120
DBMS_LOB

The `DBMS_LOB` package provides subprograms to operate on `BLOB`s, `CLOB`s, `NCLOB`s, `BFILE`s, and temporary `LOB`s. You can use `DBMS_LOB` to access and manipulate specific parts of a LOB or complete LOBs.

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Datatypes
- Operational Notes
- Rules and Limits
- Exceptions
- Summary of DBMS_LOB Subprograms

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide

DBMS_LOB Overview

`DBMS_LOB` can read and modify `BLOB`s, `CLOB`s, and `NCLOB`s; it provides read-only operations for `BFILE`s. The bulk of the LOB operations are provided by this package.

DBMS_LOB Security Model

This package must be created under `SYS`. Operations provided by this package are performed under the current calling user, not under the package owner `SYS`.

Any `DBMS_LOB` subprogram called from an anonymous PL/SQL block is executed using the privileges of the current user.

When creating the procedure, users can set the `AUTHID` to indicate whether they want definer's rights or invoker's rights. For example:

```
CREATE PROCEDURE proc1 AUTHID DEFINER ...
```

or

```
CREATE PROCEDURE proc1 AUTHID CURRENT_USER ...
```

See Also:

For more information on AUTHID and privileges, see *Oracle Database PL/SQL Language Reference*

You can provide secure access to BFILEs using the DIRECTORY feature discussed in BFILENAME function in the*Oracle Database SecureFiles and Large Objects Developer's Guide* and the *Oracle Database SQL Language Reference*.

For information about the security model pertaining to temporary LOBs, see Operational Notes.

DBMS_LOB Constants

This topic describes the constants used by the DBMS_LOB package

These are shown in following tables:

- Table 120-1
- Table 120-2
- Table 120-3
- Table 120-4
- Table 120-5
- Table 120-6

Table 120-1 DBMS_LOB Constants - Basic

Constant	Type	Value	Description
CALL	PLS_INTEGER	12	Create the TEMP LOB with call duration
FILE_READONLY	BINARY_INTEGER	0	Open the specified BFILE read-only
LOB_READONLY	BINARY_INTEGER	0	Open the specified LOB read-only
LOB_READWRITE	BINARY_INTEGER	1	Open the specified LOB read-write
LOBMAXSIZE	INTEGER	18446744073709551615	Maximum size of a LOB in bytes
SESSION	PLS_INTEGER	10	Create the TEMP LOB with session duration

Table 120-2 DBMS_LOB Constants - Option Types

Constant	Definition	Value	Description
OPT_COMPRESS	BINARY_INTEGER	1	Set/Get the SECUREFILE compress option value
OPT_DEDUPLICATE	BINARY_INTEGER	4	Set/Get the SECUREFILE Deduplicate option value

Table 120-2 (Cont.) DBMS_LOB Constants - Option Types

Constant	Definition	Value	Description
`OPT_ENCRYPT`	`BINARY_INTEGER`	2	Get the `SECUREFILE` encrypt option value

Table 120-3 DBMS_LOB Constants - Option Values

Constant	Definition	Value	Description
`COMPRESS_OFF`	`BINARY_INTEGER`	0	For SETOPTIONS Procedures, set compress off; for GETOPTIONS Functions, compress is off
`COMPRESS_ON`	`BINARY_INTEGER`	1	For SETOPTIONS Procedures, set compress on; for GETOPTIONS Functions, compress is on
`DEDUPLICATE_OFF`	`BINARY_INTEGER`	0	For SETOPTIONS Procedures, set deduplicate is off; for GETOPTIONS Functions, deduplicate is off
`DEDUPLICATE_ON`	`BINARY_INTEGER`	4	For SETOPTIONS Procedures, set deduplicate is on; for GETOPTIONS Functions, deduplicate is on
`ENCRYPT_OFF`	`BINARY_INTEGER`	0	For GETOPTIONS Functions, encrypt is off
`ENCRYPT_ON`	`BINARY_INTEGER`	2	For GETOPTIONS Functions, encrypt is on

Table 120-4 DBMS_LOB Constants - DBFS State Value Types

Constant	Definition	Value	Description
`DBFS_LINK_NEVER`	`PLS_INTEGER`	0	LOB has never been archived
`DBFS_LINK_NO`	`PLS_INTEGER`	2	LOB was archived, but as been read back in to the RDBMS
`DBFS_LINK_YES`	`PLS_INTEGER`	1	LOB is currently archived

Table 120-5 DBMS_LOB Constants - DBFS Cache Flags

Constant	Definition	Value	Description
`DBFS_LINK_CACHE`	`PLS_INTEGER`	1	Put the LOB data to the archive, but keep the data in the RDBMS as a cached version
`DBFS_LINK_NOCACHE`	`PLS_INTEGER`	0	Put the LOB data to the archive, and remove the data from the RDBMS.

Table 120-6 DBMS_LOB Constants - Miscellaneous

Constant	Definition	Value	Description
`CONTENTTYPE_MAX_SIZE`	`PLS_INTEGER`	128	Maximum number of bytes allowed in the content type string

Table 120-6 (Cont.) DBMS_LOB Constants - Miscellaneous

Constant	Definition	Value	Description
`DBFS_LINK_PATH_MAX_SIZE`	`PLS_INTEGER`	`1024`	The maximum length of DBFS pathnames

DBMS_LOB Datatypes

The table in this topic describes the datatypes used by DBMS_LOB.

.

Table 120-7 Datatypes Used by DBMS_LOB

Type	Description
`BLOB`	Source or destination binary LOB.
`RAW`	Source or destination `RAW` buffer (used with `BLOB`).
`CLOB`	Source or destination character LOB (including `NCLOB`).
`VARCHAR2`	Source or destination character buffer (used with `CLOB` and `NCLOB`).
`INTEGER`	Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.
`BFILE`	Large, binary object stored outside the database.

The `DBMS_LOB` package defines no special types.

An `NCLOB` is a `CLOB` for holding fixed-width and varying-width, multibyte national character sets.

The clause `ANY_CS` in the specification of `DBMS_LOB` subprograms for `CLOB`s enables the `CLOB` type to accept a `CLOB` or `NCLOB` locator variable as input.

DBMS_LOB Operational Notes

All `DBMS_LOB` subprograms work based on LOB locators. For the successful completion of `DBMS_LOB` subprograms, you must provide an input locator that represents a LOB that already exists in the database tablespaces or external file system.

See also Chapter 1 of *Oracle Database SecureFiles and Large Objects Developer's Guide*

Starting from 12.2 release, you can select a persistent LOB locator from a remote table into a local variable. The remote column can be of type BLOB, CLOB, or NCLOB. You cannot select `BFILE` from a remote table. The LOB variable that refers to the LOB value in a remote table is called a remote locator.

All the `DBMS_LOB` APIs other than the ones that are meant for BFILEs will now accept and support operations on remote LOB locators. All the APIs that take in two locators must have both LOBs collocated at one database.

See Also:

Distributed LOBs chapter in *Oracle Database SecureFiles and Large Objects Developer's Guide*.

To use LOBs in your database, you must first use SQL data definition language (DDL) to define the tables that contain LOB columns.

- Internal LOBs
- External LOBs
- Temporary LOBs

Internal LOBs

To populate your table with internal LOBs after LOB columns are defined in a table, you use the SQL data manipulation language (DML) to initialize or populate the locators in the LOB columns.

External LOBs

For an external LOB (BFILE) to be represented by a LOB locator, you must:

- Ensure that a `DIRECTORY` object representing a valid, existing physical directory has been defined, and that physical files (the LOBs you plan to add) exist with read permission for the database. If your operating system uses case-sensitive path names, then be sure you specify the directory in the correct format.
- Pass the `DIRECTORY` object and the filename of the external LOB you are adding to the `BFILENAME` function to create a LOB locator for your external LOB.

Once you have completed these tasks, you can insert or update a row containing a LOB column using the specified LOB locator.

After the LOBs are defined and created, you can then `SELECT` from a LOB locator into a local PL/SQL LOB variable and use this variable as an input parameter to `DBMS_LOB` for access to the LOB value.

For details on the different ways to do this, See *Oracle Database SecureFiles and Large Objects Developer's Guide*

Temporary LOBs

The database supports the definition, creation, deletion, access, and update of temporary LOBs. Your temporary tablespace stores the temporary LOB data. Temporary LOBs are not permanently stored in the database. Their purpose is mainly to perform transformations on LOB data.

For temporary LOBs, you must use the OCI, PL/SQL, or another programmatic interface to create or manipulate them. Temporary LOBs can be either `BLOB`s, `CLOB`s, or `NCLOB`s.

A temporary LOB is empty when it is created. By default, all temporary LOBs are deleted at the end of the session in which they were created. If a process dies unexpectedly or if the database crashes, then temporary LOBs are deleted, and the space for temporary LOBs is freed.

There is also an interface to let you group temporary LOBs together into a logical bucket. The duration represents this logical store for temporary LOBs. Each temporary LOB can have separate storage characteristics, such as CACHE/ NOCACHE. There is a default store for every session into which temporary LOBs are placed if you don't specify a specific duration. Additionally, you are able to perform a free operation on durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or transaction management for temporary LOBs. Because CR and roll backs are not supported for temporary LOBs, you must free the temporary LOB and start over again if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is potentially a performance impact if you assign multiple locators to the same temporary LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while another locator is also pointing to it. The locator on which a modification was performed now points to a new copy of the temporary LOB. Other locators no longer see the same data as the locator through which the modification was made. A deep copy was not incurred by permanent LOBs in these types of situations, because CR snapshots and version pages enable users to see their own versions of the LOB cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having multiple pointers to locators point to the same temporary LOB locator, if necessary. In PL/SQL, you must avoid using more than one locator for each temporary LOB. The temporary LOB locator can be passed by reference to other procedures.

Because temporary LOBs are not associated with any table schema, there are no meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary LOB instance by a user causes the engine to create and return a locator to the LOB data. The PL/SQL DBMS_LOB package, PRO*C/C++, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server.

Temporary LOBs do not support the EMPTY_BLOB or EMPTY_CLOB functions that are supported for permanent LOBs. The EMPTY_BLOB function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the DBMS_LOB package by using the appropriate FREETEMPORARY or OCIDurationEnd statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and DBMS_LOB statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or DBMS_LOB COPY command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

The database keeps track of temporary LOBs for each session in a v$ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist for each session. V$ views are for DBA use. From the session, the database can determine which user owns the temporary LOBs. By using V$TEMPORARY_LOBS in conjunction with DBA_SEGMENTS, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

The following notes are specific to temporary LOBs:

1. All functions in `DBMS_LOB` return `NULL` if any of the input parameters are `NULL`. All procedures in `DBMS_LOB` raise an exception if the LOB locator is input as `NULL`.
2. Operations based on `CLOBs` do not verify if the character set IDs of the parameters (`CLOB` parameters, `VARCHAR2` buffers and patterns, and so on) match. It is the user's responsibility to ensure this.
3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.

See Also:

Oracle Database PL/SQL Language Reference for more information on `NOCOPY` syntax

DBMS_LOB Rules and Limits

This topic describes general DBMS_LOB rules and limits, rules and limits specific to external files (BFILEs), and maximum LOB and buffer sizes.

General Rules and Limits

- Oracle Database does not support constraints on columns or attributes whose type is a LOB, with the following exceptions:
  - `NOT NULL` constraints are supported for an LOB column or attribute `IS JSON FORMAT JSON` constraints trigger a JSON validation when you close a LOB.
  - Oracle does not recommend that you use `DBMS_LOB` to modify LOBs under `IS JSON FORMAT JSON` constraints. Because LOBs under `IS JSON FORMAT JSON` constraints are not validated, you can only check that the LOB content is a valid JSON. However, if required, Oracle recommends that you wrap a LOB operation inside the Open/Close API to perform write operations on LOBs under the `IS JSON FORMAT JSON` constraints. This validates that the LOB content has valid JSON when you close the LOB. If the LOB is not open, every write operation validates that the LOB content is a valid JSON, and hence, you will not able to write the JSON in pieces.
- The following rules apply in the specification of subprograms in this package:
  - `newlen`, `offset`, and `amount` parameters for subprograms operating on `BLOBs` and `BFILEs` must be specified in terms of *bytes*.
  - `newlen`, `offset`, and `amount` parameters for subprograms operating on `CLOBs` must be specified in terms of *characters*.

  In multi-byte character sets, it is not possible to interpret these offsets correctly.
- A subprogram raises an `INVALID_ARGVAL` exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of `LOB` data are permitted: Negative offsets from the tail of the LOB are not permitted.

2. Only positive, nonzero values are permitted for the parameters that represent size and positional quantities, such as `amount`, `offset`, `newlen`, `nth`, and so on. Negative offsets and ranges observed in SQL string functions and operators are not permitted.
3. The value of `offset`, `amount`, `newlen`, `nth` must not exceed the value `lobmaxsize` 18446744073709551615 (2^{64}-1) in any `DBMS_LOB` subprogram.
4. For `CLOB`s in a database with a multibyte database character set and for NCLOBs, the maximum value for these parameters must not exceed `trunc(lobmaxsize/2)=9223372036854775807` characters.

- PL/SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for `RAW` and `VARCHAR2` parameters used in `DBMS_LOB` subprograms. For example, if you declare a variable to be:

  ```
  charbuf VARCHAR2(3000)
  ```

  Then, `charbuf` can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for `DBMS_LOB` subprograms for `CLOBs` and `NCLOBs`.
- The `%CHARSET` clause indicates that the form of the parameter with `%CHARSET` must match the form of the `ANY_CS` parameter to which it refers.

  For example, in `DBMS_LOB` subprograms that take a `VARCHAR2` buffer parameter, the form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. If the input LOB parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input LOB parameter is of type `CLOB`, then the buffer must contain `CHAR` data.

  For `DBMS_LOB` subprograms that take two `CLOB` parameters, both `CLOB` parameters must have the same form; that is, they must both be `NCLOBs`, or they must both be `CLOBs`.
- If the value of `amount` plus the `offset` exceeds the maximum LOB size allowed by the database, then access exceptions are raised.

  Under these input conditions, read subprograms, such as `READ`, `COMPARE`, `INSTR`, and `SUBSTR`, read until `End of Lob/File` is reached. For example, for a `READ` operation on a `BLOB` or `BFILE`, if the user specifies offset value of 3 GB and an amount value of 2 GB on a LOB that is 4GB in size, then `READ` returns only 1GB (4GB-3GB) bytes.
- Functions with `NULL` or invalid input values for parameters return a `NULL`. Procedures with `NULL` values for destination LOB parameters raise exceptions.
- Operations involving patterns as parameters, such as `INSTR` do not support regular expressions or special matching characters (such as `%` in the `LIKE` operator in SQL) in the `pattern` parameter or substrings.
- The `End Of` LOB condition is indicated by the `READ` procedure using a `NO_DATA_FOUND` exception. This exception is raised only upon an attempt by the user to read beyond the end of the LOB. The `READ` buffer for the last read contains 0 bytes.
- For consistent LOB updates, you must lock the row containing the destination LOB before making a call to any of the procedures (mutators) that modify LOB data.
- Unless otherwise stated, the default value for an `offset` parameter is 1, which indicates the first byte in the `BLOB` or `BFILE` data, and the first character in the `CLOB` or `NCLOB` value. No default values are specified for the `amount` parameter — you must input the values explicitly.
- You must lock the row containing the destination internal LOB before calling any subprograms that modify the LOB, such as `APPEND`, `COPY`, `ERASE`, `TRIM`, or `WRITE`. These subprograms do not implicitly lock the row containing the LOB.

Rules and Limits Specific to External Files (BFILEs)

- The subprograms `COMPARE`, `INSTR`, `READ`, `SUBSTR`, `FILECLOSE`, `FILECLOSEALL` and `LOADFROMFILE` operate only on an *opened* `BFILE` locator; that is, a successful `FILEOPEN` call must precede a call to any of these subprograms.
- For the functions `FILEEXISTS`, `FILEGETNAME` and `GETLENGTH`, a file's open/close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the `DIRECTORY` object and the file.
- `DBMS_LOB` does not support any concurrency control mechanism for `BFILE` operations.
- In the event of several open files in the session whose closure has not been handled properly, you can use the `FILECLOSEALL` subprogram to close all files opened in the session and resume file operations from the beginning.
- If you are the creator of a `DIRECTORY`, or if you have system privileges, then use the `CREATE OR REPLACE`, `DROP`, and `REVOKE` statements in SQL with extreme caution.

  If you, or other grantees of a particular directory object, have several open files in a session, then any of the preceding commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to invoke a program or anonymous block that calls `FILECLOSEALL`, reopen your files, and restart your file operations.
- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after *both* normal and abnormal termination of operations on the `BFILE`.

  In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than `SESSION_MAX_OPEN_FILES`.

  In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an exception occurs, only the exception handler has access to the `BFILE` variable in its most current state.

  After the exception transfers program control outside the PL/SQL program block, all references to the open `BFILEs` are lost. The result is a larger open file count which may or may not exceed the `SESSION_MAX_OPEN_FILES` value.

  For example, consider a `READ` operation past the end of the `BFILE` value, which generates a `NO_DATA_FOUND` exception:

```
-- This assumes a directory 'DDD' whose path is already known
DECLARE
       fil BFILE:= bfilename('DDD', 'filename.foo');
       pos INTEGER;
       amt BINARY_INTEGER;
       buf RAW(40);
BEGIN
       SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
       dbms_lob.open(fil, dbms_lob.lob_readonly);
       amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
       dbms_lob.read(fil, amt, pos, buf);
       dbms_output.put_line('Read F1 past EOF: '||
           utl_raw.cast_to_varchar2(buf));
       dbms_lob.close(fil);
END;

ORA-01403: no data found
```

```
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10
```

After the exception has occurred, the `BFILE` locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```
DECLARE
     fil BFILE;
     pos INTEGER;
     amt BINARY_INTEGER;
     buf RAW(40);
BEGIN
     SELECT ad_graphic INTO fil FROM print_media WHERE product_id = 3106;
     dbms_lob.open(fil, dbms_lob.lob_readonly);
     amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
     dbms_lob.read(fil, amt, pos, buf);
     dbms_output.put_line('Read F1 past EOF: '||
          utl_raw.cast_to_varchar2(buf));
     dbms_lob.close(fil);
     exception
     WHEN no_data_found
     THEN
       BEGIN
         dbms_output.put_line('End of File reached. Closing file');
         dbms_lob.fileclose(fil);
         -- or dbms_lob.filecloseall if appropriate
       END;
END;
     /

Statement processed.
End of File reached. Closing file
```

In general, you should ensure that files opened in a PL/SQL block using `DBMS_LOB` are closed before normal or abnormal termination of the block.

Maximum LOB Size

The maximum size for LOBs supported by the database is equal to the value of the blocksize of the tablespace the LOB column resides in times the value 2^{32}-1 (4294967295). This allows for a maximum LOB size ranging from 8 terabytes to 128 terabytes.

Maximum Buffer Size

The maximum buffer size, 32767 bytes.

For `BLOBs`, where buffer size is expressed in bytes, the number of bytes cannot exceed 32767.

For `CLOBs` or `NCLOBs`, where buffer size is expressed in characters, the number of characters cannot result in a buffer larger than 32767 bytes. For example, if you are using fixed-width, two-byte characters, then specifying 20000 characters is an error (20000*2 = 40000, which is greater than 32767).

DBMS_LOB Exceptions

The table in the topic describes the exceptions for DBMS_LOB.

Table 120-8 DBMS_LOB Exceptions

Exception	Code	Description
`ACCESS_ERROR`	`22925`	You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.
`CONTENTTYPE_TOOLONG`	`43859`	The length of the `contenttype` string exceeds the defined maximum. Modify the length of the `contenttype` string and retry the operation.
`CONTENTTYPEBUF_WRONG`	`43862`	The length of the `contenttype` buffer is less than defined constant. Modify the length of the `contenttype` buffer and retry the operation.
`INVALID_ARGVAL`	`21560`	The argument is expecting a non-`NULL`, valid value but the argument value passed in is `NULL`, invalid, or out of range.
`INVALID_DIRECTORY`	`22287`	The directory used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.
`NO_DATA_FOUND`	`1403`	`ENDOFLOB` indicator for looping read operations. This is not a hard error.
`NOEXIST_DIRECTORY`	`22285`	The directory leading to the file does not exist.
`NOPRIV_DIRECTORY`	`22286`	The user does not have the necessary access privileges on the directory or the file for the operation.
`OPEN_TOOMANY`	`22290`	The number of open files has reached the maximum limit.
`OPERATION_FAILED`	`22288`	The operation attempted on the file failed.
`QUERY_WRITE`	`14553`	Cannot perform a LOB write inside a query or PDML parallel execution server
`SECUREFILE_BADLOB`	`43856`	A non-`SECUREFILE` LOB type was used in a `SECUREFILE` only call
`SECUREFILE_BADPARAM`	`43857`	An invalid argument was passed to a `SECUREFILE` subprogram
`SECUREFILE_MARKERASED`	`43861`	The mark provided to a `FRAGMENT_*` operation has been deleted
`SECUREFILE_OUTOFBOUNDS`	`43883`	Attempted to perform a `FRAGMENT_*` operation past the LOB end
`UNOPENED_FILE`	`22289`	The file is not open for the required operation to be performed.
`VALUE_ERROR`	`6502`	PL/SQL error for invalid values to subprogram's parameters.

Summary of DBMS_LOB Subprograms

This table lists the `DBMS_LOB` subprograms and briefly describes them.

Table 120-9 DBMS_LOB Package Subprograms

Subprogram	Description
APPEND Procedures	Appends the contents of the source LOB to the destination LOB
CLOB2FILE Procedure	Writes the content of a `CLOB` into a file.
CLOSE Procedure	Closes a previously opened internal or external LOB
COMPARE Functions	Compares two entire `LOBs` or parts of two `LOBs`

Table 120-9 (Cont.) DBMS_LOB Package Subprograms

Subprogram	Description
CONVERTTOBLOB Procedure	Reads character data from a source `CLOB` or `NCLOB` instance, converts the character data to the specified character, writes the converted data to a destination `BLOB` instance in binary format, and returns the new offsets
CONVERTTOCLOB Procedure	Takes a source `BLOB` instance, converts the binary data in the source instance to character data using the specified character, writes the character data to a destination `CLOB` or `NCLOB` instance, and returns the new offsets
COPY Procedures	Copies all, or part, of the source LOB to the destination LOB
COPY_DBFS_LINK Procedures	Copies the DBFS Link in the source LOB to the destination LOB
COPY_FROM_DBFS_LINK	Retrieves the data for the LOB from the DBFS store
CREATETEMPORARY Procedures	Creates a temporary `BLOB` or `CLOB` and its corresponding index in the user's default temporary tablespace
DBFS_LINK_GENERATE_PATH Functions	Returns a unique file path name for use in creating a DBFS Link
ERASE Procedures	Erases all or part of a LOB
FILECLOSE Procedure	Closes the file
FILECLOSEALL Procedure	Closes all previously opened files
FILEEXISTS Function	Checks if the file exists on the server
FILEGETNAME Procedure	Gets the directory object name and file name
FILEISOPEN Function	Checks if the file was opened using the input `BFILE` locators
FILEOPEN Procedure	Opens a file
FRAGMENT_DELETE Procedure	Deletes the data at the specified offset for the specified length from the LOB
FRAGMENT_INSERT Procedures	Inserts the specified data (limited to 32K) into the LOB at the specified offset
FRAGMENT_MOVE Procedure	Moves the amount of bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) from the specified offset to the new offset specified
FRAGMENT_REPLACE Procedures	Replaces the data at the specified offset with the specified data (not to exceed 32k)
FREETEMPORARY Procedures	Frees the temporary `BLOB` or `CLOB` in the default temporary tablespace
GET_DBFS_LINK Functions	Returns the DBFS Link path associated with the specified SecureFile
GET_DBFS_LINK_STATE Procedures	Retrieves the current DBFS Link state of the specified SecureFile
GET_LOB_DEDUPLICATION_RATIO Function	Returns the deduplication ratio, which indicates that amount of space you can save by enabling deduplication.
GETCHUNKSIZE Functions	Returns the amount of space used in the LOB chunk to store the LOB value
GETCONTENTTYPE Functions	Returns the content ID string previously set by means of the SETCONTENTTYPE Procedure
GETLENGTH Functions	Gets the length of the LOB value

Table 120-9 (Cont.) DBMS_LOB Package Subprograms

Subprogram	Description
GETOPTIONS Functions	Obtains settings corresponding to the `option_type` field for a particular LOB
GET_STORAGE_LIMIT Function	Returns the storage limit for LOBs in your database configuration
INSTR Functions	Returns the matching position of the *nth* occurrence of the pattern in the LOB
ISOPEN Functions	Checks to see if the LOB was already opened using the input locator
ISREMOTE Function	Checks to see if the LOB is local to the database or if it belongs to a remote database.
ISSECUREFILE Function	Returns `TRUE` if the LOB locator passed to is for a SecureFiles LOB, otherwise, returns `FALSE`
ISTEMPORARY Functions	Checks if the locator is pointing to a temporary LOB
LOADBLOBFROMFILE Procedure	Loads `BFILE` data into an internal `BLOB`
LOADCLOBFROMFILE Procedure	Loads `BFILE` data into an internal `CLOB`
MOVE_TO_DBFS_LINK Procedures	Writes the specified SecureFile data to the DBFS store
OPEN Procedures	Opens a LOB (internal, external, or temporary) in the indicated mode
READ Procedures	Reads data from the LOB starting at the specified offset
SET_DBFS_LINK Procedures	Links the specified SecureFile to the specified path name. It does not copy the data to the path
SETCONTENTTYPE Procedure	Sets the content type string for the data in the LOB
SETOPTIONS Procedures	Enables CSCE features on a per-LOB basis, overriding the default LOB column settings
SUBSTR Functions	Returns part of the LOB value starting at the specified offset
TRIM Procedures	Trims the LOB value to the specified shorter length
WRITE Procedures	Writes data to the LOB from a specified offset
WRITEAPPEND Procedures	Writes a buffer to the end of a LOB

APPEND Procedures

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

Syntax

```
DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY BLOB,
   src_lob  IN             BLOB);

DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY CLOB  CHARACTER SET ANY_CS,
   src_lob  IN             CLOB  CHARACTER SET dest_lob%CHARSET);
```

Parameters

Table 120-10 APPEND Procedure Parameters

Parameter	Description
`dest_lob`	Locator for the internal LOB to which the data is to be appended.
`src_lob`	Locator for the internal LOB from which the data is to be read.

Exceptions

Table 120-11 APPEND Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Either the source or the destination LOB is `NULL`.
`QUERY_WRITE`	Cannot perform a LOB write inside a query or PDML parallel execution server

Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

  If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.
- If `APPEND` is called on a LOB that has been archived, it implicitly gets the LOB before the first byte is written
- If `APPEND` is called on a SecureFiles LOB that is a DBFS Link, an exception is thrown.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

CLOB2FILE Procedure

This procedure writes the content of a `CLOB` into a `bfile`. This procedure gets called from the deprecated `dbms_xslprocessor.clob2file` internally.

Syntax

```
DBMS_LOB.CLOB2FILE(
   src_cl       IN   CLOB,
   file_loc    IN  VARCHAR2,
   file_name   IN  VARCHAR2,
   csid        IN  NUMBER   := 0,
   open_mode   IN  VARCHAR2 :='wb');
```

Parameters

Table 120-12 CLOB2FILE Procedure Parameters

Parameter	Description
`src_cl`	Source CLOB locator to write into a file
`file_loc`	Directory object name where the file is located
`file_name`	File name
`csid`	Character set id of the CLOB locator • Must be a valid Oracle id; otherwise returns an error • If the value is `0`, then the content of the output file will be in the database character set
`open_mode`	The mode to open the output file in. • `wb` — write byte mode, overwrites the file The default value is `wb`.

CLOSE Procedure

This procedure closes a previously opened internal or external LOB.

Syntax

```
DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY BLOB);

DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);

DBMS_LOB.CLOSE (
   file_loc   IN OUT NOCOPY BFILE);
```

Parameters

Table 120-13 CLOSE Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.

Exceptions

No error is returned if the `BFILE` exists but is not opened. An error is returned if the LOB is not open.

Usage Notes

`CLOSE` requires a round-trip to the server for both internal and external LOBs. For internal LOBs, `CLOSE` triggers other code that relies on the close call, and for external LOBs (`BFILEs`), `CLOSE` actually closes the server-side operating system file.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

COMPARE Functions

This function compares two entire LOBs or parts of two LOBs.

Syntax

```
DBMS_LOB.COMPARE (
   lob_1            IN BLOB,
   lob_2            IN BLOB,
   amount           IN INTEGER := DBMS_LOB.LOBMAXSIZE,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN CLOB   CHARACTER SET ANY_CS,
   lob_2            IN CLOB   CHARACTER SET lob_1%CHARSET,
   amount           IN INTEGER := DBMS_LOB.LOBMAXSIZE,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN BFILE,
   lob_2            IN BFILE,
   amount           IN INTEGER,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 120-14 COMPARE Function Parameters

Parameter	Description
`lob_1`	LOB locator of first target for comparison.
`lob_2`	LOB locator of second target for comparison.
`amount`	Number of bytes (for `BLOBs`) or characters (for `CLOBs`/`NCLOBSs`) to compare.
`offset_1`	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.
`offset_2`	Offset in bytes or characters on the second LOB (origin: 1) for the comparison.

Return Values

- `INTEGER`: 0 if the comparison succeeds, nonzero if not.
- `NULL`, if any of `amount, offset_1` or `offset_2` is not a valid LOB offset value. A valid offset is within the range of 1 to `LOBMAXSIZE` inclusive.

Usage Notes

- You can only compare LOBs of the same datatype (`LOBs` of `BLOB` type with other `BLOBs`, and `CLOBs` with `CLOBs`, and `BFILEs` with `BFILEs`). For `BFILEs`, the file must be already opened using a successful `FILEOPEN` operation for this operation to succeed.
- `COMPARE` returns 0 if the data exactly matches over the range specified by the `offset` and `amount` parameters. `COMPARE` returns -1 if the first `CLOB` is less than the second, and 1 if it is greater.
- For fixed-width *n*-byte `CLOBs`, if the input amount for `COMPARE` is specified to be greater than (`DBMS_LOB.LOBMAXSIZE`/*n*), then `COMPARE` matches characters in a range of size (`DBMS_LOB.LOBMAXSIZE`/*n*), or Max(length(clob1), length(clob2)), whichever is lesser.
- If `COMPARE` is called on any LOB that has been archived, it implicitly gets the LOB before the compare begins.
- If `COMPARE()` is called on a SecureFiles LOB that is a DBFS Link, the linked LOB is streamed from DBFS, if possible, otherwise an exception is thrown.

Exceptions

Table 120-15 COMPARE Function Exceptions for BFILE operations

Exception	Description
`UNOPENED_FILE`	File was not opened using the input locator.
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.
`INVALID_OPERATION`	File does not exist, or you do not have access privileges on the file.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

CONVERTTOBLOB Procedure

This procedure reads character data from a source `CLOB` or `NCLOB` instance, converts the character data to the character set you specify, writes the converted data to a destination `BLOB` instance in binary format, and returns the new offsets.

You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

Syntax

```
DBMS_LOB.CONVERTTOBLOB(
  dest_lob       IN OUT     NOCOPY  BLOB,
  src_clob       IN         CLOB CHARACTER SET ANY_CS,
  amount         IN         INTEGER,
  dest_offset    IN OUT     INTEGER,
  src_offset     IN OUT     INTEGER,
  blob_csid      IN         NUMBER,
  lang_context   IN OUT     INTEGER,
  warning        OUT        INTEGER);
```

Parameters

Table 120-16 CONVERTTOBLOB Procedure Parameters

Parameter	Description
`dest_lob`	LOB locator of the destination LOB instance.
`src_clob`	LOB locator of the source LOB instance.
`amount`	Number of characters to convert from the source LOB. If you want to copy the entire LOB, pass the constant `DBMS_LOB.LOBMAXSIZE`. If you pass any other value, it must be less than or equal to the size of the LOB.
`dest_offset`	`(IN)` Offset in bytes in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. `(OUT)` The new offset in bytes after the end of the write.
`src_offset`	`(IN)` Offset in characters in the source LOB for the start of the read. `(OUT)` Offset in characters in the source LOB right after the end of the read.
`blob_csid`	Desired character set ID of the converted data.
`lang_context`	`(IN)` Language context, such as shift status, for the current conversion. `(OUT)` The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.

Table 120-16 (Cont.) CONVERTTOBLOB Procedure Parameters

Parameter	Description
`warning`	`(OUT)` Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant `warn_inconvertible_char` in the `DBMS_LOB` package.

Usage Notes

Preconditions

Before calling the `CONVERTTOBLOB` procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked. To lock the row, select the LOB using the `FOR UPDATE` clause of the `SELECT` statement.

Constants and Defaults

All parameters are required. You must pass a variable for each `OUT` or `IN OUT` parameter. You must pass either a variable or a value for each `IN` parameter.

Table 120-17 gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the `dbmslob.sql` package specification file.

Table 120-17 DBMS_LOB.CONVERTTOBLOB Typical Values

Parameter	Value	Description
`amount`	`LOBMAXSIZE (IN)`	convert the entire file
`dest_offset`	`1 (IN)`	start from the beginning
`src_offset`	`1 (IN)`	start from the beginning
`blob_csid`	`DEFAULT_CSID (IN)`	default `CSID`, use same `CSID` as source LOB
`lang_context`	`DEFAULT_LANG_CTX (IN)`	default language context
`warning`	`NO_WARNING (OUT)`	no warning message, success
	`WARN_INCONVERTIBLE_CHAR (OUT)`	character in source cannot be properly converted

General Notes

- You must specify the desired character set for the destination LOB in the `blob_csid` parameter. You can pass a zero value for `blob_csid`. When you do so, the database assumes that the desired character set is the same as the source LOB character set.

- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The `amount` and `src_offset` values are in characters and the `dest_offset` is in bytes. To convert the entire LOB, you can specify `LOBMAXSIZE` for the `amount` parameter.
- `CONVERTTOBLOB` gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

Exceptions

Table 120-18 gives possible exceptions this procedure can throw. The first column lists the exception string and the second column describes the error conditions that can cause the exception.

Table 120-18 CONVERTTOBLOB Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	One or more of the following: - `src_offset` or `dest_offset` < 1. - `src_offset` or `dest_offset` > `LOBMAXSIZE`. - `amount` < 1. - `amount` > `LOBMAXSIZE`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for more information on using LOBs in application development

CONVERTTOCLOB Procedure

This procedure takes a source `BLOB` instance, converts the binary data in the source instance to character data using the character set you specify, writes the character data to a destination `CLOB` or `NCLOB` instance, and returns the new offsets.

You can use this interface with any combination of persistent or temporary LOB instances as the source or destination.

Syntax

```
DBMS_LOB.CONVERTTOCLOB(
   dest_lob       IN OUT NOCOPY  CLOB CHARACTER SET ANY_CS,
   src_blob       IN             BLOB,
   amount         IN             INTEGER,
   dest_offset    IN OUT         INTEGER,
   src_offset     IN OUT         INTEGER,
   blob_csid      IN             NUMBER,
   lang_context   IN OUT         INTEGER,
   warning        OUT            INTEGER);
```

Parameters

Table 120-19 CONVERTTOCLOB Procedure Parameters

Parameter	Description
`dest_lob`	LOB locator of the destination LOB instance.
`src_blob`	LOB locator of the source LOB instance.
`amount`	Number of bytes to convert from the source LOB. If you want to copy the entire BLOB, pass the constant `DBMS_LOB.LOBMAXSIZE`. If you pass any other value, it must be less than or equal to the size of the `BLOB`.
`dest_offset`	`(IN)` Offset in characters in the destination LOB for the start of the write. Specify a value of 1 to start at the beginning of the LOB. `(OUT)` The new offset in characters after the end of the write. This offset always points to the beginning of the first complete character after the end of the write.
`src_offset`	`(IN)` Offset in bytes in the source LOB for the start of the read. `(OUT)` Offset in bytes in the source LOB right after the end of the read.
`blob_csid`	The character set ID of the source data
`lang_context`	`(IN)` Language context, such as shift status, for the current conversion. `(OUT)` The language context at the time when the current conversion is done. This information is returned so you can use it for subsequent conversions without losing or misinterpreting any source data. For the very first conversion, or if do not care, use the default value of zero.
`warning`	Warning message. This parameter indicates when something abnormal happened during the conversion. You are responsible for checking the warning message. Currently, the only possible warning is — inconvertible character. This occurs when the character in the source cannot be properly converted to a character in destination. The default replacement character (for example, '?') is used in place of the inconvertible character. The return value of this error message is defined as the constant `warn_inconvertible_char` in the `DBMS_LOB` package.

Usage Notes

Preconditions

Before calling the `CONVERTTOCLOB` procedure, the following preconditions must be met:

- Both the source and destination LOB instances must exist.
- If the destination LOB is a persistent LOB, the row must be locked before calling the `CONVERTTOCLOB` procedure. To lock the row, select the LOB using the `FOR UPDATE` clause of the `SELECT` statement.

Constants and Defaults

All parameters are required. You must pass a variable for each `OUT` or `IN OUT` parameter. You must pass either a variable or a value for each `IN` parameter.

Table 120-20 gives a summary of typical values for each parameter. The first column lists the parameter, the second column lists the typical value, and the last column describes the result of passing the value. Note that constants are used for some values. These constants are defined in the `dbmslob.sql` package specification file.

Table 120-20 DBMS_LOB.CONVERTTOCLOB Typical Values

Parameter	Value	Description
`amount`	`LOBMAXSIZE (IN)`	convert the entire file
`dest_offset`	`1 (IN)`	start from the beginning
`src_offset`	`1 (IN)`	start from the beginning
`csid`	`DEFAULT_CSID (IN)`	default `CSID`, use destination `CSID`
`lang_context`	`DEFAULT_LANG_CTX (IN)`	default language context

`warning`	`NO_WARNING (OUT)`
	`WARN_INCONVERTIBLE_CHAR (OUT)`
		no warning message, success
		character in source cannot be properly converted

General Notes

- You must specify the desired character set for the source LOB in the `blob_csid` parameter. You can pass a zero value for `blob_csid`. When you do so, the database assumes that the desired character set is the same as the destianation LOB character set.
- You must specify the offsets for both the source and destination LOBs, and the number of characters to copy from the source LOB. The `amount` and `src_offset` values are in bytes and the `dest_offset` is in characters. To convert the entire LOB, you can specify `LOBMAXSIZE` for the `amount` parameter.
- `CONVERTTOCLOB` gets the source and/or destination LOBs as necessary prior to conversion and write of the data.

Exceptions

Table 120-21 CONVERTTOCLOB Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	One or more of the following: - `src_offset` or `dest_offset` < 1. - `src_offset` or `dest_offset` > `LOBMAXSIZE`. - `amount` < 1. - `amount` > `LOBMAXSIZE`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for more information on using LOBs in application development

COPY Procedures

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

Syntax

```
DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY BLOB,
  src_lob     IN            BLOB,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);

DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY CLOB  CHARACTER SET ANY_CS,
  src_lob     IN            CLOB  CHARACTER SET dest_lob%CHARSET,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);
```

Parameters

Table 120-22 COPY Procedure Parameters

Parameter	Description
`dest_lob`	LOB locator of the copy target.
`src_lob`	LOB locator of source for the copy.
`amount`	Number of bytes (for `BLOB`s) or characters (for `CLOB`s) to copy.
`dest_offset`	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.
`src_offset`	Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.

Exceptions

Table 120-23 COPY Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or invalid.
`INVALID_ARGVAL`	Either: - `src_offset` or `dest_offset` < 1 - `src_offset` or `dest_offset` > `LOBMAXSIZE` - `amount` < 1 - `amount` > `LOBMAXSIZE`
`QUERY_WRITE`	Cannot perform a LOB write inside a query or PDML parallel execution server

Usage Notes

- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination `BLOB` or `CLOB` respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the `src_offset` to the end of the source LOB.
- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.
- Prior to copy, the source and destination LOBs are retrieved, if they are currently archived. For a complete over-write, the destination LOB is not retrieved.
- If the source LOB is a DBFS Link, the data is streamed from DBFS, if possible, otherwise an exception is thrown. If the destination LOB is a DBFS Link, an exception is thrown.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

COPY_DBFS_LINK Procedures

This procedure copies the DBFS Link in the source LOB to the destination LOB.

Syntax

```
DBMS_LOB.COPY_DBFS_LINK (
  lob_loc_dst     IN OUT BLOB,
  lob_loc_src     IN     BLOB,
  flags           IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);

DBMS_LOB.COPY_DBFS_LINK (
  lob_loc_dst     IN OUT CLOB CHARACTER SET ANY_CS,
  lob_loc_src     IN     CLOB CHARACTER SET ANY_CS,
  flags           IN     PLS_INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

Parameters

Table 120-24 COPY_DBFS_LINK Procedure Parameters

Parameter	Description
`lob_loc_dst`	LOB to be made to reference the same storage data as `lob_loc_src`
`lob_loc_src`	LOB from which to copy the reference
`flags`	Options to `COPY_DBFS_LINK`: • `DBFS_LINK_NOCACHE` specifies to only copy the DBFS Link • `DBFS_LINK_CACHE` specifies to copy the DBFS Link and read the data into the database LOB specified by `lob_loc_dst` so that the data is cached

Exceptions

Table 120-25 COPY_DBFS_LINK Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	Either `lob_loc_src` or `lob_loc_dst` is not a `SECUREFILE`
`INVALID_ARGVAL`	`lob_loc_src` LOB has not been archived
`ORA-01555`	If the source LOB has been retrieved, never archived, or if the LOB has been migrated in and out (modified or not) since the locator was gotten.

COPY_FROM_DBFS_LINK

This procedure retrieves the archived SecureFiles LOB data from the DBFS HSM store and to the database.

Syntax

```
DBMS_LOB.COPY_FROM_DBFS_LINK (
  lob_loc       IN OUT BLOB);

DBMS_LOB.COPY_FROM_DBFS_LINK (
  lob_loc       IN OUT CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 120-26 COPY_FROM_DBFS_LINK Procedure Parameters

Parameter	Description
`lob_loc`	LOB to be retrieved from the archive

Usage Note

`COPY_FROM_DBFS_LINK` does not remove the underlying DBFS file.

If the LOB is successfully retrieved, `COPY_FROM_DBFS_LINK` silently returns success.

Exceptions

Table 120-27 COPY_FROM_DBFS_LINK Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`
`ORA-01555`	If the LOB has already been retrieved and has been modified since retrieval, if the LOB has been migrated in and out (modified or not) since the locator was retrieved

CREATETEMPORARY Procedures

This procedure creates a temporary `BLOB` or `CLOB` and its corresponding index in your default temporary tablespace.

Syntax

```
DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY BLOB,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := DBMS_LOB.SESSION);

DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);
```

Parameters

Table 120-28 CREATETEMPORARY Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.
`cache`	Specifies if LOB should be read into buffer cache or not.
`dur`	1 of 2 predefined duration values (`SESSION` or `CALL`) which specifies a hint as to whether the temporary LOB is cleaned up at the end of the session or call. If `dur` is omitted, then the session duration is used.

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure
- *Oracle Database PL/SQL Language Reference* for more information about NOCOPY and passing temporary lobs as parameters

DBFS_LINK_GENERATE_PATH Functions

This subprogram returns a unique file path name for use in creating a DBFS Link.

Syntax

```
DBMS_LOB.DBFS_LINK_GENERATE_PATH (
  lob_loc       IN BLOB,
  storage_dir   IN VARCHAR2)
 RETURN VARCHAR2;

DBMS_LOB.DBFS_LINK_GENERATE_PATH (
  lob_loc       IN CLOB CHARACTER SET ANY_CS,
  storage_dir   IN VARCHAR2)
 RETURN VARCHAR2;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(dbfs_link_generate_path,
       WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-29   DBFS_LINK_GENERATE_PATH Function Parameters

Parameter	Description
`lob_loc`	LOB to be retrieved from DBFS
`storage_dir`	DBFS directory that will be the parent directory of the file

Exceptions

Table 120-30   DBFS_LINK_GENERATE_PATH Function Exceptions

Exception	Description
`SECUREFILE_WRONGTYPE`	`lob_loc` is not a `SECUREFILE`

Usage Notes

Returns a globally unique file pathname that can be used for archiving. This is guaranteed to be globally unique across all calls to this function for different LOBs and versions of that LOB. It is always the same for the same LOB and version.

ERASE Procedures

This procedure erases an entire internal LOB or part of an internal LOB.

Syntax

```
DBMS_LOB.ERASE (
   lob_loc           IN OUT   NOCOPY   BLOB,
   amount            IN OUT   NOCOPY   INTEGER,
   offset            IN                INTEGER := 1);

DBMS_LOB.ERASE (
   lob_loc           IN OUT   NOCOPY   CLOB CHARACTER SET ANY_CS,
   amount            IN OUT   NOCOPY   INTEGER,
   offset            IN                INTEGER := 1);
```

Parameters

Table 120-31   ERASE Procedure Parameters

Parameter	Description
`lob_loc`	Locator for the LOB to be erased.For more information, see Operational Notes.
`amount`	Number of bytes (for `BLOB`s or `BFILES`) or characters (for `CLOB`s or `NCLOB`s) to be erased.
`offset`	Absolute offset (origin: 1) from the beginning of the LOB in bytes (for `BLOB`s) or characters (`CLOB`s).

Usage Notes

- When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for `BLOB`s or `CLOB`s respectively.
- The actual number of bytes or characters erased can differ from the number you specified in the `amount` parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the `amount` parameter.
- `ERASE` gets the LOB if it is archived, unless the erase covers the entire LOB.
- If the LOB to be erased is a DBFS Link, an exception is thrown.

Note:

The length of the LOB is not decreased when a section of the LOB is erased. To decrease the length of the LOB value, see the "TRIM Procedures".

Exceptions

Table 120-32   ERASE Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any input parameter is `NULL`.
`INVALID_ARGVAL`	Either: - `amount` < 1 or `amount` > `LOBMAXSIZE` - `offset` < 1 or `offset` > `LOBMAXSIZE`
`QUERY_WRITE`	Cannot perform a LOB write inside a query or PDML parallel execution server

Usage Notes

It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.

See Also:

- TRIM Procedures
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FILECLOSE Procedure

This procedure closes a `BFILE` that has already been opened through the input locator.

Note:

The database has only read-only access to `BFILE`s. This means that `BFILE`s cannot be written through the database.

Syntax

```
DBMS_LOB.FILECLOSE (
    file_loc IN OUT NOCOPY BFILE);
```

Parameters

Table 120-33 FILECLOSE Procedure Parameters

Parameter	Description
`file_loc`	Locator for the `BFILE` to be closed.

Exceptions

Table 120-34 FILECLOSE Procedure Exceptions

Exception	Description
`VALUE_ERROR`	`NULL` input value for `file_loc`.
`UNOPENED_FILE`	File was not opened with the input locator.
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.

Table 120-34 (Cont.) FILECLOSE Procedure Exceptions

Exception	Description
`INVALID_OPERATION`	File does not exist, or you do not have access privileges on the file.

See Also:

- "FILEOPEN Procedure"
- "FILECLOSEALL Procedure"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FILECLOSEALL Procedure

This procedure closes all `BFILEs` opened in the session.

Syntax

```
DBMS_LOB.FILECLOSEALL;
```

Exceptions

Table 120-35 FILECLOSEALL Procedure Exception

Exception	Description
`UNOPENED_FILE`	No file has been opened in the session.

See Also:

- "FILEOPEN Procedure"
- "FILECLOSE Procedure"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FILEEXISTS Function

This function finds out if a specified `BFILE` locator points to a file that actually exists on the server's file system.

Syntax

```
DBMS_LOB.FILEEXISTS (
   file_loc     IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-36 FILEEXISTS Function Parameter

Parameter	Description
`file_loc`	Locator for the `BFILE`.

Return Values

Table 120-37 FILEEXISTS Function Return Values

Return	Description
`0`	Physical file does not exist.
`1`	Physical file exists.

Exceptions

Table 120-38 FILEEXISTS Function Exceptions

Exception	Description
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.

See Also:

- "FILEISOPEN Function".
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FILEGETNAME Procedure

This procedure determines the directory object and filename, given a `BFILE` locator.

This function only indicates the directory object name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the `dir_alias` buffer is 30, and for the entire path name, it is 2000.

Syntax

```
DBMS_LOB.FILEGETNAME (
   file_loc    IN     BFILE,
   dir_alias   OUT    VARCHAR2,
   filename    OUT    VARCHAR2);
```

Parameters

Table 120-39 FILEGETNAME Procedure Parameters

Parameter	Description
`file_loc`	Locator for the `BFILE`
`dir_alias`	Directory object name
`filename`	Name of the `BFILE`

Exceptions

Table 120-40 FILEGETNAME Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	`dir_alias` or filename are `NULL`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

FILEISOPEN Function

This function finds out whether a `BFILE` was opened with the specified `FILE` locator.

Syntax

```
DBMS_LOB.FILEISOPEN (
   file_loc    IN     BFILE)
  RETURN  INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(fileisopen, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-41 FILEISOPEN Function Parameter

Parameter	Description
`file_loc`	Locator for the `BFILE`.

Return Values

`INTEGER`: 0 = file is not open, 1 = file is open

Usage Notes

If the input `FILE` locator was never passed to the `FILEOPEN` procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

Exceptions

Table 120-42 FILEISOPEN Function Exceptions

Exception	Description
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.

See Also:

- "FILEEXISTS Function"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FILEOPEN Procedure

This procedure opens a `BFILE` for read-only access. `BFILE` data may not be written through the database.

Syntax

```
DBMS_LOB.FILEOPEN (
   file_loc   IN OUT NOCOPY  BFILE,
   open_mode  IN             BINARY_INTEGER := file_readonly);
```

Parameters

Table 120-43 FILEOPEN Procedure Parameters

Parameter	Description
`file_loc`	Locator for the `BFILE`.
`open_mode`	File access is read-only.

Exceptions

Table 120-44 FILEOPEN Procedure Exceptions

Exception	Description
`VALUE_ERROR`	`file_loc` or `open_mode` is `NULL`.
`INVALID_ARGVAL`	`open_mode` is not equal to `FILE_READONLY`.
`OPEN_TOOMANY`	Number of open files in the session exceeds `session_max_open_files`.
`NOEXIST_DIRECTORY`	Directory associated with `file_loc` does not exist.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.
`INVALID_OPERATION`	File does not exist, or you do not have access privileges on the file.

See Also:

- "FILECLOSE Procedure"
- "FILECLOSEALL Procedure"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

FRAGMENT_DELETE Procedure

This procedure deletes the data at the specified offset for the specified length from the LOB without having to rewrite all the data in the LOB following the specified offset.

Syntax

```
DBMS_LOB.FRAGMENT_DELETE (
   lob_loc      IN OUT NOCOPY BLOB,
   amount       IN            INTEGER,
   offset       IN            INTEGER);

DBMS_LOB.FRAGMENT_DELETE (
   lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount       IN            INTEGER,
   offset       IN            INTEGER);
```

Parameters

Table 120-45 FRAGMENT_DELETE Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.
`amount`	Number of bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) to be removed from the LOB
`offset`	Offset into the LOB in bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) to begin the deletion

Exceptions

Table 120-46 FRAGMENT_DELETE Procedure Exceptions

Exception	Description
`INVALID_ARGVAL`	A parameter value was invalid
`QUERY_WRITE`	Cannot perform operation during a query
`SECUREFILE_BADLOB`	A non-`SECUREFILE` LOB was used in a `SECUREFILE` LOB only call
`SECUREFILE_OUTOFBOUNDS`	Attempted to perform a `FRAGMENT_*` operation past LOB end

FRAGMENT_INSERT Procedures

This procedure inserts the specified data (limited to 32K) into the LOB at the specified offset.

Syntax

```
DBMS_LOB.FRAGMENT_INSERT (
   lob_loc      IN OUT NOCOPY BLOB,
   amount       IN            INTEGER,
   offset       IN            INTEGER,
   buffer       IN            RAW);

DBMS_LOB.FRAGMENT_INSERT (
   lob_loc      IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount       IN            INTEGER,
   offset       IN            INTEGER,
   buffer       IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 120-47 FRAGMENT_INSERT Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator.For more information, see Operational Notes.
`amount`	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be inserted into the LOB
`offset`	Offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to begin the insertion

Table 120-47 (Cont.) FRAGMENT_INSERT Procedure Parameters

Parameter	Description
`buffer`	Data to insert into the LOB

Exceptions

Table 120-48 FRAGMENT_INSERT Procedure Exceptions

Exception	Description
`INVALID_ARGVAL`	A parameter value was invalid
`QUERY_WRITE`	Cannot perform operation during a query
`SECUREFILE_BADLOB`	A non-`SECUREFILE` LOB was used in a `SECUREFILE` LOB only call
`SECUREFILE_OUTOFBOUNDS`	Attempted to perform a `FRAGMENT_*` operation past LOB end

Usage Notes

`FRAGMENT_INSERT` gets the LOB, if necessary, before performing operations on the LOB.

FRAGMENT_MOVE Procedure

This procedure moves the amount of bytes (BLOB) or characters (CLOB/NCLOB) from the specified offset to the new offset specified.

Syntax

```
DBMS_LOB.FRAGMENT_MOVE (
   lob_loc       IN OUT NOCOPY BLOB,
   amount        IN            INTEGER,
   src_offset    IN            INTEGER,
   dest_offset   IN            INTEGER);

DBMS_LOB.FRAGMENT_MOVE (
   lob_loc       IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount        IN            INTEGER,
   src_offset    IN            INTEGER,
   dest_offset   IN            INTEGER);
```

Parameters

Table 120-49 FRAGMENT_MOVE Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.
`amount`	Number of bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) to be moved in the LOB
`src_offset`	Beginning offset into the LOB in bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) to put the data
`dest_offset`	Beginning offset into the LOB in bytes (`BLOB`) or characters (`CLOB`/`NCLOB`) to remove the data

Exceptions

Table 120-50 FRAGMENT_MOVE Procedure Exceptions

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

Usage Notes

- All offsets are pre-move offsets.
- Offsets of more than 1 past the end of the LOB are not permitted.
- FRAGMENT_MOVE gets the LOB, if necessary, before performing operations on the LOB.

FRAGMENT_REPLACE Procedures

This procedure replaces the data at the specified offset with the specified data (not to exceed 32k).

Syntax

```
DBMS_LOB.FRAGMENT_REPLACE (
   lob_loc     IN OUT NOCOPY BLOB,
   old_amount  IN            INTEGER,
   new_amount  IN            INTEGER,
   offset      IN            INTEGER,
   buffer      IN            RAW);

DBMS_LOB.FRAGMENT_REPLACE (
   lob_loc     IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,   old_amount  IN
INTEGER,
   new_amount  IN           INTEGER,
   offset      IN           INTEGER,
   buffer      IN           VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 120-51 FRAGMENT_REPLACE Function Parameters

Parameter	Description
lob_loc	LOB locator. For more information, see Operational Notes.
old_amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to be replaced in the LOB
new_amount	Number of bytes (BLOB) or characters (CLOB/NCLOB) to written to the LOB
offset	Beginning offset into the LOB in bytes (BLOB) or characters (CLOB/NCLOB) to put the data
buffer	Data to insert into the LOB

Exceptions

Table 120-52 FRAGMENT_REPLACE Procedure Exceptions

Exception	Description
INVALID_ARGVAL	A parameter value was invalid
QUERY_WRITE	Cannot perform operation during a query
SECUREFILE_BADLOB	A non-SECUREFILE LOB was used in a SECUREFILE LOB only call
SECUREFILE_OUTOFBOUNDS	Attempted to perform a FRAGMENT_* operation past LOB end

Usage Notes

- Invoking this procedure is equivalent to deleting the old amount of bytes/characters at offset and then inserting the new amount of bytes/characters at offset.
- FRAGMENT_REPLACE gets the LOB, if necessary, before performing operations on the LOB.

FREETEMPORARY Procedures

This procedure frees the temporary BLOB or CLOB in the default temporary tablespace.

Syntax

```
DBMS_LOB.FREETEMPORARY (
   lob_loc   IN OUT  NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (
   lob_loc   IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS);
```

Parameters

Table 120-53 FREETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.For more information, see Operational Notes.

Usage Notes

- When a new temporary LOB is created, and there is currently no temporary LOB in use with the same duration (session, call), a new temporary LOB segment is created. When the temporary LOB is freed, the space it consumed is released to the temporary segment. If there are no other temporary LOBs for the same duration, the temporary segment is also freed.
- After the call to FREETEMPORARY, the LOB locator that was freed is marked as invalid.
- If an invalid LOB locator is assigned to another LOB locator using OCILobLocatorAssign in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

GET_DBFS_LINK Functions

This function returns the DBFS path name for the specified SecureFile LOB.

Syntax

```
DBMS_LOB.GET_DBFS_LINK (
  lob_loc              IN     BLOB,
  storage_path         OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  lob_length           OUT NUMBER);

DBMS_LOB.GET_DBFS_LINK (
  lob_loc              IN     CLOB CHARACTER SET ANY_CS,
  storage_path         OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  lob_length           OUT NUMBER);
```

Parameters

Table 120-54 GET_DBFS_LINK Function Parameters

Parameter	Description
`lob_loc`	LOB to be retrieved from DBFS
`storage_path`	Path where the LOB is stored in DBFS
`lob_length`	LOB length at the time of write to DBFS

Return Values

The Archive ID

Exceptions

Table 120-55 GET_DBFS_LINK Function Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`
`ORA-01555`	The LOB has already been retrieved and has been modified since retrieval or the LOB has been migrated in and out (modified or not) since the locator was retrieved

GET_DBFS_LINK_STATE Procedures

`GET_DBFS_LINK_STATE` retrieves the current link state of the specified SecureFile.

Syntax

```
DBMS_LOB.GET_DBFS_LINK_STATE (
  lob_loc       IN BLOB,
  storage_path  OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  state         OUT NUMBER,
  cached        OUT BOOLEAN);

DBMS_LOB.GET_DBFS_LINK_STATE (
  lob_loc       IN CLOB CHARACTER SET ANY_CS,
  storage_path  OUT VARCHAR2(DBFS_LINK_PATH_MAX_SIZE),
  state         OUT NUMBER,
  cached        OUT BOOLEAN);
```

Parameters

Table 120-56 GET_DBFS_LINK_STATE Procedure Parameters

Parameter	Description
`lob_loc`	LOB to be retrieved from the archive
`storage_path`	Path where the LOB is stored in the DBFS HSM store
`state`	One of `DBFS_LINK_NEVER`, `DBFS_LINK_NO` or `DBFS_LINK_YES`
`cached`	If the LOB is archived and the data was specified to be cashed on put

Exceptions

Table 120-57 GET_DBFS_LINK_STATE Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`

Usage Notes

- If the LOB has never been archived, state is set to `DBMS_LOB.DBFS_LINK_NEVER`. If the LOB has been archived, state is set to `DBMS_LOB.DBFS_LINK_YES`. If the LOB has been previously retrieved from the archive, state is set to `DBFS_LINK_NO`.
- If the LOB was archived, but the data was left in the RDBMS, `cached` is set to `TRUE`. If the data was removed after the link was created, `cached` is set to `FALSE`, and `NULL` if state is `DBMS_LOB.DBFS_LINK_NEVER`.

GET_LOB_DEDUPLICATION_RATIO Function

The `GET_LOB_DEDUPLICATION_RATIO` function estimates the storage space that you can save by enabling the deduplication feature for an existing SecureFile LOB and returns the deduplication ratio.

The deduplication ratio is estimated for the number of rows in the LOB column that you specify. For example, let's consider that the deduplication ratio is 2.33. It indicates that after you enable the deduplication feature, you can save around half of the space for the sampled rows in the LOB column.

Disclaimer: The deduplication ratio is an approximate value, which is calculated based on the sampled rows in the LOB column. The actual space that you save when you enable deduplication for the complete table may be different.

Syntax

```
DBMS_LOB.GET_LOB_DEDUPLICATION_RATIO (
   tablespacename        IN     VARCHAR2,
   tabowner              IN     VARCHAR2,
   tabname               IN     VARCHAR2,
   lobcolumnname         IN     VARCHAR2,
   partname              IN     VARCHAR2,
   dedup_ratio           OUT    NUMBER,
   subset_numrows        IN     NUMBER DEFAULT DEDUP_RATIO_LOB_MAXROWS
)
```

Parameters

Table 120-58 GET_DBFS_LINK_STATE Procedure Parameters

Parameter	Description
`tablespacename`	Name of the tablespace
`tabowner`	Owner of the table
`tabname`	Name of the table
`lobcolumnname`	Name of the LOB column for which you want to calculate the deduplication ratio
`partname`	In case of partitioned tables, enter the related partition name
`subset_numrows`	Number of rows sampled to estimate the deduplication ratio. By default, the deduplication ratio is calculated for all the rows.

Return Values

Returns the deduplication ratio, `dedup_ratio`, which indicates the space that you can save by enabling the deduplication feature.

Example

The following sample code calculates and returns the deduplication ratio for `C`, a LOB column, in `ACME_TABLE`, which is owned by `JOHN`.

```
DECLARE
 dedup_ratio    number;
 l_table_name   varchar2(128) = "ACME_TABLE";
 l_column_name  varchar2(3000) = "C";
 l_tablespace_name varchar2(128) := "TBS";
 l_owner varchar2(128) := "JOHN";
BEGIN
dedup_ratio := dbms_lob.GET_LOB_DEDUPLICATION_RATIO(
    l_tablespace_name,
    l_owner,
    l_table_name,
    l_column_name,
    '',
    dedup_ratio,
    -1);
dbms_output.put_line('Deduplication ratio: ' || dedup_ratio);
END;
```

Usage Notes

- The maximum number of LOBs that this function can process is 100000 or 1% of the total number of rows in the table, whichever is lesser.

GETCONTENTTYPE Functions

This procedure returns the content type string previously set by means of the SETCONTENTTYPE Procedure.

Syntax

```
DBMS_LOB.GETCONTENTTYPE (
   lob_loc  IN BLOB)
 RETURN VARCHAR2;

DBMS_LOB.GETCONTENTTYPE (
   lob_loc  IN CLOB CHARACTER SET ANY_CS)
 RETURN VARCHAR2;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(getcontenttype, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-59 GETCONTENTTYPE Function Parameters

Parameter	Description
`lob_loc`	LOB whose content type is to be retrieved

Return Values

The returned content type.

If the SecureFiles LOB does not have a `contenttype` associated with it, `GETCONTENTTYPE()` returns `NULL`.

Exceptions

Table 120-60 GETCONTENTTYPE Function Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`

Related Topics

- SETCONTENTTYPE Procedure
  This procedure sets the content type string for the data in the LOB.

GET_STORAGE_LIMIT Function

This function returns the LOB storage limit for the specified LOB.

Syntax

```
DBMS_LOB.GET_STORAGE_LIMIT (
   lob_loc  IN CLOB CHARACTER SET ANY_CS)
 RETURN INTEGER;

DBMS_LOB.GET_STORAGE_LIMIT (
   lob_loc  IN BLOB)
 RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(get_storage_limit, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-61 GET_STORAGE_LIMIT Function Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.

Return Value

The value returned from this function is the maximum allowable size for specified LOB locator. For `BLOB`s, the return value depends on the block size of the tablespace the LOB resides in and is calculated as (2^{32})-1 (4294967295) times the block size of the tablespace. For `CLOB`s/`NCLOB`s, the value returned is the(2^{32})-1 (4294967295) times the block size of the tablespace divided by the character width of the `CLOB`/`NCLOB`.

Usage

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for details on LOB storage limits

GETCHUNKSIZE Functions

When creating the table, you can specify the chunking factor, a multiple of tablespace blocks in bytes. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value. This function returns the amount of space used in the LOB chunk to store the LOB value.

Syntax

```
DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(getchunksize, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-62 GETCHUNKSIZE Function Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.

Return Values

The return value is a usable chunk size in bytes.

Usage Notes

- With regard to basic LOB files, performance is improved if you enter read/write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is done or duplicated. You could batch up the `WRITE` until you have enough for a chunk, instead of issuing several `WRITE` calls for the same chunk.

  These tactics of performance improvement do not apply to SecureFiles.
- Note that chunk size is independent of LOB type (`BLOB`, `CLOB`, `NCLOB`, Unicode or other character set).

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

GETLENGTH Functions

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a `BFILE` includes the `EOF`, if it exists. Any 0-byte or space filler in the LOB caused by previous `ERASE` or `WRITE` operations is also included in the length count. The length of an empty internal LOB is 0.

Syntax

```
DBMS_LOB.GETLENGTH (
   lob_loc    IN  BLOB)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   lob_loc    IN  CLOB   CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   file_loc    IN  BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 120-63 GETLENGTH Function Parameter

Parameter	Description
`file_loc`	The file locator for the LOB whose length is to be returned.

Return Values

The length of the LOB in bytes or characters as an `INTEGER`. `NULL` is returned if the input LOB is `NULL` or if the input `lob_loc` is `NULL`. An error is returned in the following cases for `BFILE`s:

- `lob_loc` does not have the necessary directory and operating system privileges
- `lob_loc` cannot be read because of an operating system read error

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

GETOPTIONS Functions

This function obtains compression, deduplication, and encryption settings corresponding to the `option_type` field for a particular LOB.

Syntax

```
DBMS_LOB.GETOPTIONS (
   lob_loc             IN     BLOB,
   option_types        IN     PLS_INTEGER)
 RETURN PLS_INTEGER;

DBMS_LOB.GETOPTIONS (
  lob_loc             IN     CLOB CHARACTER SET ANY_CS,
  option_types        IN     PLS_INTEGER)
RETURN PLS_INTEGER;
```

Parameters

Table 120-64 GETOPTIONS Function Parameter

Parameter	Description
`lob_loc`	Locator for the LOB to be examined. For more information, see Operational Notes.
`option_type`	See Table 120-2

Return Values

The return values are a combination of `COMPRESS_ON`, `ENCRYPT_ON` and `DEDUPLICATE_ON` (see Table 120-3) depending on which option types (see Table 120-2) are passed in.

Exceptions

Table 120-65 GETOPTIONS Procedure Exceptions

Exception	Description
`INVALID_ARGVAL`	A parameter value was invalid
`QUERY_WRITE`	Cannot perform operation during a query
`SECUREFILE_BADLOB`	A non-`SECUREFILE` LOB was used in a `SECUREFILE` LOB only call

Usage Notes

You cannot turn compression or deduplication on or off for a SecureFile column that does not have those features on. The GetOptions Functions and SETOPTIONS Procedures work on individual SecureFiles. You can turn off a feature on a particular SecureFile and turn on a feature that has already been turned off by SetOptions, but you cannot turn on an option that has not been given to the SecureFile when the table was created.

INSTR Functions

This function returns the matching position of the *nth* occurrence of the pattern in the LOB, starting from the offset you specify.

Syntax

```
DBMS_LOB.INSTR (
   lob_loc    IN   BLOB,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   lob_loc    IN   CLOB      CHARACTER SET ANY_CS,
   pattern    IN   VARCHAR2  CHARACTER SET lob_loc%CHARSET,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   file_loc   IN   BFILE,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 120-66 INSTR Function Parameters

Parameter	Description
`lob_loc`	Locator for the LOB to be examined. For more information, see Operational Notes.
`file_loc`	The file locator for the LOB to be examined.
`pattern`	Pattern to be tested for. The pattern is a group of `RAW` bytes for `BLOBs`, and a character string (`VARCHAR2`) for `CLOBs`.The maximum size of the pattern is 16383 bytes.
`offset`	Absolute offset in bytes (`BLOBs`) or characters (`CLOBs`) at which the pattern matching is to start. (origin: 1)
`nth`	Occurrence number, starting at 1.

Return Values

Table 120-67 INSTR Function Return Values

Return	Description
`INTEGER`	Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.

Table 120-67 (Cont.) INSTR Function Return Values

Return	Description
NULL	Either: -any one or more of the `IN` parameters was `NULL` or `INVALID`. -`offset` < 1 or `offset` > `LOBMAXSIZE`. -`nth` < 1. -`nth` > `LOBMAXSIZE`.

Usage Notes

The form of the `VARCHAR2` buffer (the `pattern` parameter) must match the form of the `CLOB` parameter. In other words, if the input LOB parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input LOB parameter is of type `CLOB`, then the buffer must contain `CHAR` data.

For `BFILEs`, the file must be already opened using a successful `FILEOPEN` operation for this operation to succeed.

Operations that accept `RAW` or `VARCHAR2` parameters for pattern matching, such as `INSTR`, do not support regular expressions or special matching characters (as in the case of SQL `LIKE`) in the pattern parameter or substrings.

Exceptions

Table 120-68 INSTR Function Exceptions for BFILES

Exception	Description
`UNOPENED_FILE`	File was not opened using the input locator.
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.
`INVALID_OPERATION`	File does not exist, or you do not have access privileges on the file.

See Also:

- "SUBSTR Functions"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

ISOPEN Functions

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

Syntax

```
DBMS_LOB.ISOPEN (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   file_loc IN BFILE)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(isopen, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-69 ISOPEN Function Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.
`file_loc`	File locator.

Return Values

The return value is 1 if the LOB is open, 0 otherwise.

Usage Notes

For `BFILES`, openness is associated with the locator. If the input locator was never passed to `OPEN`, the `BFILE` is not considered to be opened by this locator. However, a different locator may have opened the `BFILE`. More than one `OPEN` can be performed on the same `BFILE` using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, `ISOPEN` requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (`BFILEs`), `ISOPEN` also requires a round-trip, because that's where the state is kept.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

ISREMOTE Function

This function checks to see if the LOB is local to the database or if it belongs to a remote database.

Syntax

```
DBMS_LOB.ISREMOTE (
   lob_loc IN BLOB)
  RETURN BOOLEAN;

DBMS_LOB.ISREMOTE (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN BOOLEAN;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(isremote, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-70   ISREMOTE Function Parameter

Parameter	Description
`lob_loc`	Locator for the `LOB`.

Return Values

`BOOLEAN`: `TRUE` for remote LOBs obtained over a database link; `FALSE` for LOBs obtained from local database

See Also:

- *Distributed LOBs* chapter in Database SecureFiles and Large Objects Developer's Guide for more details on the usage of this procedure.

ISSECUREFILE Function

This function returns `TRUE` if the LOB locator passed to it is for a SecureFile LOB. It returns `FALSE` otherwise.

Syntax

```
DBMS_LOB ISSECUREFILE(
    lob_loc     IN       BLOB)
  RETURN BOOLEAN;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(issecurefile, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-71 ISSECUREFILE Function Parameter

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.

Return Values

This function returns `TRUE` if the LOB locator passed to it is for a SecureFile LOB. It returns `FALSE` otherwise.

ISTEMPORARY Functions

This function determines whether a LOB instance is temporary.

Syntax

```
DBMS_LOB.ISTEMPORARY (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISTEMPORARY (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 120-72 ISTEMPORARY Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.

Return Values

The return value is 1 if the LOB is temporary and exists; 0 if the LOB is not temporary or does not exist; `NULL` if the given locator is `NULL`.

Usage Notes

When you free a Temporary LOB with `FREETEMPORARY`, the LOB locator is not set to `NULL`. Consequently, `ISTEMPORARY` will return 0 for a locator that has been freed but not explicitly reset to `NULL`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

LOADBLOBFROMFILE Procedure

This procedure loads data from `BFILE` to internal `BLOB`. This achieves the same outcome as `LOADFROMFILE`, and returns the new offsets.

Syntax

```
DBMS_LOB.LOADBLOBFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_bfile   IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN OUT        INTEGER,
   src_offset  IN OUT        INTEGER);
```

Parameters

Table 120-73 LOADBLOBFROMFILE Procedure Parameters

Parameter	Description
`dest_lob`	`BLOB` locator of the target for the load.
`src_bfile`	`BFILE` locator of the source for the load.
`amount`	Number of bytes to load from the `BFILE`. You can also use `DBMS_LOB.LOBMAXSIZE` to load until the end of the `BFILE`.
`dest_offset`	(`IN`) Offset in bytes in the destination `BLOB` (origin: 1) for the start of the write. `(OUT) New` offset in bytes in the destination BLOB right after the end of this write, which is also where the next write should begin.
`src_offset`	`(IN) Offset` in bytes in the source `BFILE` (origin: 1) for the start of the read. `(OUT) Offset` in bytes in the source `BFILE` right after the end of this read, which is also where the next read should begin.

Usage Notes

- You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source `BFILE`. The `amount` and `src_offset`, because they refer to the `BFILE`, are in terms of bytes, and the `dest_offset` is in bytes for `BLOB`s.
- If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination `BLOB`. If the offset is less than the current length of the destination LOB, then existing data is overwritten.
- There is an error if the input amount plus offset exceeds the length of the data in the `BFILE` (unless the amount specified is `LOBMAXSIZE` which you can specify to continue loading until the end of the `BFILE` is reached).
- It is not mandatory that you wrap the LOB operation inside the `OPEN/CLOSE` operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the `OPEN/CLOSE`, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

- LOADFROMFILE gets the destination LOB prior to the load unless the load covers the entire LOB.

Constants and Defaults

There is no easy way to omit parameters. You must either declare a variable for `IN/OUT` parameter or provide a default value for the `IN` parameter. Here is a summary of the constants and the defaults that can be used.

Table 120-74 Suggested Values of the Parameter

Parameter	Default Value	Description
`amount`	`DBMS_LOB.LOBMAXSIZE (IN)`	Load the entire file
`dest_offset`	`1 (IN)`	start from the beginning
`src_offset`	`1 (IN)`	start from the beginning

Constants defined in `DBMSLOB.SQL`

```
lobmaxsize                     CONSTANT INTEGER        := DBMS_LOB.LOBMAXSIZE;
```

Exceptions

Table 120-75 LOADBLOBFROMFILE Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	Either: - `src_offset` or `dest_offset` < 1. - `src_offset` or `dest_offset` > `LOBMAXSIZE`. - `amount` < 1. - `amount` > `LOBMAXSIZE`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

LOADCLOBFROMFILE Procedure

This procedure loads data from a `BFILE` to an internal `CLOB/NCLOB` with necessary character set conversion and returns the new offsets.

Syntax

```
DBMS_LOB.LOADCLOBFROMFILE (
   dest_lob       IN OUT NOCOPY   NOCOPY CLOB CHARACTER SET ANY_CS,
   src_bfile      IN              BFILE,
   amount         IN              INTEGER,
   dest_offset    IN OUT          INTEGER,
   src_offset     IN OUT          INTEGER,
   bfile_csid      IN            NUMBER,
   lang_context    IN OUT        INTEGER,
   warning         OUT           INTEGER);
```

Parameters

Table 120-76 LOADCLOBFROMFILE Procedure Parameters

Parameter	Description
`dest_lob`	`CLOB/NCLOB` locator of the target for the load.
`src_bfile`	`BFILE` locator of the source for the load.
`amount`	Number of bytes to load from the `BFILE`. Use `DBMS_LOB.LOBMAXSIZE` of load until the end of the `BFILE`.
`dest_offset`	`(IN) Offset` in characters in the destination `CLOB` (origin: 1) for the start of the write. `(OUT)` The new offset in characters right after the end of this load, which is also where the next load should start. It always points to the beginning of the first complete character after the end of load. If the last character is not complete, offset goes back to the beginning of the partial character.
`src_offset`	`(IN) Offset` in bytes in the source `BFILE` (origin: 1) for the start of the read. `(OUT)` Offset in bytes in the source `BFILE` right after the end of this read, which is also where the next read should begin.
`bfile_csid`	`Character set id of the source (BFILE)` file.
`lang_context`	`(IN)` Language context, such as shift status, for the current load. `(OUT)` The language context at the time when the current load stopped, and what the next load should be using if continuing loading from the same source. This information is returned to the user so that they can use it for the continuous load without losing or misinterpreting any source data. For the very first load or if do not care, simply use the default 0. The details of this language context is hidden from the user. One does not need to know what it is or what's in it in order to make the call
`warning`	(OUT) Warning message. This indicates something abnormal happened during the loading. It may or may not be caused by the user's mistake. The loading is completed as required, and it's up to the user to check the warning message. Currently, the only possible warning is the inconvertible character. This happens when the character in the source cannot be properly converted to a character in destination, and the default replacement character (for example, '?') is used in place. The message is defined the constant value `DBMS_LOB.WARN_INCONVERTIBLE_CHAR`.

Usage Notes

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source `BFILE`. The `amount` and `src_offset`, because they refer to the `BFILE`, are in terms of bytes, and the `dest_offset` is in characters for `CLOB`s.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination `CLOB`. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the `BFILE` (unless the amount specified is `LOBMAXSIZE` which you can specify to continue loading until the end of the `BFILE` is reached).

Note the following requirements:

- The destination character set is always the same as the database character set in the case of `CLOB` and national character set in the case of `NCLOB`.
- `csid=0` indicates the default behavior that uses database `csid` for `CLOB` and national `csid` for `NCLOB` in the place of source `csid`. Conversion is still necessary if it is of varying width
- It is not mandatory that you wrap the LOB operation inside the `OPEN/CLOSE` operations. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

  If you do not wrap the LOB operation inside the `OPEN/CLOSE`, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. Therefore, it is recommended that you enclose write operations to the LOB within the `OPEN` or `CLOSE` statement.

The source `BFILE` can contain data in the Unicode character set. The Unicode standard defines many encoding schemes that provide mappings from Unicode characters to sequences of bytes. Table 120-77 lists Unicode encodings schemes supported by this subprogram.

Table 120-77 Supported Unicode Encoding Schemes

Encoding Scheme	Oracle Name	bfile_csid Value
`UTF-8`	`AL32UTF8`	873
`UTF-16BE`	`AL16UTF16`	2000
`UTF-16LE`	`AL16UTF16LE`	2002
`CESU-8`	`UTF8`	871
`UTF-EBCDIC`	`UTFE`	872
`UTF-16`	`UTF16`	1000

All three `UTF-16` encoding schemes encode Unicode characters as 2-byte unsigned integers. Integers can be stored in big-endian or in little-endian byte order. The `UTF-16BE` encoding scheme defines big-endian data. The `UTF-16LE` scheme defines little-endian data. The `UTF-16` scheme requires that the source `BFILE` contains the Byte Order Mark (BOM) character in the first two bytes to define the byte order. The BOM code is `0xFEFF`. If the code is stored as `{0xFE,0xFF}`, the data is interpreted as big-endian. If it is stored as `{0xFF,0xFE}`, the data is interpreted as little-endian.

In `UTF-8` and in `CESU-8` encodings the Byte Order Mark is stored as `{0xEF,0xBB, 0xBF}`. With any of the Unicode encodings, the corresponding BOM sequence at the beginning of the file is recognized and not loaded into the destination LOB.

Constants

Here is a summary of the constants and the suggested values that can be used.

Table 120-78 Suggested Values of the LOADCLOBFROMFILE Parameter

Parameter	Suggested Value	Description
`amount`	`DBMS_LOB.LOBMAXSIZE (IN)`	Load the entire file
`dest_offset`	`1 (IN)`	start from the beginning
`src_offset`	`1 (IN)`	start from the beginning
`csid`	`0 (IN)`	default csid, use destination csid
`lang_context`	`0 (IN)`	default language context
`warning`	`0 (OUT)`	no warning message, everything is ok

Constants defined in `DBMSLOB.SQL`

```
lobmaxsize                    CONSTANT INTEGER        := 18446744073709551615;
warn_inconvertible_char       CONSTANT INTEGER        := 1;
default_csid                  CONSTANT INTEGER        := 0;
default_lang_ctx              CONSTANT INTEGER        := 0;
no_warning                    CONSTANT INTEGER        := 0;
```

Exceptions

Table 120-79 LOADCLOBFROMFILE Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of the input parameters are `NULL` or `INVALID`.
`INVALID_ARGVAL`	Either: - `src_offset` or `dest_offset` < 1. - `src_offset` or `dest_offset` > `LOBMAXSIZE`. - `amount` < 1. - `amount` > `LOBMAXSIZE`.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

MOVE_TO_DBFS_LINK Procedures

This procedure archives the specified LOB data (from the database) into the DBFS HSM Store.

Syntax

```
DBMS_LOB.MOVE_TO_DBFS_LINK (
  lob_loc       IN OUT BLOB,
  storage_path  IN     VARCHAR2(dbfs_link_path_max_size),
  flags         IN     BINARY INTEGER DEFAULT DBFS_LINK_NOCACHE);

DBMS_LOB.MOVE_TO_DBFS_LINK (
  lob_loc       IN OUT CLOB CHARACTER SET ANY_CS,
  storage_path  IN     VARCHAR2(dbfs_link_path_max_size),
  flags         IN     BINARY INTEGER DEFAULT DBFS_LINK_NOCACHE);
```

Parameters

Table 120-80 MOVE_TO_DBFS_LINK Procedure Parameters

Parameter	Description
`lob_loc`	LOB to be archived
storage_path	Path where the LOB will be stored
`flags`	Either `DBFS_LINK_CACHE` or `DBFS_LINK_NOCACHE`. If `DBFS_LINK_CACHE` is specified, the LOB data continues to be stored in the RDBMS as well as being written to the DBFS store. `DBFS_LINK_NOCACHE` specifies that the LOB data should be deleted from the `RDBMS` once written to the DBFS.

Exceptions

Table 120-81 MOVE_TO_DBFS_LINK Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`

Usage Notes

- If the LOB is already archived, the procedure silently returns as if the put was successful. In that case, if `DBFS_LINK_NOCACHE` is specified, or `flags` is defaulted, the LOB data is removed from the RDBMS.
- Calling this procedure multiple times on the same LOB with the same flags has no effect.
- Calling the procedure on a LOB that is already archived causes the LOB to be cached (`DBFS_LINK_CACHE`) or removed (`DBFS_LINK_NOCACHE`) according to the flag setting.

OPEN Procedures

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read/write.

Syntax

```
DBMS_LOB.OPEN (
   lob_loc    IN OUT NOCOPY BLOB,
   open_mode  IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   open_mode  IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   file_loc   IN OUT NOCOPY BFILE,
   open_mode  IN            BINARY_INTEGER := file_readonly);
```

Parameters

Table 120-82 OPEN Procedure Parameters

Parameter	Description
`lob_loc`	LOB locator. For more information, see Operational Notes.
`open_mode`	Mode in which to open. For `BLOB` and `CLOB` types, the mode can be either: `LOB_READONLY` or `LOB_READWRITE`. For `BFILE` types, the mode must be `FILE_READONLY`.

Usage Notes

Note:

If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. `BFILE` can only be opened with read-only mode.

`OPEN` requires a round-trip to the server for both internal and external LOBs. For internal LOBs, `OPEN` triggers other code that relies on the `OPEN` call. For external LOBs (`BFILEs`), `OPEN` requires a round-trip because the actual operating system file on the server side is being opened.

It is not mandatory that you wrap all LOB operations inside the Open/Close interfaces. However, if you open a LOB, you must close it before you commit the transaction; an error is produced if you do not. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed, but the domain and function-based indexes are not updated. If this happens, you should rebuild the functional and domain indexes on the LOB column.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

READ Procedures

This procedure reads a piece of a LOB, and returns the specified amount into the `buffer` parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the `amount` parameter. If the input `offset` points past the End of LOB, then `amount` is set to 0, and a `NO_DATA_FOUND` exception is raised.

Syntax

```
DBMS_LOB.READ (
   lob_loc   IN              BLOB,
   amount    IN OUT  NOCOPY INTEGER,
   offset    IN              INTEGER,
   buffer    OUT             RAW);

DBMS_LOB.READ (
   lob_loc   IN              CLOB CHARACTER SET ANY_CS,
   amount    IN OUT  NOCOPY INTEGER,
   offset    IN              INTEGER,
   buffer    OUT             VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
   file_loc   IN              BFILE,
   amount    IN OUT   NOCOPY INTEGER,
   offset    IN               INTEGER,
   buffer    OUT              RAW);
```

Parameters

Table 120-83 READ Procedure Parameters

Parameter	Description
`lob_loc`	Locator for the LOB to be read. For more information, see Operational Notes.
`file_loc`	The file locator for the LOB to be examined.
`amount`	Number of bytes (for `BLOB`s) or characters (for `CLOB`s) to read, or number that were read.
`offset`	Offset in bytes (for `BLOB`s) or characters (for `CLOB`s) from the start of the LOB (origin: 1).
`buffer`	Output buffer for the read operation.

Exceptions

Table 120-84 lists exceptions that apply to any LOB instance. Table 120-85 lists exceptions that apply only to `BFILE`s.

Table 120-84 READ Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of `lob_loc`, `amount`, or `offset` parameters are `NULL`.
`INVALID_ARGVAL`	Either: - `amount` < 1 - `amount` > `32767 bytes (or the character equivalent)` - `offset` < 1 - `offset` > `LOBMAXSIZE` - `amount` is greater, in bytes or characters, than the capacity of `buffer`.
`NO_DATA_FOUND`	End of the LOB is reached, and there are no more bytes or characters to read from the LOB: `amount` has a value of 0.

Table 120-85 READ Procedure Exceptions for BFILEs

Exception	Description
`UNOPENED_FILE`	File is not opened using the input locator.
`NOEXIST_DIRECTORY`	Directory does not exist.
`NOPRIV_DIRECTORY`	You do not have privileges for the directory.
`INVALID_DIRECTORY`	Directory has been invalidated after the file was opened.
`INVALID_OPERATION`	File does not exist, or you do not have access privileges on the file.

Usage Notes

- The form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. In other words, if the input LOB parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input LOB parameter is of type `CLOB`, then the buffer must contain `CHAR` data.
- When calling `DBMS_LOB.READ` from the client (for example, in a `BEGIN`/`END` block from within SQL*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.
- READ gets the LOB, if necessary, before the read.
- If the LOB is a DBFS LINK, data is streamed from DBFS, if possible, otherwise an exception is thrown.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for additional details on usage of this procedure

SET_DBFS_LINK Procedures

This function links the specified SecureFile to the specified path name. It does not copy the data to the path.

Syntax

```
DBMS_LOB.SET_DBFS_LINK (
  lob_loc        IN OUT BLOB,
  archive_id     IN     RAW(1024));

DBMS_LOB.SET_DBFS_LINK(
  lob_loc_dst    IN OUT CLOB CHARACTER SET ANY_CS,
  archive_id     IN     RAW(1024));
```

Parameters

Table 120-86 SET_DBFS_LINK Procedure Parameters

Parameter	Description
`lob_loc`	LOB for which to store the reference value
`archive_id`	Archive ID as returned by calling either of the GET_DBFS_LINK Functions Functions

Exceptions

Table 120-87 SET_DBFS_LINK Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`

SETCONTENTTYPE Procedure

This procedure sets the content type string for the data in the LOB.

Syntax

```
DBMS_LOB.SETCONTENTTYPE (
   lob_loc      IN OUT NOCOPY BLOB,
   contenttype  IN            VARCHAR2);

DBMS_LOB.SETCONTENTTYPE (
   lob_loc     IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   contenttype IN            VARCHAR2);
```

Parameters

Table 120-88 SETCONTENTTYPE Procedure Parameters

Parameter	Description
`lob_loc`	LOB to be assigned the content type
`contenttype`	String to be assigned

Exceptions

Table 120-89 SETCONTENTTYPE Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	`lob_loc` is not a `SECUREFILE`

Usage Notes

To clear an existing content type associated with a `SECUREFILE`, invoke `SETCONTENTTYPE` with `contenttype` set to empty string.

SETOPTIONS Procedures

This procedure enables/disables compression and deduplication on a per-LOB basis, overriding the default LOB column settings.

Syntax

```
DBMS_LOB.SETOPTIONS (
   lob_loc              IN     BLOB,
   option_types         IN     PLS_INTEGER,
   options              IN     PLS_INTEGER);

DBMS_LOB.SETOPTIONS (
  lob_loc              IN     CLOB CHARACTER SET ANY_CS,
  option_types         IN     PLS_INTEGER,
  options              IN     PLS_INTEGER);
```

Parameters

Table 120-90   SETOPTIONS Procedure Parameter

Parameter	Description
`lob_loc`	Locator for the LOB to be examined. For more information, see Operational Notes.
`option_type`	See Table 120-2
`options`	See Table 120-3

Exceptions

Table 120-91   SETOPTIONS Procedure Exceptions

Exception	Description
`SECUREFILE_BADLOB`	Unsupported object type for the operation
`INVALID_ARGVAL`	A parameter value was invalid
`QUERY_WRITE`	Cannot perform operation during a query

Usage Notes

- `DBMS_LOB.SETOPTIONS` cannot be used to enable or disable encryption on individual LOBs.
- You cannot turn the compression or deduplication features on or off for a SecureFile column if they were not turned when the table was created.

  The GETOPTIONS Functions and `SETOPTIONS` Procedures work on individual SecureFiles. You can turn off compression or deduplication on a particular SecureFiles LOB and turn on them on, *if* they have already been turned off by `SETOPTIONS`.
- This call incurs a round-trip to the server to make the changes persistent.

SUBSTR Functions

This function returns `amount` bytes or characters of a LOB, starting from an absolute `offset` from the beginning of the LOB.

For fixed-width `n`-byte `CLOB`s, if the input amount for `SUBSTR` is greater than (32767/`n`), then `SUBSTR` returns a character buffer of length (32767/`n`), or the length of the `CLOB`, whichever is lesser. For CLOBs in a varying-width character set, `n` is the maximum byte-width used for characters in the CLOB.

Syntax

```
DBMS_LOB.SUBSTR (
   lob_loc     IN    BLOB,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;

DBMS_LOB.SUBSTR (
   lob_loc     IN    CLOB   CHARACTER SET ANY_CS,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;

DBMS_LOB.SUBSTR (
   file_loc     IN    BFILE,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 120-92 SUBSTR Function Parameters

Parameter	Description
`lob_loc`	Locator for the LOB to be read. For more information, see Operational Notes.
`file_loc`	The file locator for the LOB to be examined.
`amount`	Number of bytes (for `BLOB`s) or characters (for `CLOB`s) to be read.
`offset`	Offset in bytes (for `BLOB`s) or characters (for `CLOB`s) from the start of the LOB (origin: 1).

Return Values

Table 120-93 SUBSTR Function Return Values

Return	Description
`RAW`	Function overloading that has a `BLOB` or `BFILE` in parameter.
`VARCHAR2`	`CLOB` version.

Table 120-93 (Cont.) SUBSTR Function Return Values

Return	Description
NULL	Either: - any input parameter is NULL - amount < 1 - amount > 32767 - offset < 1 - offset > LOBMAXSIZE

Exceptions

Table 120-94 SUBSTR Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.
- When calling DBMS_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. The database converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.
- DBMS_LOB.SUBSTR will return 8191 or more characters based on the characters stored in the LOBs. If all characters are not returned as a consequence of the character byte size exceeding the available buffer, the user should either call DBMS_LOB.SUBSTR with a new offset to read the remaining characters, or call the subprogram on loop until all the data is extracted.
- SUBSTR gets the LOB, if necessary, before read.
- If the LOB is a DBFS Link, the data is streamed from DBFS, if possible, otherwise, an exception is thrown.

See Also:

- "INSTR Functions"
- "READ Procedures"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

TRIM Procedures

This procedure trims the value of the internal LOB to the length you specify in the `newlen` parameter.

Specify the length in bytes for `BLOBs`, and specify the length in characters for `CLOBs`.

Note:

The `TRIM` procedure decreases the length of the LOB to the value specified in the `newlen` parameter.

If you attempt to `TRIM` an empty LOB, then nothing occurs, and `TRIM` returns no error. If the new length that you specify in `newlen` is greater than the size of the LOB, then an exception is raised.

Syntax

```
DBMS_LOB.TRIM (
   lob_loc        IN OUT  NOCOPY BLOB,
   newlen         IN             INTEGER);

DBMS_LOB.TRIM (
   lob_loc        IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS,
   newlen         IN             INTEGER);
```

Parameters

Table 120-95 TRIM Procedure Parameters

Parameter	Description
`lob_loc`	Locator for the internal LOB whose length is to be trimmed. For more information, see Operational Notes.
`newlen`	New, trimmed length of the LOB value in bytes for `BLOBs` or characters for `CLOBs`.

Exceptions

Table 120-96 TRIM Procedure Exceptions

Exception	Description
`VALUE_ERROR`	`lob_loc` is `NULL`.
`INVALID_ARGVAL`	Either: - `new_len` < 0 - `new_len` > `LOBMAXSIZE`
`QUERY_WRITE`	Cannot perform a LOB write inside a query or PDML parallel execution server

Usage Notes

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the `LOB` column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.
- `TRIM` gets the LOB, if necessary, before altering the length of the LOB, unless the new length specified is '0'

See Also:

- "ERASE Procedures"
- "WRITEAPPEND Procedures"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

WRITE Procedures

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the `buffer` parameter.

`WRITE` replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

Syntax

```
DBMS_LOB.WRITE (
   lob_loc   IN OUT NOCOPY  BLOB,
   amount    IN             INTEGER,
   offset   IN             INTEGER,
   buffer   IN             RAW);

DBMS_LOB.WRITE (
   lob_loc  IN OUT  NOCOPY CLOB   CHARACTER SET ANY_CS,
   amount   IN             INTEGER,
   offset   IN             INTEGER,
   buffer   IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 120-97 WRITE Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to. For more information, see Operational Notes
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.
buffer	Input buffer for the write

Exceptions

Table 120-98 WRITE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > 32767 bytes (or the character equivalent) - offset < 1 - offset > LOBMAXSIZE
QUERY_WRITE	Cannot perform a LOB write inside a query or PDML parallel execution server
SECUREFILE_OUTOFBOUNDS	Attempted to perform a write operation past the end of a LOB having FRAGMENT_* on it

Usage Notes

- There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.
- The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.
- When calling DBMS_LOB.WRITE from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. The database

converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.
- `WRITE` gets the LOB, if necessary, before writing the LOB, unless the write is specified to overwrite the entire LOB.

See Also:

- "APPEND Procedures"
- "COPY Procedures"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure

WRITEAPPEND Procedures

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the `buffer` parameter.

Syntax

```
DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY BLOB,
   amount  IN            INTEGER,
   buffer  IN            RAW);

DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount  IN            INTEGER,
   buffer  IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Parameters

Table 120-99 WRITEAPPEND Procedure Parameters

Parameter	Description
`lob_loc`	Locator for the internal LOB to be written to. For more information, see Operational Notes
`amount`	Number of bytes (for `BLOB`s) or characters (for `CLOB`s) to write
`buffer`	Input buffer for the write

Usage Notes

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

Exceptions

Table 120-100 WRITEAPPEND Procedure Exceptions

Exception	Description
`VALUE_ERROR`	Any of `lob_loc`, `amount`, or `offset` parameters are `NULL`, out of range, or `INVALID`.
`INVALID_ARGVAL`	Either: - `amount` < 1 - `amount` > `32767 bytes (or the character equivalent)`
`QUERY_WRITE`	Cannot perform a LOB write inside a query or PDML parallel execution server

Usage Notes

- The form of the `VARCHAR2` buffer must match the form of the `CLOB` parameter. In other words, if the input LOB parameter is of type `NCLOB`, then the buffer must contain `NCHAR` data. Conversely, if the input LOB parameter is of type `CLOB`, then the buffer must contain `CHAR` data.
- When calling `DBMS_LOB.WRITEAPPEND` from the client (for example, in a `BEGIN`/`END` block from within SQL*Plus), the buffer must contain data in the client's character set. The database converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.
- It is not mandatory that you wrap the LOB operation inside the Open/Close interfaces. If you did not open the LOB before performing the operation, the functional and domain indexes on the LOB column are updated during the call. However, if you opened the LOB before performing the operation, you must close it before you commit the transaction. When an internal LOB is closed, it updates the functional and domain indexes on the LOB column.
- If you do not wrap the LOB operation inside the Open/Close API, the functional and domain indexes are updated each time you write to the LOB. This can adversely affect performance. If a LOB is under `IS JSON FORMAT JSON` constraints, and you do not wrap the LOB operations inside the Open/Close API, it checks that the LOB content is a valid JSON for every write operation on the LOB.
- `WRITEAPPEND` gets the LOB, if necessary, before appending to the LOB.

See Also:

- "APPEND Procedures"
- "COPY Procedures"
- "WRITE Procedures"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for additional details on usage of this procedure