAR2);
```

This procedure is similar to the `List_Directory`, but it fills the **NN$FOPS$DIR_LIST** table with the list of files in the `zip_file` ZIP or JAR archive stored as an external OS file. Calling user should have correct Java permissions granted to him to be able to open the `zip_file`. Retrieved listing persists in the **NN$FOPS$DIR_LIST** table until commit or rollback. Note that file names in ZIP or JAR archive may be prefixed with relative path. The procedure currently does not list directories in the ZIP file if they do not have an entry in the ZIP file directory. Directory paths that may prefix file names are not automatically extracted and stripped from file names. Retrieved file names are case-sensitive. Also note that file names in a ZIP archive may be up to 65535 bytes long, but the size of `FILENAME` column in `NN$FOPS$DIR_LIST` table is limited to 1024 bytes. All names that exceed this 1024 bytes limit are trimmed to fit the column. If the referenced ZIP file is inaccessible or does not exist, `JAVA_ERROR` exception is raised.

```
PROCEDURE List_ZIP(zip_file IN BLOB);
```

Overloaded version of `List_ZIP` procedure that retrieves listing of ZIP/JAR archive stored as an internal `BLOB` instead of external OS file.

Delete_File

Prototype:

```
PROCEDURE Delete_File(filename IN VARCHAR2);
```

This procedure attempts to delete a file identified by `filename`. The procedure will raise `JAVA_ERROR` exception if the file does not exist or could not be deleted due to security or file system restrictions.

Rename_File

Prototype:

```
PROCEDURE Rename_File(fnFrom IN VARCHAR2,
                      fnTo   IN VARCHAR2);
```


This procedure attempts to rename a file named fnFrom to fnTo. The procedure will raise JAVA_ERROR exception if the fnFrom file does not exist or could not be renamed to the new name due to security or file system restrictions (for example, if the file with requested new name already exists.) The procedure will **not** move the source file to a new location in the file system, renaming operation that could cause file movement will fail.

Copy_File

Prototype:

```
PROCEDURE Copy_File(fnFrom IN VARCHAR2,
                    fnTo   IN VARCHAR2 );
```

This procedure attempts to create a copy of a file identified by fnFrom with the new name of fnTo. The procedure will raise JAVA_ERROR exception if the fnFrom file was not found or the fnTo file could not be created or is not writeable. The caller should have adequate write permissions on the fnTo location; otherwise the copy operation will fail.

Move_File

Prototype:

```
PROCEDURE Move_File(fnFrom IN VARCHAR2,
                    fnTo   IN VARCHAR2 );
```

This procedure attempts to move a file identified by fnFrom to the new location fnTo. The procedure will raise JAVA_ERROR exception if the fnFrom file was not found or could not be deleted, or the fnTo file could not be created or is not writeable. The caller should have write permissions on both fnFrom and fnTo files, otherwise the move operation will fail. If the source file can't be deleted, the whole move operation will be undone and target file will be removed.

Create_Directory

Prototype:

```
PROCEDURE Create_Directory(dirname IN VARCHAR2, recursive IN BOOLEAN := FALSE);
```

This procedure will attempt to create new directory dirname. If recursive is set to TRUE, the procedure will also attempt to create all parent directories as necessary (that is, it will create the whole directory chain.) The procedure will raise JAVA_ERROR exception if the directory or any of its parent directories could not be created due to security or file system restrictions. Note that if the procedure fails in recursive mode it may have succeeded in creating some of the parent directories.

Deflate

Prototype:

```
FUNCTION Deflate( in_blob IN BLOB ) RETURN BLOB;
```

This function compresses in_blob BLOB using 'deflate' compression algorithm and returns its compressed representation as TEMPORARY BLOB. Returned BLOB should be freed using DBMS_LOB.FREETEMPORARY() when no longer needed, or it will be released automatically when session ends. The function may raise JAVA_ERROR exception if the input BLOB could not be compressed. Exception message will have error details.
This function can be used in SQL.

GZIP

Prototype:


```
FUNCTION GZIP( in_blob IN BLOB ) RETURN BLOB;
```

This function compresses in_blob BLOB using 'gzip' compression algorithm and returns its compressed representation as TEMPORARY BLOB. Returned BLOB should be freed using DBMS_LOB.FREETEMPORARY() when no longer needed, or it will be released automatically when session ends. The function may raise JAVA_ERROR exception if the input BLOB could not be compressed. Exception message will have error details.
This function can be used in SQL.

Inflate

Prototype:

```
FUNCTION Inflate( in_blob IN BLOB ) RETURN BLOB;
```

This function decompresses in_blob BLOB compressed using 'deflate' compression algorithm and returns its uncompressed representation as TEMPORARY BLOB. Returned BLOB should be freed using DBMS_LOB.FREETEMPORARY() when no longer needed, or it will be released automatically when session ends. The function does not check if the in_blob was indeed compressed before attempting to decompress it. The function may raise JAVA_ERROR exception if the input BLOB could not be decompressed. Exception message will have error details.
This function can be used in SQL.

UnGZIP

Prototype:

```
FUNCTION UnGZIP( in_blob IN BLOB ) RETURN BLOB;
```

This function decompresses in_blob BLOB compressed using 'gzip' compression algorithm and returns its uncompressed representation as TEMPORARY BLOB. Returned BLOB should be freed using DBMS_LOB.FREETEMPORARY() when no longer needed, or it will be released automatically when session ends. The function does not check if the in_blob was indeed compressed before attempting to decompress it. The function may raise JAVA_ERROR exception if the input BLOB could not be decompressed. Exception message will have error details.
This function can be used in SQL.

Create_ZIP

Prototype:

```
FUNCTION Create_ZIP( in_blob           IN BLOB
                    ,filename          IN VARCHAR2
                    ,compression_level IN NUMBER := 9 ) RETURN BLOB;
```

This function compresses in_blob using 'deflate' algorithm with specified compression level (defaults to 9 - maximum compression) and returns ZIP archive with compressed in_blob named as filename. Returned BLOB is TEMPORARY and should be freed using DBMS_LOB.FREETEMPORARY() when no longer needed, or it will be released automatically when session ends. The function may raise JAVA_ERROR exception if the input BLOB could not be compressed or ZIP archive could not be created. Exception message will have error details.

Add_To_ZIP

Prototype:

```
PROCEDURE Add_To_ZIP( zip_file          IN OUT NOCOPY BLOB
                     ,in_blob           IN BLOB
                     ,filename          IN VARCHAR2
                     ,compression_level IN NUMBER := 9);
```


This procedure takes a zip archive `zip_file` and adds `in_blob` to the archive with file name of `filename` and compression level of `compression_level` (using 'deflate' algorithm). If the `zip_file` is empty, new archive will be automatically created. If the file already exists in the archive, it will be updated (replaced with `in_blob` contents). To add a directory, pass a `NULL in_blob` and make sure that `filename` is terminated with slash (`'/'`). If anything goes wrong, `JAVA_ERROR` exception will be raised; exception message will have error details.

`Delete_From_ZIP`

Prototype:

```
PROCEDURE Delete_From_ZIP( zip_file IN OUT NOCOPY BLOB
                          ,filename IN VARCHAR2);
```

This procedure attempts to delete a file identified by `filename` from the ZIP archive in `zip_file`. If the file is not there, nothing will happen. If anything goes wrong, `JAVA_ERROR` exception will be raised; exception message will have error details.

`getVersion`

Prototype:

```
FUNCTION getVersion RETURN VARCHAR2;
```

This function returns string representation of the package version.

EXAMPLES OF USE.

Setting Java Access Permissions.

The following call will grant user SCOTT read and write permissions to the directory /usr/local/scott and all of its files and subdirectories. Please note that in addition to this grant the directory should be accessible to Oracle itself (with correct permissions set on OS level). Also note that current active sessions will not see the changes in security policy – only new Oracle sessions will see them.

```
DBMS_JAVA.GRANT_PERMISSION('SCOTT',
                           'java.io.FilePermission',
                           '/usr/local/scott/-',
                           'read,write');
```

For more information on Java2 security and DBMS_JAVA package, please refer to Oracle Java Developer's Guide, Java Stored Procedures Developer's Guide and Java Tools Reference.

Loading and Saving Binary OS Files.

The following code will read a file into BLOB b and then save it under new name.

```
DECLARE
   b BLOB;
BEGIN
   b := UTL_BinFile.Load_BLOB_From_File('/usr/local/scott/binary_file.ext');
   UTL_BinFile.Save_BLOB_To_File(B,'/usr/local/scott/copy_of_binary_file.ext');
END;
```

You can use UTL_BINFILE to load arbitrary OS files and attach them to the email messages, which then can be sent with PSP_Mail:

```
DECLARE
 attachments PSP_Mail.TBAttachments;
[…]
attachments(1).content :=
              UTL_BinFile.Load_BLOB_From_File('/full/os/path/to/file.ext');
attachments(1).name := 'file.ext';
attachments(1).content_type := 'application/octet-stream';
[…]
```

Listing Files in an OS Directory.

The following example will fill the TBAttachments array with .doc files (with case-insensitive extension) found in /tmp/files directory. Hidden .doc files will not be inserted into the array. The code will throw the application exception –20000 if Java code encountered any exception it couldn't handle (unhandled exceptions thrown in Java are not distinguished by Oracle – ORA-29532 is always thrown in PL/SQL for any unhandled Java exception.)

```
DECLARE
  attachments PSP_Mail.TBAttachments;
  cnt         PLS_INTEGER := 1;
  -- declare and initialize 'unhandled Java exception' exception
  JAVA_EXCEPTION EXCEPTION;
  PRAGMA EXCEPTION_INIT(JAVA_EXCEPTION, -29532);
BEGIN
 BEGIN
  UTL_BinFile.List_Directory('/tmp/files');
  FOR file IN (SELECT filename FROM nn$fops$dir_list
               WHERE filetype = 'FILE' AND UPPER(filename) LIKE '%.DOC' )
   LOOP
    attachments(cnt).content :=
```



```
          UTL_BinFile.Load_BLOB_From_File('/tmp/files/'||file.filename);
    attachments(cnt).name := file.filename;
    attachments(cnt).content_type := 'application/msword';
    cnt := cnt + 1;
   END LOOP;
  COMMIT; -- clear the NN$FOPS$DIR_LIST table
 EXCEPTION
   WHEN JAVA_EXCEPTION THEN
    -- unhandled Java exception caught, probably no
    -- permission for the directory or file
     RAISE_APPLICATION_ERROR(-20000, 'Access denied.');
   WHEN OTHERS THEN
     RAISE; -- something different, rethrow the exception
 END;
[…]
END;
```

Listing Files in a ZIP Archive.

The following example demonstrates how to use the **List_ZIP** procedure to obtain the list of files stored in an external ZIP archive (Java permission to read the `c:\oracle\ora81\plsql\jlib\` directory and all files in it was granted to the user executing the sample code):

```
SQL> EXEC UTL_BinFile.List_ZIP('c:\oracle\ora81\plsql\jlib\plsql.jar')

PL/SQL procedure successfully completed.

SQL> COLUMN filename FORMAT A70
SQL> SELECT filename, filetype ft FROM nn$fops$dir_list;

FILENAME                                                               FT
---------------------------------------------------------------------- ----
META-INF/                                                              DIR
META-INF/MANIFEST.MF                                                   FILE
oracle/plsql/net/TCPConnection.class                                   FILE
oracle/plsql/net/InternetAddress.class                                 FILE
```

Note that in the example above, all files are prefixed with absolute paths from the archive root, while only one directory is listed. `oracle`, `plsql` and `net` directories do not have separate entries in the ZIP archive central directory.

Extracting Files from a ZIP Archive.

The following example will extract a file from a ZIP archive stored as internal `BLOB` (we assume that the archive is stored in `ZIP_FILES` table):

```
DECLARE
   ZIP  BLOB;
   F    BLOB;
BEGIN
   SELECT ARCHIVE INTO ZIP
     FROM ZIP_FILES
    WHERE NAME = 'MyZipFile.zip';
   F := UTL_BinFile.Load_BLOB_From_ZIP(ZIP, 'MyArchivedFile.doc');
   IF (F IS NOT NULL) THEN
     -- file found and extracted, process it
     NULL;
   END IF;
END;
```


Compressing and Decompressing a BLOB.

The following example will compress and update an internal BLOB with compressed representation using 'deflate' compression (the example uses fictious FILES table, which is assumed to have BLOB_COL BLOB, NAME VARCHAR2 and COMPRESSED VARCHAR2 columns):

```
UPDATE FILES
   SET BLOB_COL   = UTL_BinFile.Deflate(BLOB_COL),
       COMPRESSED = 'DEFLATE'
 WHERE NAME = 'mydocument.doc';
```

The following example function will extract decompressed BLOB, automatically selecting correct decompression algorithm based on COMPRESSED column value:

```
CREATE OR REPLACE FUNCTION getMyBLOB( in_name IN FILES.NAME%TYPE )
AS
  R FILES%ROWTYPE;
BEGIN
 SELECT * INTO R
   FROM FILES
  WHERE NAME = in_name;
 IF R.COMPRESSED = 'DEFLATE' THEN
   RETURN UTL_BinFile.Inflate(R.BLOB_COL);
 ELSIF R.COMPRESSED = 'GZIP' THEN
   RETURN UTL_BinFile.UnGZIP(R.BLOB_COL);
 ELSE
   -- assume the BLOB is not compressed
   RETURN R.BLOB_COL;
 END IF;
END getMyBLOB;
/
```

Using the function above, you can easily extract uncompressed BLOBs from the FILES table regardless their actual compression state:

```
SELECT getMyBLOB(BLOB_COL)
  FROM FILES
 WHERE NAME = 'mydocument.doc';
```

Creating and Populating a ZIP archive.

The following example will create a ZIP archive and add several files and directories to it:

```
DECLARE
  ZIP  BLOB; -- BLOB to hold the archive
  R    FILES%ROWTYPE;
BEGIN
  DBMS_LOB.createTemporary(ZIP, TRUE);
  -- adding a directory
  UTL_BinFile.Add_To_ZIP(ZIP, NULL, 'Directory/');
  -- adding files
  FOR R IN ( SELECT * FROM FILES
              WHERE NAME IN ('Document1.doc', 'Document2.doc') ) LOOP
    UTL_BinFile.Add_To_ZIP(ZIP, R.BLOB_COL, 'Directory/'||R.NAME);
  END LOOP;
END;
```


Note that you can easily manipulate ZIP archives stored externally (in the file system) as well: you first load an archive into a `BLOB` using `Load_BLOB_From_File()` function, alter it as needed and then save it back using `Save_BLOB_To_File()` procedure. Because of this, separate overloaded routines for manipulating external ZIP archives were not implemented.

