5
BFILEs

BFILEs are data objects stored in operating system files, outside the database tablespaces. Data stored in a table column of type BFILE is physically located in an operating system file, not in the database. The BFILE column stores a reference to the operating system file.

BFILEs are read-only data types. The database allows read-only byte stream access to data stored in BFILEs. You cannot write to or update a BFILE from within your application.

You create BFILEs to hold the following types of data:

- Binary data that does not change while your application is running, such as graphics.
- Data that is loaded into other large object types, such as a BLOB or CLOB, where the data can be manipulated.
- Data that is appropriate for byte-stream access, such as multimedia.

Any storage device accessed by your operating system can hold BFILE data, including hard disk drives, CD-ROMs, PhotoCDs, and DVDs. The database can access BFILEs provided the operating system supports stream-mode access to the operating system files.

- DIRECTORY Objects
  A BFILE locator is initialized by using the function BFILENAME(DIRECTORY, FILENAME). This section describes how to initialize the DIRECTORY Object.
- BFILE Locators
  For BFILEs, the value is stored in a server-side operating system file, in other words, BFILEs are external to the database. The BFILE locator that refers to the file is stored in the database row.
- BFILE APIs
  This section discusses the different operations supported through BFILEs.
- BFILE APIs in Different Programmatic Interfaces
  This section lists all the APIs from different Programmatic Interfaces supported by Oracle Database.

5.1 DIRECTORY Objects

A BFILE locator is initialized by using the function BFILENAME(DIRECTORY, FILENAME). This section describes how to initialize the DIRECTORY Object.

A DIRECTORY object specifies a *logical alias name* for a physical directory on the database server file system under which the file to be accessed is located. You can access a file in the server file system only if you have the required access privilege on the DIRECTORY object. You can also use Oracle Enterprise Manager Cloud Control to manage the DIRECTORY objects.

The DIRECTORY object provides the flexibility to manage the locations of the files, instead of forcing you to hard-code the absolute path names of physical files in your applications.

A DIRECTORY object name is used in conjunction with the BFILENAME function, in SQL and PL/SQL, or the OCILobFileSetName() function in OCI, for initializing a BFILE locator.

- DIRECTORY Name Specification
  You must have `CREATE ANY DIRECTORY` system privilege to create directories.
- Security on Directory Objects
  This section describes the security on `DIRECTORY` objects.

> **See Also:**
>
> - `CREATE DIRECTORY` in *Oracle Database SQL Language Reference*
> - See *Oracle Database Administrator's Guide* for the description of Oracle Enterprise Manager Cloud Control

5.1.1 DIRECTORY Name Specification

You must have `CREATE ANY DIRECTORY` system privilege to create directories.

The naming convention for `DIRECTORY` objects is the same as that for tables and indexes. That is, normal identifiers are interpreted in uppercase, but delimited identifiers are interpreted as is. For example, the following statement:

```
CREATE OR REPLACE DIRECTORY scott_dir AS '/usr/home/scott';
```

creates or redefines a `DIRECTORY` object whose name is `'SCOTT_DIR'` (in uppercase). But if a delimited identifier is used for the `DIRECTORY` name, as shown in the following statement

```
CREATE DIRECTORY "Mary_Dir" AS '/usr/home/mary';
```

then the `DIRECTORY`directory object name is `'Mary_Dir'`. Use `'SCOTT_DIR'` and `'Mary_Dir'` when calling `BFILENAME`. For example:

```
BFILENAME('SCOTT_DIR', 'afile')
BFILENAME('Mary_Dir', 'afile')
```

> **WARNING:**
>
> The database does not verify that the directory and path name you specify actually exist. You must ensure to specify a valid directory name in your operating system. If your operating system uses case-sensitive path names, then be sure that you specify the directory name in the correct format. There is no requirement to specify a terminating slash (for example, `/tmp/` is not necessary, simply use `/tmp`).
>
> Directory specifications cannot contain ".." anywhere in the path (for example: `../../abc/def` or `abc/../def` or `abc/def/hij..`

On Windows Platform

On Windows platforms the directory names are case-insensitive. Therefore the following two statements refer to the same directory:

```
CREATE DIRECTORY "big_cap_dir" AS "g:\data\source";

CREATE DIRECTORY "small_cap_dir" AS "G:\DATA\SOURCE";
```

5.1.2 Security on Directory Objects

This section describes the security on DIRECTORY objects.

The DIRECTORY object model has two distinct levels of security:

- SQL DDL: CREATE or DROP a DIRECTORY object
- SQL DML: READ system and object privileges on DIRECTORY objects

DBA Privileges: CREATE / DROP DIRECTORY

The DIRECTORY object is a *system owned* object. Oracle Database supports the following system privileges, which are granted only to DBA:

- CREATE ANY DIRECTORY: For creating or altering the DIRECTORY object creation
- DROP ANY DIRECTORY: For deleting the DIRECTORY object

> **WARNING:**
>
> Because **CREATE ANY DIRECTORY** and **DROP ANY DIRECTORY** privileges potentially expose the server file system to all database users, the DBA should be prudent in granting these privileges to normal database users to prevent security breach.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about system owned objects, CREATE DIRECTORY and DROP DIRECTORY

USER Privileges: READ Permission on the Directory

READ permission on the DIRECTORY object enables you to read files located under that directory. The creator of the DIRECTORY object automatically earns the READ privilege.

If you have been granted the READ permission with GRANT option, then you may in turn grant this privilege to other users or roles and then add them to your privilege domains.

> **Note:**
>
> The READ permission is defined only on the DIRECTORY *object*, not on individual files. Hence there is no way to assign different privileges to files in the same directory.

The physical directory that it represents may or may not have the corresponding operating system privileges (*read* in this case) for the Oracle Server process.

It is the responsibility of the DBA to ensure the following:

- That the physical directory exists
- *Read* permission for the Oracle Server process is enabled on the file, the directory, and the path leading to it

- The directory remains available, and *read* permission remains enabled, for the entire duration of file access by database users

The privilege just implies that as far as the Oracle Server is concerned, you may read from files in the directory. These privileges are checked and enforced by the PL/SQL `DBMS_LOB` package and OCI APIs at the time of the actual file operations.

> **See Also:**
>
> - Guidelines for DIRECTORY Usage
> - *Oracle Database SQL Language Reference* for information about the `GRANT`, `REVOKE` and `AUDIT` system and object privileges that provide security for `BFILE`s.

Catalog Views on DIRECTORY Objects

Catalog views are provided for `DIRECTORY` objects to enable users to view object names and corresponding paths and privileges. Following are the supported views:

- `ALL_DIRECTORIES` (`OWNER`, `DIRECTORY_NAME`, `DIRECTORY_PATH`)
  This view describes all directories accessible to the user.
- `DBA_DIRECTORIES`(`OWNER`, `DIRECTORY_NAME`, `DIRECTORY_PATH`)
  This view describes all directories specified for the entire database.

5.2 BFILE Locators

For `BFILE`s, the value is stored in a server-side operating system file, in other words, `BFILE`s are external to the database. The `BFILE` locator that refers to the file is stored in the database row.

To associate an operating system file to a `BFILE`, first create a `DIRECTORY` object that is an alias for the full path name to the operating system file. Then, you can initialize an instance of `BFILE` type, using the `BFILENAME` function in SQL or PL/SQL, or `OCILobFileSetName()` in OCI. You can use this `BFILE` instance in one of the following ways:

- If your need for a particular `BFILE` is temporary and limited within the module on which you are working, then you can assign this `BFILE` instance to a PL/SQL or OCI local variable of type `BFILE`. Subsequently, you can use the `BFILE` related APIs on this variable without having to associate this with a column in the database. The `BFILE` API operations on a temporary instance are executed on the client side, without any round-trips to the server.
- You can insert a persistent reference to a `BFILE` in the `BFILE` column using an `INSERT` or `UPDATE` statement. Before using SQL to insert or update a row with a `BFILE`, you must initialize the `BFILE` variable to either `NULL` or a `DIRECTORY` object name and file name.

> **Note:**
>
> The `OCISetAttr()` function does not allow you to set a `BFILE` locator to `NULL`. To insert a `NULL BFILE` in OCI, you must set the bind value to `NULL`.

It is possible to have multiple `BFILE` columns in the same record or different records referring to the same file. For example, the following `UPDATE` statements set the `BFILE` column of the row with `key_value = 21` in `lob_table` to point to the same file as the row with `key_value = 22`.

```
UPDATE lob_table SET f_lob = (SELECT f_lob FROM lob_table WHERE key_value =
22) WHERE
      key_value = 21;
```

> **See Also:**
>
> Loading BFILEs with SQL*Loader

BFILEs in Objects

If you are using `BFILE`s in objects, you must first set the `BFILE` value, and then flush the object to the database. So, you must first call the `OCIObjectNew()` function, followed by the `OCILobFileSetName()` function and the `OCIObjectFlush()` function respectively.

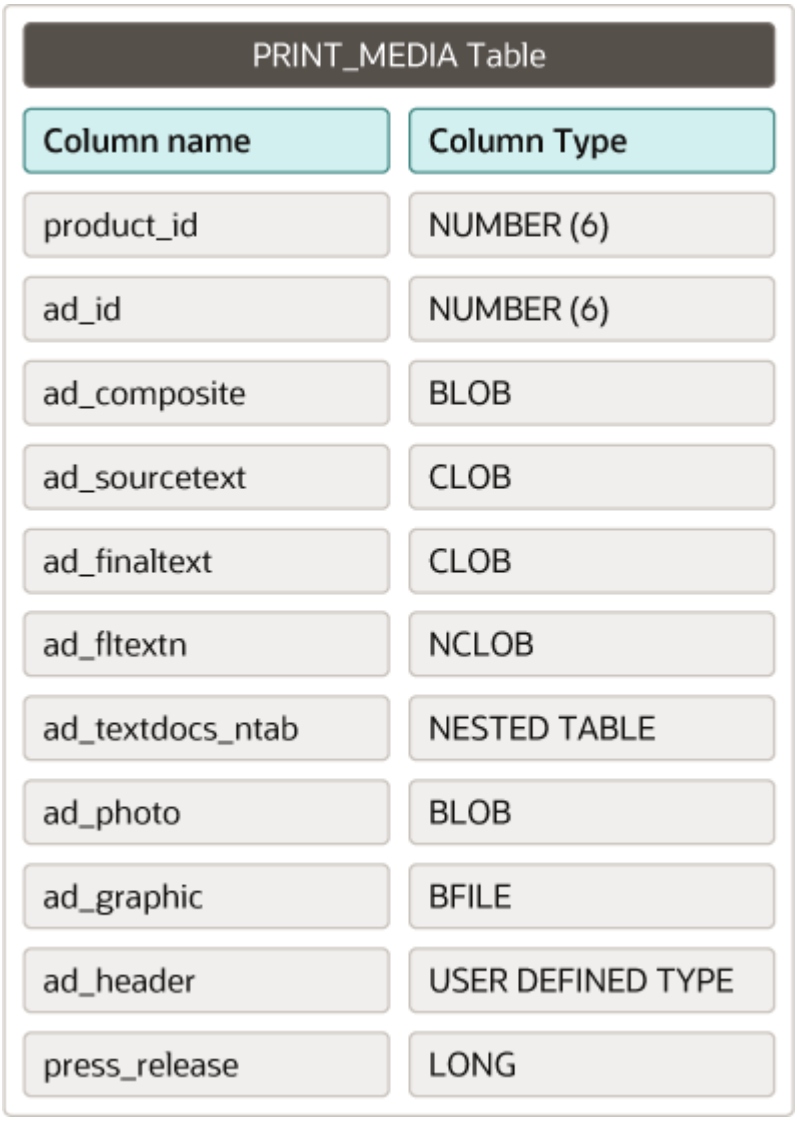
BFILEs in Shared Server (Multithreaded Server) Mode

The database does not support session migration for `BFILE` data types in shared server (multithreaded server) mode. This implies that in shared server sessions, `BFILE` operations are bound to one shared server, they cannot migrate from one server to another, and open `BFILE` instances can persist beyond the end of a call to a shared server.

Examples of Creating Directory Objects and BFILE Locators

Many examples in the following sections use the `print_media` table. Following is the structure of the table:

Figure 5-1 print_media table



PRINT_MEDIA Table	
Column name	**Column Type**
product_id	NUMBER (6)
ad_id	NUMBER (6)
ad_composite	BLOB
ad_sourcetext	CLOB
ad_finaltext	CLOB
ad_fltextn	NCLOB
ad_textdocs_ntab	NESTED TABLE
ad_photo	BLOB
ad_graphic	BFILE
ad_header	USER DEFINED TYPE
press_release	LONG

Example 5-1 Inserting BFILEs in SQL and PL/SQL

```
conn system/manager

 -- The DBA creates DIRECTORY object and grants READ to the user
create or replace directory MYDIR as '/your/directory/path/here';
GRANT read ON DIRECTORY MYDIR TO pm;

conn pm/pm

 -- Use BFILENAME to create a BFILE locator for INSERT
INSERT INTO print_media
(product_id, ad_id, ad_composite, ad_sourcetext, ad_graphic)
VALUES
(1, 1, empty_blob(), empty_clob(), BFILENAME('MYDIR','file1.txt'));

-- After this statement, 2 rows point to the same BFILE
```

```
INSERT INTO print_media
(product_id, ad_id, ad_composite, ad_sourcetext, ad_graphic)
    select 2, ad_id, ad_composite, ad_sourcetext, ad_graphic from
print_media;

-- Update the 2nd row to point to a different file
UPDATE print_media SET ad_graphic = BFILENAME('MYDIR','file2.txt')  WHERE
product_id =2;

-- Insert a 3rd row with invalid file name
INSERT INTO print_media
(product_id, ad_id, ad_composite, ad_sourcetext, ad_graphic)
VALUES
(3, 3, empty_blob(), empty_clob(),
BFILENAME('MYDIR','file_does_not_exist.txt'));

-- Insert a NULL for BFILE
INSERT INTO print_media
(product_id, ad_id, ad_composite, ad_sourcetext, ad_graphic)
VALUES
(4, 4, empty_blob(), empty_clob(), NULL);

-- Inserting in PLSQL using a BFILE variable
DECLARE
    f BFILE;
BEGIN
    f := BFILENAME('MYDIR','file5.txt');
    INSERT INTO print_media (product_id, ad_id, ad_composite, ad_sourcetext,
ad_graphic)
    VALUES (5, 5, NULL, NULL, f);
END;
/
SELECT product_id, ad_id, ad_graphic FROM print_media ORDER BY 1,2;
```

Example 5-2 Inserting BFILEs in OCI

```
STATIC TEXT *insstmt = "INSERT INTO print_media (product_id, ad_id,
ad_graphic) VALUES (:1, :1, :2)";
sword insert_bfile()
{
  OCILobLocator *f = (OCILobLocator *)0;

  OCIStmt       *stmthp;
  OCIBind       *bndp1  = (OCIBind *)   0;
  OCIBind       *bndp2  = (OCIBind *)   0;

  ub4            id;

  CHECK_ERROR (OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                              OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

  /*************** Allocate descriptor **********************/
  CHECK_ERROR (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &f,
                                  (ub4)OCI_DTYPE_FILE, (size_t) 0,
                                  (dvoid **) 0));
```

```
  /********** Execute insstmt to insert f ********************/
  id = 6;
  CHECK_ERROR (OCILobFileSetName(envhp, errhp, &f,
                                 (text*)"MYDIR", sizeof("MYDIR") -1,
                                 (text*)"file6.txt",
                                 sizeof("file6.txt") -1));

  CHECK_ERROR (OCIStmtPrepare(stmthp, errhp, insstmt,
                              (ub4) strlen((char *) insstmt),
                              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  CHECK_ERROR (OCIBindByPos(stmthp, &bndp1, errhp, (ub4) 1, (dvoid *) &id,
                              (sb4) sizeof(id), SQLT_INT, (dvoid *) 0, (ub2
*) 0,
                              (ub2 *)0, (ub4) 0, (ub4*) 0, (ub4)
OCI_DEFAULT));

  CHECK_ERROR (OCIBindByPos(stmthp, &bndp2, errhp, (ub4) 2, (dvoid *) &f4,
                              (sb4) -1, SQLT_BFILE, (dvoid *) 0, (ub2 *) 0,
                              (ub2 *)0, (ub4) 0, (ub4*) 0, (ub4)
OCI_DEFAULT));

  CHECK_ERROR (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                              (CONST OCISnapshot *) NULL, (OCISnapshot *)
NULL,
                             OCI_DEFAULT));

  /********** Execute insstmt to insert NULL ********************/
  id = 7;
  CHECK_ERROR (OCIStmtPrepare(stmthp, errhp, insstmt,
                              (ub4) strlen((char *) insstmt),
                              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  CHECK_ERROR (OCIBindByPos(stmthp, &bndp1, errhp, (ub4) 1, (dvoid *) &id,
                              (sb4) sizeof(id), SQLT_INT, (dvoid *) 0, (ub2
*) 0,
                              (ub2 *)0, (ub4) 0, (ub4*) 0, (ub4)
OCI_DEFAULT));

  CHECK_ERROR (OCIBindByPos(stmthp, &bndp2, errhp, (ub4) 2, (dvoid *) NULL,
                              (sb4) -1, SQLT_BFILE, (dvoid *) 0, (ub2 *) 0,
                              (ub2 *)0, (ub4) 0, (ub4*) 0, (ub4)
OCI_DEFAULT));

  CHECK_ERROR (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                              (CONST OCISnapshot *) NULL, (OCISnapshot *)
NULL,
                             OCI_DEFAULT));

}
```

5.3 BFILE APIs

This section discusses the different operations supported through BFILEs.

Once you initialize a BFILE variable either by using the BFILENAME function or an equivalent API, or by using a SELECT operation on a BFILE column, you can perform read operations on the BFILE using APIs such as DBMS_LOB. Note that BFILE is a read-only data type. So, you cannot update or delete the operating system files, accessed using BFILEs, through the BFILE APIs.
The operations performed on BFILEs are divided into following categories:

Table 5-1 Operations on BFILEs

Category	Operation	Example function /procedure in DBMS_LOB package
Sanity Checking	Check if the BFILE exists on the server	FILEEXISITS
	Get the DIRECTORY object name and file name	FILEGETNAME
	Set the name of a BFILE in a locator without checking if the directory or file exists	BFILENAME
Open / Close	Open a file	OPEN
	Check if the file was opened using the input BFILE locators	ISOPEN
	Close the file	CLOSE
	Close all previously opened files	FILECLOSEALL
Read Operations	Get the length of the BFILE	GETLENGTH
	Read data from the BFILE starting at the specified offset	READ
	Return part of the BFILE value starting at the specified offset using SUBSTR	SUBSTR
	Return the matching position of a pattern in a BFILE using INSTR	INSTR
Operations involving multiple locators	Assign BFILE locator src to BFILE locator dst	dst := src
	Load BFILE data into a LOB	LOADCLOBFROMFILE, LOADBLOBFROMFILE
	Compare all or part of the value of two BFILEs	COMPARE

- Sanity Checking
  Sanity Checking functions on BFILEs enable you to retrieve information about the BFILEs.
- Opening and Closing a BFILE
  You must OPEN a BFILE before performing any operations on it, and CLOSE it before you terminate your program.
- Reading from a BFILE
  You can perform many different read operations on the BFILE data, including reading its length, reading part of the data, or reading the whole data.

- Working with Multiple BFILE Locators
  Some `BFILE` operations accept two locators, at least one of which is a `BFILE` locator. For the assignment and the comparison operations involving BFILES, both the locators must be of `BFILE` type.

5.3.1 Sanity Checking

Sanity Checking functions on BFILEs enable you to retrieve information about the BFILEs.

Recall that the `BFILENAME()` and `OCILobFileSetName()` functions do not verify that the directory and path name you specify actually exist. You can use the sanity checking functions to verify that a `BFILE` exists and to extract the directory and file names from a `BFILE` locator.

5.3.2 Opening and Closing a BFILE

You must `OPEN` a `BFILE` before performing any operations on it, and `CLOSE` it before you terminate your program.

A `BFILE` locator operates like a file descriptor available as part of the standard input/output library of most conventional programming languages. This implies that once you define and initialize a `BFILE` locator, and open the file pointed to by this locator, all subsequent operations until the closure of the file must be done from within the same program block using the locator or local copies of it. The `BFILE` locator variable can be used as a parameter to other procedures, member methods, or external function callouts. However, it is recommended that you open and close a file from the same program block at the same nesting level.

You must close all the open `BFILE` instances even in cases, where an exception or unexpected termination of your application occurs. In these cases, if a `BFILE` instance is not closed, then it is still considered open by the database. Ensure that your exception handling strategy does not allow `BFILE` instances to remain open in these situations.

You can close all open `BFILE`s together by using a procedure like `DBMS_LOB.FILECLOSEALL` or `OCILobFileCloseAll()`.

5.3.3 Reading from a BFILE

You can perform many different read operations on the `BFILE` data, including reading its length, reading part of the data, or reading the whole data.

When reading from a large `BFILE`, you can use the streaming read mode in JDBC or OCI. In JDBC, you can achieve this by using the `getBinaryStream()` method. In OCI, you can achieve it in the way as described in the following section.

Streaming Read in OCI

The most efficient way to read large amounts of `BFILE` data is by using the `OCILobRead2()` function with the streaming mechanism enabled, and using polling or callback. To do so, specify the starting point of the read using the `offset` parameter as follows:

```
ub8  char_amt =  0;
ub8  byte_amt =  0;
ub4  offset = 1000;

OCILobRead2(svchp, errhp, locp, &byte_amt, &char_amt, offset, bufp, bufl,
            OCI_ONE_PIECE, 0, 0, 0, 0);
```

When using polling mode, be sure to look at the value of the byte_amt parameter after each OCILobRead2() call to see how many bytes were read into the buffer, because the buffer may not be entirely full.

When using callbacks, the lenp parameter, which is input to the callback, indicates how many bytes are filled in the buffer. Be sure to check the lenp parameter during your callback processing because the entire buffer may not be filled with data.

Amount Parameter

- When calling the DBMS_LOB.READ API, the size of the amount parameter can be larger than the size of the data. However, this parameter should be less than or equal to the size of the buffer. In PL/SQL, the buffer size is limited to 32K.
- When calling the OCILobRead2() function, you can pass a value of UB8MAXVAL for the byte_amt parameter to read to the end of the BFILE.

5.3.4 Working with Multiple BFILE Locators

Some BFILE operations accept two locators, at least one of which is a BFILE locator. For the assignment and the comparison operations involving BFILES, both the locators must be of BFILE type.

Loading a LOB with BFILE data involves special considerations that we will discuss in the following sections:

Loading a LOB with BFILE Data

In PLSQL, the DBMS_LOB.LOADFROMFILE procedure is deprecated in favor of DBMS_LOB.LOADBLOBFROMFILE and DBMS_LOB.LOADCLOBFROMFILE. Specifically, when you use DBMS_LOB.LOADCLOBFROMFILE procedure to load a CLOB or NCLOB instance, it will perform the character set conversions.

Specifying the Amount of BFILE Data to Load

The value you pass for the amount parameter to functions listed in the table below must be one of the following:

- An amount less than or equal to the actual size (in bytes) of the BFILE you are loading.
- The maximum allowable LOB size (in bytes). Passing this value, loads the entire BFILE. You can use this technique to load the entire BFILE without determining the size of the BFILE before loading. To get the maximum allowable LOB size, use the technique described in the following table:

Table 5-2 Maximum LOB Size for Load from File Operations

Environment	Function	To pass maximum LOB size, get value of:
DBMS_LOB	DBMS_LOB.LOADBLOBFROMFILE	DBMS_LOB.LOBMAXSIZE
DBMS_LOB	DBMS_LOB.LOADCLOBFROMFILE	DBMS_LOB.LOBMAXSIZE
OCI	OCILobLoadFromFile2()	UB8MAXVAL
OCI	OCILobLoadFromFile() (For LOBs less than 4 gigabytes in size.)	UB4MAXVAL

Loading a BLOB with BFILE Data

The DBMS_LOB.LOADBLOBFROMFILE procedure loads a BLOB with data from a BFILE. It can be used to load data into any persistent or temporary BLOB instance. This procedure returns the new source and the destination offsets of the BLOB, which you can then pass into subsequent calls, when used in a loop.

Loading a CLOB with BFILE Data

The DBMS_LOB.LOADCLOBFROMFILE procedure loads a CLOB or NCLOB with character data from a BFILE. It can be used to load data into a persistent or temporary CLOB or NCLOB instance. You can specify the character set ID of the BFILE when calling this procedure and ensure that the character set is properly converted from the BFILE data character set to the destination CLOB or NCLOB character set. This procedure returns the new source and destination offsets of the CLOB or NCLOB, which you can then passe into subsequent calls, when used in a loop.

The following example illustrates:

- How to use default csid(0).
- How to load the entire file without calling getlength for the BFILE.
- How to find out the actual amount loaded using return offsets.

This example assumes that ad_source is a BFILE in UTF8 character set format and the database character set is UTF8.

```
CREATE OR REPLACE PROCEDURE loadCLOB1_proc (dst_loc IN OUT CLOB) IS
  src_loc     BFILE := BFILENAME('MEDIA_DIR','monitor_3060.txt') ;
  amt         NUMBER := DBMS_LOB.LOBMAXSIZE;
  src_offset  NUMBER := 1 ;
  dst_offset  NUMBER := 1 ;
  lang_ctx    NUMBER := DBMS_LOB.DEFAULT_LANG_CTX;
  warning     NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('------------ LOB LOADCLOBFORMFILE EXAMPLE
------------');
  DBMS_LOB.FILEOPEN(src_loc, DBMS_LOB.FILE_READONLY);

  /* The default_csid can be used when the BFILE encoding is in the same
charset
   * as the destination CLOB/NCLOB charset
   */
   DBMS_LOB.LOADCLOBFROMFILE(dst_loc,src_loc, amt, dst_offset,
src_offset,
       DBMS_LOB.DEFAULT_CSID, lang_ctx,warning) ;
  DBMS_OUTPUT.PUT_LINE(' Amount specified ' || amt ) ;
  DBMS_OUTPUT.PUT_LINE(' Number of bytes read from source: ' ||
(src_offset-1));
  DBMS_OUTPUT.PUT_LINE(' Number of characters written to destination: ' ||
(dst_offset-1) );
  IF (warning = DBMS_LOB.WARN_INCONVERTIBLE_CHAR)
  THEN
    DBMS_OUTPUT.PUT_LINE('Warning: Inconvertible character');
  END IF;
  DBMS_LOB.FILECLOSEALL() ;
END;
/
```

The following example illustrates:

- How to get the character set ID from the character set name using the `NLS_CHARSET_ID` function.
- How to load a stream of data from a single `BFILE` into different LOBs using the returned offset value and the language context `lang_ctx`.
- How to read a warning message

This example assumes that `ad_file_ext_01` is a `BFILE` in `JA16TSTSET` format and the database national character set is `AL16UTF16`.

```
CREATE OR REPLACE PROCEDURE loadCLOB2_proc (dst_loc1 IN OUT NCLOB,dst_loc2 IN
OUT NCLOB) IS
  src_loc     BFILE := BFILENAME('MEDIA_DIR','monitor_3060.txt');
  amt         NUMBER := 100;
  src_offset  NUMBER := 1;
  dst_offset  NUMBER := 1;
  src_osin    NUMBER;
  cs_id       NUMBER := NLS_CHARSET_ID('JA16TSTSET'); /* 998 */
  lang_ctx    NUMBER := dbms_lob.default_lang_ctx;
  warning     NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('------------ LOB LOADCLOBFORMFILE EXAMPLE
------------');
  DBMS_LOB.FILEOPEN(src_loc, DBMS_LOB.FILE_READONLY);
  DBMS_OUTPUT.PUT_LINE(' BFILE csid is ' || cs_id) ;

  /* Load the first 1KB of the BFILE into dst_loc1 */

  DBMS_OUTPUT.PUT_LINE(' ----------------------------' ) ;
  DBMS_OUTPUT.PUT_LINE('   First load  ' ) ;
  DBMS_OUTPUT.PUT_LINE(' ----------------------------' ) ;

  DBMS_LOB.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
      cs_id, lang_ctx, warning);

  /* the number bytes read may or may not be 1k */

  DBMS_OUTPUT.PUT_LINE(' Amount specified ' || amt ) ;
  DBMS_OUTPUT.PUT_LINE(' Number of bytes read from source: ' ||
      (src_offset-1));
  DBMS_OUTPUT.PUT_LINE(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    DBMS_OUTPUT.PUT_LINE('Warning: Inconvertible character');
  end if;

  /* load the next 1KB of the BFILE into the dst_loc2 */

  DBMS_OUTPUT.PUT_LINE(' ----------------------------' ) ;
  DBMS_OUTPUT.PUT_LINE('   Second load  ' ) ;
  DBMS_OUTPUT.PUT_LINE(' ----------------------------' ) ;

  /* Notice we are using the src_offset and lang_ctx returned from the
previous
   * load. We do not use value 1001 as the src_offset here because sometimes
```

```
the
   * actual amount read may not be the same as the amount specified.
   */

  src_osin := src_offset;
  dst_offset := 1;
  DBMS_LOB.LOADCLOBFROMFILE(dst_loc2, src_loc, amt, dst_offset, src_offset,
      cs_id, lang_ctx, warning);
  DBMS_OUTPUT.PUT_LINE(' Number of bytes read from source: ' ||
      (src_offset-src_osin) );
  DBMS_OUTPUT.PUT_LINE(' Number of characters written to destination: ' ||
      (dst_offset-1) );
  if (warning = DBMS_LOB.WARN_INCONVERTIBLE_CHAR)
  then
    DBMS_OUTPUT.PUT_LINE('Warning: Inconvertible character');
  end if;
  DBMS_LOB.FILECLOSEALL() ;

END;
/
```

5.4 BFILE APIs in Different Programmatic Interfaces

This section lists all the APIs from different Programmatic Interfaces supported by Oracle Database.

> **Note:**
>
> The PL/SQL `DBMS_LOB` package provides a rich set of operations on `BFILE`s. If you are using a different Programmatic Interface where some of these operations are not provided, then call the corresponding PL/SQL `DBMS_LOB` procedure or function.

- PL/SQL APIs for BFILEs
  This section describes the PL/SQL APIs that you can use with BFILEs.
- JDBC API for BFILEs
  This section describes the JDBC APIs that you can use to work with BFILEs.
- OCI API for BFILEs
  This section describes the OCI APIs that you can use with BFILEs.
- ODP.NET API for BFILEs
  This section describes the ODP.NET APIs that you can use with BFILEs.
- OCCI API for BFILEs
  This section describes the OCCI APIs that you can use with `BFILE`s.
- Pro*C/C++ and Pro*COBOL API for BFILEs
  This section describes Pro*C/C++ and Pro*COBOL APIs APIs you can use for BFILEs.

> **See Also:**
>
> Comparing the LOB Interfaces

5.4.1 PL/SQL APIs for BFILEs

This section describes the PL/SQL APIs that you can use with BFILEs.

> **See Also:**
>
> DBMS_LOB

Table 5-3 DBMS_LOB functions and procedures for BFILEs

Category	Function/ Procedure	Description
Sanity Checking	FILEEXISTS	Checks if the BFILE exists on the server
	FILEGETNAME	Gets the DIRECTORY object name and file name
	BFILENAME	Sets the name of a BFILE in a locator without checking if the directory or file exists
Open/Close	OPEN, FILEOPEN	Opens a file. Use OPEN instead of FILEOPEN.
	ISOPEN, FILEISOPEN	Checks if the file was opened using the input BFILE locators. Use ISOPEN instead of FILEISOPEN.
	CLOSE, FILECLOSE	Closes the file. Use CLOSE instead of FILECLOSE.
	FILECLOSEALL	Closes all previously opened files.
Read Operations	GETLENGTH	Gets the length of the BFILE
	READ	Reads data from the BFILE starting at the specified offset.
	SUBSTR	Returns part of the BFILE value starting at the specified offset.
	INSTR	Returns the matching position of the nth occurrence of the pattern in the BFILE.
Operations involving multiple locators	:= (operator)	Assigns a BFILE locator to another
	LOADCLOBFROMFILE	Loads character data from a file into a LOB
	LOADBLOBFROMFILE	Loads binary data from a file into a LOB
	LOADFROMFILE	Loads BFILE data into a LOB (deprecated)
	COMPARE	Compares the value of two BFILEs.

Example 5-3 PL/SQL API for BFILEs

```
declare
  f           BFILE;
  f2          BFILE;
  b           BLOB;
  c           CLOB;
  dest_offset NUMBER;
  src_offset  NUMBER;
  lang        NUMBER;
  warn        NUMBER;
  buffer      RAW(128);
  amt         NUMBER;
  len         NUMBER;
  pos         NUMBER;
  filename    VARCHAR2(128);
  dirname     VARCHAR2(128);
BEGIN

   /* Select out a BFILE locator */
  SELECT ad_graphic INTO f FROM print_media WHERE product_id = 1 AND ad_id =
1;

  /*---------------------------------------------------------------------*/
  /*---------------------- Sanity Checking ------------------------------*/
  /*---------------------------------------------------------------------*/

  /*-------------- Determining Whether a BFILE Exists ----------------*/
  if DBMS_LOB.FILEEXISTS(f) = 1 then
    DBMS_OUTPUT.PUT_LINE('F exists!');
  else
    DBMS_OUTPUT.PUT_LINE('F does not exist :(');
    return;
  end if;

  /*------ Getting Directory Object Name and File Name of a BFILE ----*/
  DBMS_LOB.FILEGETNAME(f, dirname, filename);
  DBMS_OUTPUT.PUT_LINE('F: directory: '|| dirname ||' filename: '|| filename);

  /*---------------------------------------------------------------------*/
  /*----------------------- Open/Close ----------------------------------*/
  /*---------------------------------------------------------------------*/

  /*----------------------- Opening a BFILE --------------------------*/
  DBMS_LOB.OPEN(f, DBMS_LOB.LOB_READONLY);

  /*-------------- Determining Whether a BFILE Is Open ---------------*/
  if DBMS_LOB.ISOPEN(f) = 1 then
    DBMS_OUTPUT.PUT_LINE('F is open!');
  else
    DBMS_OUTPUT.PUT_LINE('F is not open :(');
  end if;

  /*----------------------- Closing a BFILE --------------------------*/
  DBMS_LOB.CLOSE(f);
```

```
  /*------------ Closing All Open BFILEs with FILECLOSEALL -----------*/
  DBMS_LOB.FILECLOSEALL;

  /*-----------------------------------------------------------------*/
  /*-------------------- BFILE operations ---------------------------*/
  /*-----------------------------------------------------------------*/

  DBMS_LOB.OPEN(f, dbms_lob.lob_readonly);

  /*----------------- Getting the Length of a BFILE ------------------*/
  len := DBMS_LOB.GETLENGTH(f);
  DBMS_OUTPUT.PUT_LINE('dbms_lob.getlength: '||len);

  /*----------------------- Reading BFILE Data ----------------------*/
  amt := 15;
  DBMS_LOB.READ(f, amt, 1, buffer);
  DBMS_OUTPUT.PUT_LINE('dbms_lob.read: '||UTL_RAW.CAST_TO_VARCHAR2(buffer));

  /*--------- Reading a Portion of BFILE Data Using SUBSTR -----------*/
  buffer := DBMS_LOB.SUBSTR(f, 15, 3);
  DBMS_OUTPUT.PUT_LINE('dbms_lob.substr: '||UTL_RAW.CAST_TO_VARCHAR2(buffer));

  /*------ Checking If a Pattern Exists in a BFILE Using INSTR -------*/
  pos :=  DBMS_LOB.INSTR(f, utl_raw.cast_to_raw('BFILE'), 1, 1);
  if pos != 0 then
    DBMS_OUTPUT.PUT_LINE('dbms_lob.instr: "BFILE" word exists in position '
|| pos);
  else
    DBMS_OUTPUT.PUT_LINE('dbms_lob.instr: "BFILE" word does not exist in
file');
  end if;

  /*-----------------------------------------------------------------*/
  /*------------- Operations involving 2 locators -------------------*/
  /*-----------------------------------------------------------------*/

  /*----------------- Assigning a BFILE Locator ----------------------*/
  f2 := f; -- where f2 is also a bfile variable

  amt := 15;
  DBMS_LOB.READ(f2, amt, 1, buffer);
  DBMS_OUTPUT.PUT_LINE('assign: dbms_lob.read: '||
UTL_RAW.CAST_TO_VARCHAR2(buffer));

  /*--------------- Loading a LOB with BFILE Data --------------------*/
  /* Select out BLOB and CLOB for update so we can write to them */
  select ad_composite, ad_sourcetext into b, c
  from print_media where product_id = 1 and ad_id = 1 for update;

  /* Load BLOB from BFILE */
  dest_offset := 1;
  src_offset  := 1;

  DBMS_LOB.LOADBLOBFROMFILE(b, f, dbms_lob.lobmaxsize, dest_offset,
src_offset);
```

```
  /* Load CLOB from BFILE, for this operation is necessary to know the charset
   * id of BFILE to read it correctly */
  dest_offset := 1;
  src_offset  := 1;
  lang        := 0;
  /* Specifying the amount as DBMS_LOB.LOBMAXSIZE to copy till end of file */
  DBMS_LOB.LOADCLOBFROMFILE(c, f, DBMS_LOB.LOBMAXSIZE, dest_offset,
src_offset,
                            NLS_CHARSET_ID('utf8'), lang, warn);

  /*-------------- Comparing All or Parts of Two BFILES -------------*/
  SELECT ad_graphic INTO f2 FROM print_media WHERE product_id = 2 AND ad_id =
1;
  DBMS_LOB.OPEN(f2, dbms_lob.lob_readonly);
  if DBMS_LOB.COMPARE(f, f2, 10, 1, 1) = 0 then
    DBMS_OUTPUT.PUT_LINE('dbms_lob.compare: They are equals!!');
  else
    DBMS_OUTPUT.PUT_LINE('dbms_lob.compare: They are not equals :(');
  end if;

  -- Close just f
  DBMS_LOB.CLOSE(f);

  -- Close the rest of bfiles opended
  DBMS_LOB.FILECLOSEALL;

END;
/
```

5.4.2 JDBC API for BFILEs

This section describes the JDBC APIs that you can use to work with BFILEs.

In JDBC, the `oracle.jdbc.OracleBfile` interface provides methods for performing operations on `BFILE` data in the database. It encapsulates the `BFILE` locators, so you do not deal with locators, but instead use methods and properties provided to perform operations and get state information.

To retrieve the locator for the most current row, you must call the `getBFILE()` method on the `OracleResultSet` each time a move operation is made, depending on whether the instance is a `BFILE`.

> **See Also:**
>
> Working with LOBs and BFILEs

Table 5-4 JDBC APIs for BFILEs

Category	Function/ Procedure	Description
Sanity Checking	`boolean fileExists()`	Checks if the `BFILE` exists on the server

Table 5-4 (Cont.) JDBC APIs for BFILEs

Category	Function/ Procedure	Description
	`public java.lang.String getName()`	Gets the file name
	`public java.lang.String getDirAlias()`	Gets the `DIRECTORY` object name
Open/Close	`public void openFile()`	Opens a file.
	`public boolean isFileOpen()`	Checks if the file was opened using the input `BFILE` locators. .
	`public void closeFile()`	Closes the file. Use `CLOSE` instead of `FILECLOSE`.
Read Operations	`long length()`	Gets the length of the `BFILE`
	`public java.io.InputStream getBinaryStream()`	Reads the `BFILE` as a binary stream.
	`byte[] getBytes(long, int)`	Gets the contents of the `BFILE` as an array of bytes, given an offset
	`int getBytes(long, int, byte[])`	Reads a subset of the `BFILE` into a byte array
	`long position(oracle.jdbc.OracleBfile, long)`	Finds the first appearance of the given `BFILE` contents within the LOB, from the given offset.
	`long position(byte[], long)`	Finds the first appearance of the given byte array within the `BFILE`, from the given offset
Operations involving multiple locators	[use equal sign]	Assigns a `BFILE` locator to another

Example 5-4 JDBC API for BFILEs

```
static void run_query() throws Exception {

  try(
      OracleConnection con = getConnection();
      Statement       stmt = con.createStatement();
      ){

    ResultSet   rs       = null;

    OracleBfile f        = null;
    OracleBfile f2       = null;
    OracleBfile f3       = null;

    InputStream in       = null;
    String      output   = null;
    byte        buffer[] = new byte[15];
    long        pos;

    String      filename = null;
    String      dirname  = null;
    long        len      = 0;
```

```
    rs = stmt.executeQuery("select ad_graphic from print_media where
product_id = 1");
    rs.next();
    f = (OracleBfile)((OracleResultSet)rs).getBfile(1);
    rs.close();

    rs = stmt.executeQuery("select ad_graphic from print_media where
product_id = 2");
    rs.next();
    f2 = (OracleBfile)((OracleResultSet)rs).getBfile(1);
    rs.close();

    stmt.close();

    /*-------------------------------------------------------------------*/
    /*---------------------- Sanity Checking ----------------------------*/
    /*-------------------------------------------------------------------*/

    /*-------------- Determining Whether a BFILE Exists ----------------*/
    if (f.fileExists())
      System.out.println("F exists!");
    else
      System.out.println("F does not exist :(");

    /*------ Getting Directory Object Name and File Name of a BFILE ----*/
    dirname  = f.getDirAlias();
    filename = f.getName();

    System.out.println("Directory: " + dirname + " Filename: " + filename);

    /*-------------------------------------------------------------------*/
    /*----------------------- Open/Close --------------------------------*/
    /*-------------------------------------------------------------------*/

    /*----------------------- Opening a BFILE ---------------------------*/
    f.open(LargeObjectAccessMode.MODE_READONLY);

    /*-------------- Determining Whether a BFILE Is Open ---------------*/
    if (f.isOpen())
      System.out.println("F is open!");
    else
      System.out.println("F is not open :(");

    /*----------------------- Closing a BFILE ---------------------------*/
    f.close();

    /*-------------------------------------------------------------------*/
    /*--------------------- BFILE operations ----------------------------*/
    /*-------------------------------------------------------------------*/

    f.open(LargeObjectAccessMode.MODE_READONLY);

    /*----------------- Getting the Length of a BFILE -------------------*/
    len = f.length();
    System.out.println("F Length: "+len);
```

```
    /*------------------------ Reading BFILE Data ----------------------*/
    in = f.getBinaryStream();
    in.read(buffer);
    in.close();

    output = new String(buffer);
    System.out.println("Buffer: " + output);

    /*---- Checking If a Pattern Exists in a BFILE Using POSITION ------*/
    pos = f.position("BFILE".getBytes(), 1);

    if (pos != -1)
      System.out.println("\"BFILE\" word exists in position: " + pos);
    else
      System.out.println("\"BFILE\" word doesn't exist :( " );

    /*-------------------------------------------------------------------*/
    /*------------- Operations involing 2 locators ----------------------*/
    /*-------------------------------------------------------------------*/

    /*----------------- Assigning a BFILE Locator ----------------------*/
    f3 = f;

    in = f3.getBinaryStream();
    in.read(buffer);
    in.close();

    output = new String(buffer);
    System.out.println("assign: Buffer: " + output);

    /*-------------- Comparing All or Parts of Two BFILES -------------*/
    f2.open(LargeObjectAccessMode.MODE_READONLY);
    pos = f.position(f2, 1);

    if (pos != -1)
      System.out.println("f2 exists in position " + pos);
    else
      System.out.println("f2 doesn't exist in position");

    f.close();
    f2.close();
    f3.close();
  }
}
```

5.4.3 OCI API for BFILEs

This section describes the OCI APIs that you can use with BFILEs.

> **See Also:**
>
> LOB and BFILE Operations

Table 5-5 OCI APIs for BFILEs

Category	Function/ Procedure	Description
Sanity Checking	`OCILobFileExists()`	Checks if the `BFILE` exists on the server
	`OCILobFileGetName()`	Gets the `DIRECTORY` object name and the file name
	`OCILobFileSetName()`	Sets the name of a `BFILE` in a locator without checking if the directory or file exists
	`OCILobLocatorIsInit()`	Checks whether a LOB Locator is initialized
Open/Close	`OCILobOpen()` and `OCILobFileOpen()`	Opens a file. Use `OciLobOpen()` instead of `OCILobFileOpen()`.
	`OCILobIsOpen()` and `OCILobFileIsOpen()`	Checks if the file was opened using the input `BFILE` locators. Use `OCILobIsOpen()` instead of `OciLobFileIsOpen()`.
	`OCILobClose()` and `OCILobFileClose()`	Closes the file. Use `OciLobClose()` instead of `OciLobFileClose()`.
	`OCILobFileCloseAll()`	Closes all previously opened files.
Read Operations	`OCILobGetLength2()`	Gets the length of the `BFILE`
	`OCILobRead2()`	Reads data from the `BFILE` starting at the specified offset.
	`OCILobArrayRead()`	Reads data using multiple locators in one round trip.
Operations involving multiple locators	`OCILobLocatorAssign()`	Assigns a `BFILE` locator to another
	`OCILobLoadFromFile2()`	Loads `BFILE` data from a file into a LOB

Example 5-5 OCI API for BFILEs

```
static text *selstmt = (text *) "select ad_graphic, ad_composite,
ad_sourcetext from print_media where product_id = 1 and ad_id = 1 for update"
sword run_query()
{
  OCILobLocator *f = (OCILobLocator *)0;
  OCILobLocator *f2 = (OCILobLocator *)0;

  OCILobLocator *b = (OCILobLocator *)0;
  OCILobLocator *c = (OCILobLocator *)0;

  OCIStmt       *stmthp;
  OCIDefine     *defn1p = (OCIDefine *) 0;
  OCIDefine     *defn2p = (OCIDefine *) 0;
  OCIDefine     *defn3p = (OCIDefine *) 0;

  ub4            bfilelen;
  ub1            lbuf[128];
```

```
  ub8             amt = 15;
  boolean         flag = FALSE;
  ub4             id = 10;

  text            filename[128];
  ub2             filename_len;
  text            dirname[128];
  ub2             dirname_len;

  CHECK_ERROR (OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                              OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

  /************** Allocate descriptors ***********************/
  CHECK_ERROR (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &f,
                                  (ub4)OCI_DTYPE_FILE, (size_t) 0,
                                  (dvoid **) 0));

  CHECK_ERROR (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &f2,
                                 (ub4)OCI_DTYPE_FILE, (size_t) 0,
                                 (dvoid **) 0));

  CHECK_ERROR (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &b,
                                  (ub4)OCI_DTYPE_LOB, (size_t) 0,
                                  (dvoid **) 0));

  CHECK_ERROR (OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &c,
                                  (ub4)OCI_DTYPE_LOB, (size_t) 0,
                                  (dvoid **) 0));

  /********** Execute selstmt to get f, b, c **********************/
  CHECK_ERROR (OCIStmtPrepare(stmthp, errhp, selstmt,
                              (ub4) strlen((char *) selstmt),
                              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  CHECK_ERROR (OCIDefineByPos(stmthp, &defn1p, errhp, (ub4) 1, (dvoid *) &f,
                              (sb4) -1, SQLT_BFILE, (dvoid *) 0, (ub2 *) 0,
                              (ub2 *)0, (ub4) OCI_DEFAULT));
  CHECK_ERROR (OCIDefineByPos(stmthp, &defn2p, errhp, (ub4) 2, (dvoid *) &b,
                              (sb4) -1, SQLT_BLOB, (dvoid *) 0, (ub2 *) 0,
                              (ub2 *)0, (ub4) OCI_DEFAULT));
  CHECK_ERROR (OCIDefineByPos(stmthp, &defn3p, errhp, (ub4) 3, (dvoid *) &c,
                              (sb4) -1, SQLT_CLOB, (dvoid *) 0, (ub2 *) 0,
                              (ub2 *)0, (ub4) OCI_DEFAULT));

  CHECK_ERROR (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                              (CONST OCISnapshot *) NULL, (OCISnapshot *)
NULL,
                              OCI_DEFAULT));

  /*---------------------------------------------------------------*/
  /*---------------------- Sanity Checking ------------------------*/
  /*---------------------------------------------------------------*/

  /*-------------- Determining Whether a BFILE Exists ----------------*/
  CHECK_ERROR (OCILobFileExists(svchp, errhp, f, &flag));
  printf("OCILobFileExists: %s\n", (flag)?"TRUE":"FALSE");
```

```
  /*------ Getting Directory Object Name and File Name of a BFILE ----*/
  CHECK_ERROR (OCILobFileGetName(envhp, errhp, f, (text*)dirname,
&dirname_len,
                                 (text*)filename, &filename_len));
  printf("OCILobFileGetName: Directory: %.*s Filaname: %.*s \n",
         dirname_len, dirname, filename_len, filename);

  /*-------------------------------------------------------------------*/
  /*----------------------- Open/Close --------------------------------*/
  /*-------------------------------------------------------------------*/

  /*----------------------- Opening a BFILE ---------------------------*/
  CHECK_ERROR (OCILobFileOpen(svchp, errhp, f, OCI_FILE_READONLY));
  printf("OCILobFileOpen: Works\n");

  /*-------------- Determining Whether a BFILE Is Open ---------------*/
  CHECK_ERROR (OCILobFileIsOpen(svchp, errhp, f, &flag));
  printf("OCILobFileIsOpen: %s\n", (flag)?"TRUE":"FALSE");

  /*----------------------- Closing a BFILE ---------------------------*/
  CHECK_ERROR (OCILobFileClose (svchp, errhp, f));

  /*------------ Closing All Open BFILEs with FILECLOSEALL -----------*/
  CHECK_ERROR (OCILobFileCloseAll(svchp, errhp));

  /*-------------------------------------------------------------------*/
  /*-------------------- BFILE operations -----------------------------*/
  /*-------------------------------------------------------------------*/

  CHECK_ERROR (OCILobFileOpen(svchp, errhp, f, OCI_FILE_READONLY));
  printf("OCILobFileOpen: Works\n");

  /*----------------- Getting the Length of a BFILE ------------------*/
  CHECK_ERROR (OCILobGetLength(svchp, errhp, b, &bfilelen));
  printf("OCILobGetLength: loblen: %d \n", bfilelen);

  /*------------------------ Reading BFILE Data -----------------------*/
  CHECK_ERROR (OCILobRead2(svchp, errhp, f, &amt,
                           NULL, (oraub8)1, lbuf,
                           (oraub8)sizeof(lbuf), OCI_ONE_PIECE ,(dvoid*)0,
                           NULL, (ub2)0, (ub1)SQLCS_IMPLICIT));
  printf("OCILobRead2: buf: %.*s amt: %lu\n", amt, lbuf, amt);

  /*-------------------------------------------------------------------*/
  /*------------- Operations involing 2 locators ----------------------*/
  /*-------------------------------------------------------------------*/

  /*----------------- Assigning a BFILE Locator ----------------------*/
  CHECK_ERROR (OCILobLocatorAssign(svchp, errhp, f, &f2));
  printf("OCILobLocatorAssign: Works! \n");

  amt = 15;
  CHECK_ERROR (OCILobRead2(svchp, errhp, f2, &amt,
                          NULL, (oraub8)1, lbuf,
                          (oraub8)sizeof(lbuf), OCI_ONE_PIECE ,(dvoid*)0,
```

```
                          NULL, (ub2)0, (ub1)SQLCS_IMPLICIT));
  printf("OCILobLocatorAssign: OCILobRead2: buf: %.*s amt: %lu\n", amt, lbuf,
amt);

  /*--------------- Loading a LOB with BFILE Data --------------------*/
  /* Load BLOB from BFILE. Specify amount = UB8MAXVAL to copy till end of
bfile */
  CHECK_ERROR (OCILobLoadFromFile2(svchp, errhp, b, f, UB8MAXVAL, 1,1));
  printf("OCILobLoadFromFile2: BLOB case Works\n");

  /* Load CLOB from BFILE. Specify amount = UB8MAXVAL to copy till end of
bfile.
   * Note that there is no character set conversion here. */
  CHECK_ERROR (OCILobLoadFromFile2(svchp, errhp, c, f, UB8MAXVAL, 1,1));
  printf("OCILobLoadFromFile2: CLOB case Works\n");

  /*  Close just f */
  CHECK_ERROR (OCILobFileClose (svchp, errhp, f));

  /* Close the rest of bfiles opened */
  CHECK_ERROR (OCILobFileCloseAll(svchp, errhp));

  OCIDescriptorFree((dvoid *) b, (ub4) SQLT_BLOB);
  OCIDescriptorFree((dvoid *) c, (ub4) SQLT_CLOB);
  OCIDescriptorFree((dvoid *) f, (ub4) SQLT_BFILE);
  OCIDescriptorFree((dvoid *) f2, (ub4) SQLT_BFILE);

  CHECK_ERROR (OCIHandleFree((dvoid *) stmthp, OCI_HTYPE_STMT));
}
```

5.4.4 ODP.NET API for BFILEs

This section describes the ODP.NET APIs that you can use with BFILEs.

> **See Also:**
>
> OracleBFile Class

Table 5-6 ODP.NET methods in OracleBfileClass

Category	Function/Description	Description
Sanity Checking	`FileExists`	Checks if the BFILE exists on the server
	`FileName`	Sets or gets the file name
	`DirectoryName`	Sets or gets the `DIRECTORY` object name
Open/Close	`OpenFile`	Opens a file. Use `OPEN` instead of `FILEOPEN`.
	`IsOpen`	Checks if the file was opened using the input BFILE locators. Use `ISOPEN` instead of `FILEISOPEN`.

Table 5-6 (Cont.) ODP.NET methods in OracleBfileClass

Category	Function/Description	Description
	`CloseFile`	Closes the file.
Read Operations	`Length`	Get the length of the BFILE
	`Value`	Returns the entire LOB data as a string for CLOB and a byte array for BLOB
	`Read`	Reads data from the BFILE starting at the specified offset.
	`Search`	Returns the matching position of the nth occurrence of the pattern in the BFILE.
Operations involving multiple locators	`Compare`	Compares the values of two BFILEs
	`IsEqual`	Check if two LOBs point to the same LOB data

5.4.5 OCCI API for BFILEs

This section describes the OCCI APIs that you can use with `BFILE`s.

In OCCI, the `Bfile` class enables you to instantiate a `Bfile` object in your C++ application. You must then use methods of the `Bfile` class, such as the `setName()` method, to initialize the `Bfile` object, which associates the object properties with an object of type `BFILE` in a `BFILE` column of the database.

> **See Also:**
>
> Bfile Class

Amount Parameter for OCCI LOB copy() Methods

The `copy()` method on `Clob` and `Blob` enables you to load data from a `BFILE`. You can pass one of the following values for the `amount` parameter to this method:

- An amount smaller than the size of the `BFILE` to load a portion of the data
- An amount equal to the size of the `BFILE` to load all of the data
- The `UB8MAXVAL` constant to load all of the `BFILE` data

You cannot specify an amount larger than the length of the `BFILE`.

Amount Parameter for OCCI read() Operations

The `read()` method on an `Clob`, `Blob`, or `Bfile` object, reads data from a `BFILE`. You can pass one of these values for the amount parameter to specify the amount of data to read:

- An amount smaller than the size of the `BFILE` to load a portion of the data
- An amount equal to the size of the `BFILE` to load all of the data
- An amount equal to zero (`0`) to read until the end of the `BFILE` in streaming mode

You cannot specify an amount larger than the length of the `BFILE`.

Table 5-7 OCCI Methods for BFILEs

Category	Function/ Procedure	Description
Sanity Checking	`fileExists()`	Checks if the `BFILE` exists on the server
	`getFileName()`	Gets the file name
	`getDirAlias()`	Gets the `DIRECTORY` object name
	`setName()`	Sets the name of a `BFILE` in a locator without checking if the directory or file exists.
	`isInitialized()`	Checks whether a `BFILE` is initialized.
Open/Close	`open()`	Opens a file.
	`isOpen()`	Checks if the file was opened using the input BFILE locators.
	`close()`	Closes the file.
Read Operations	`length()`	Gets the length of the `BFILE`
	`read()`	Reads data from the `BFILE` starting at the specified offset.
Operations involving multiple locators	(operator) =	Assigns a `BFILE` locator to another. Use the assignment operator (=) or the copy constructor.
	`Blob.copy() or Clob.copy()`	Loads `BFILE`data into a LOB

5.4.6 Pro*C/C++ and Pro*COBOL API for BFILEs

This section describes Pro*C/C++ and Pro*COBOL APIs APIs you can use for BFILEs.

> **See Also:**
>
> - Pro*C/C++ Programmer's Guide
> - Pro*COBOL Programmer's Guide

Table 5-8 Pro*C/C++ and Pro*COBOL APIs for BFILEs

Category	Function/ Procedure	Description
Sanity Checking	`DESCRIBE[FILEEXISTS]`	Checks if the `BFILE` exists on the server
	`DESCRIBE[DIRECTORY,FILENAME]`	Gets the directory object name and file name
	`FILE SET`	Sets the name of a `BFILE` in a locator without checking if the directory or file exists
Open/Close	`OPEN`	Opens a file.

Table 5-8 (Cont.) Pro*C/C++ and Pro*COBOL APIs for BFILEs

Category	Function/ Procedure	Description
	DESCRIBE[ISOPEN]	Checks if the file was opened using the input BFILE locators.
	CLOSE	Closes the file.
	FILE CLOSE ALL	Closes all previously opened files.
Read Operations	DESCRIBE[LENGTH]	Gets the length of the BFILE
	READ	Reads data from the BFILE starting at the specified offset.
Operations involving multiple locators	ASSIGN	Assigns a BFILE locator to another
	LOAD FROM FILE	Loads BFILE data into a LOB