92

DBMS_FS

The DBMS_FS package for performing operations on an Oracle file system (make, mount, unmount and destroy operations) in an Oracle database.

This chapter contains the following topics:

DBMS_FS Overview

The DBMS_FS package contains Oracle file system (OFS) procedures that you can use to create, mount, unmount, and destroy an Oracle file system.

Starting 19c release, the file systems are supported by PDB. Oracle Database supports maximum 5 file systems per PDB and 1000 file systems in total.

The DBMS_FS package enables applications to access database objects from a universal client such as an NFS server. This feature interfaces with Oracle SecureFiles to provide the file system access.

See Also:

Oracle Database SecureFiles and Large Objects Developer's Guide for a detailed description of managing an NFS server in Oracle Database

DBMS_FS Security Model

You must have the SYSDBA administrative privilege to use the DBMS_FS package.

The operations that you perform using the DBMS_FS package are equivalent to the file system operations that are performed in an operating system by the root user. Access to the individual file system that is created and mounted by this package is enforced using Access Control Lists (ACLs) and the permissions on the mounted directories to the operating system user.

Summary of DBMS_FS Subprograms

The following table lists the DBMS_FS subprograms and briefly describes them.

Table 92-1 DBMS_FS Subprograms

Subprogram	Description
DESTROY_ORACLE_FS Procedure	Destroys an Oracle file system, using the `fstype` and of name `fsname`.
FS_EXISTS Procedure	Validates if the specified file system exists. Run this procedure before executing any mount, unmount or destroy operation.
LIST_FILES Procedure	Lists all the files within a specified directory in a file system.
MAKE_ORACLE_FS Procedure	Creates a file system of type specified by `fstype` and of name `fsname`.
MOUNT_ORACLE_FS Procedure	Mounts an Oracle file system on the specified mount point.
UNMOUNT_ORACLE_FS Procedure	Unmounts an Oracle file system on the specified mount point.

DESTROY_ORACLE_FS Procedure

This procedure destroys an Oracle file system and then frees the resources that were associated with it. Run the `dbms_fs.destroy_oracle_fs()` procedure to destroy file systems that are no longer in use.

Syntax

```
DBMS_FS.DESTROY_ORACLE_FS (
   fstype      IN VARCHAR2,
   fsname      IN VARCHAR2);
```

Parameters

Table 92-2 DBMS_FS Parameters

Parameter	Description
`fstype`	File system type. Oracle File System (OFS) and Database File System (DBFS) are supported.
`fsname`	Name of the file system

Usage Notes

- You can find information about the currently mounted file systems by querying the `V$OFSMOUNT` dynamic view.
- For more information about the file system types, see the `fstype` description in MAKE_ORACLE_FS Procedure.
- Before you run the `DBMS_FS.DESTROY_ORACLE_FS` procedure, you must unmount the file system by using the `DBMS_FS.UNMOUNT_ORACLE_FS` procedure. If you run this procedure on a file system that is still in use, it results in an error.
- After you run `DBMS_FS.DESTROY_ORACLE_FS`, Oracle Database destroys the file system and frees the associated resources.

Example

The following sample code shows how to destroy a DBFS file system, `dbfs_fs1`.

```
BEGIN
 DBMS_FS.DESTROY_ORACLE_FS (
  fstype           => 'dbfs',
  fsname           => 'dbfs_fs1');
END;
/
```

FS_EXISTS Procedure

Use the `dbms_fs.fs_exists()` procedure to validate if the given file system exists before executing any mount, unmount or destroy operation.

Syntax

```
DBMS_FS.FS_EXISTS (
   fsname       IN VARCHAR2);
```

Parameters

Table 92-3   DBMS_FS Parameters

Parameter	Description
`fsname`	Name of the file system.

Example

The following sample code shows how to check if a file system exists with the specified file system name, `data`.

```
SQL> declare
     fsname varchar2(64) := 'data';
     fsexists integer;

     begin
      fs_exists := dbms_fs.fs_exists(fsname);
      dbms_output.put_line('fs exists: ' || fsexists);
     end;
     /
SQL> select dbms_fs.fs_exists('data') from dual;
```

LIST_FILES Procedure

Run the `dbms_fs.list_files()` procedure to list all the files within a specified directory in a file system.

Syntax

```
DBMS_FS.LIST_FILES (
   fsname       IN VARCHAR2,
   dirpath      IN VARCHAR2);
```

Parameters

Table 92-4 DBMS_FS Parameters

Parameter	Description
`fsname`	Name of the file system.
`dirpath`	Path to an existing directory in the file system. Enter a path that is relative to the mount path. For example, if you enter `'/'` as the directory path, it is relative to the mount path.

Example

The following sample code shows how to list all the files within a specified directory in a file system.

```
SQL> select * from DBMS_FS.LIST_FILES ('data', '/') from dual;
```

Where, the path to the directory (`'/'`) is relative to the mount path.

MAKE_ORACLE_FS Procedure

This procedure creates a new file system of type `DBFS` or `OFS`, on top of an existing Oracle tablespace or other database object.

Syntax

```
DBMS_FS.MAKE_ORACLE_FS (
   fstype       IN VARCHAR2,
   fsname       IN VARCHAR2,
   fsoptions    IN VARCHAR2);
```

Parameters

Table 92-5 DBMS_FS Parameters

Parameter	Description
`fstype`	File system type. Enter one of the following values: • `ofs`—to create an Oracle File System • `dbfs`—to create a Database File System
`fsname`	Name of the file system. Enter a string no longer than 256 characters, using alphanumeric characters.
`fsoptions`	Specify an existing tablespace to use for the Oracle file system, using the following format: `"tablespace=`*`tablespace_name`*`"`

Usage Notes

- If you want to create a database file system (DBFS), then you must run the `dbfs_create_filesystem.sql` script, which in turn calls the `dbfs_create_filesystem_advanced.sql` script. By default, this script is in the `$ORACLE_HOME/rdbms/admin` directory. When you run this script, provide the name of an existing tablespace and a name for the file system that will be stored in the database. The size of the file system will be the same as the table size. For example, to create a file system in the `dbfs_ts` tablespace, in the file system `dbfs_tab`:

```
@/$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem.sql dbfs_ts dbfs_tab
```

After you run this script, you can use the other procedures in the `DBMS_FS` package to mount, unmount, and destroy the file system.

Note:

Staring Oracle Database `19.3.1.0` release, the `DBMS_FS.MAKE_ORACLE_FS` is used to create a `DBFS` filesystem; hence no auxiliary SQL script is needed to create a `DBFS` filesystem.

- Running the `DBMS_FS.MAKE_ORACLE_FS` procedure on the database instance is equivalent to running the `mkfs` command by root in an operating system.
- The tablespace that you specified in the `fsoptions` parameter must already exist before you execute the `DBMS_FS.MAKE_ORACLE_FS` procedure. To find existing tablespaces, query the `DBA_TABLESPACES` data dictionary view.
- The size of the file system is the same size as this tablespace.

Example

The following example shows how to create a DBFS file system named `dbfs_fs1` in the tablespace `dbfs_fs1_tbspc`.

```
BEGIN
 DBMS_FS.MAKE_ORACLE_FS (
  fstype           => 'dbfs',
  fsname           => 'dbfs_fs1',
  fsoptions        => 'TABLESPACE=dbfs_fs1_tbspc');
END;
/
```

MOUNT_ORACLE_FS Procedure

Use the `dbms_fs.mount_oracle_fs()` procedure to mount an Oracle file system or OFS managed file system on the specified mount point.

Before you begin, complete the following checks to ensure that:

- You have created the file system using the `dbms_fs.make_oracle_fs()` procedure.
- The mount point that you specify exists in the local node.
- The Oracle user has access permissions.
- The mount path is empty, which means that the directory specified by the mount point does not have any files.

Oracle supports an extensive list of mount options that you can use to have better control on resources and achieve good performance. When you use the `persist` mount option, file systems are automatically remounted every time instance restarts. Since OFSD is a non-fatal background process, it gets automatically restarted after the death of a process. File systems that are mounted at the time of the death of the OFSD process are automatically remounted after starting a new OFSD process. This ensures continuous availability of the mounted file systems even in the case of an error.

To improve the throughput, OFSD has its own local cache for writes and reads. The write cache uses 8 (1 MB) buffers per file connection, and it can use up to 256 MB per file system. You can modify this value through the mount option, `wcache_size`. A read-ahead algorithm is implemented for read operation and it uses 2 (1 MB) per file connection and it can use up to 256 MB per file system. Use the mount option, `rcache_size`, to modify this value.

The read-write cache is maintained per node, so this provides local cache consistency. In an RAC environment, consistency is guaranteed only after the `flush()` or `close()` operation is performed on the file. When two different processes on two different RAC nodes modify a single file and write to the same offset, then the first process that performs the `close()` operation on the file will have its data written into the file.

Syntax

```
DBMS_FS.MOUNT_ORACLE_FS (
  fstype            IN VARCHAR2,
  fsname            IN VARCHAR2,
  mount_point       IN VARCHAR2,
  mount_options     IN VARCHAR2);
```

Parameters

Table 92-6 MOUNT_ORACLE_FS Procedure Parameters

Parameter	Description
`fstype`	File system type. Oracle File System (OFS) and Database File System (DBFS) are supported.
`fsname`	Name of the file system. Enter a string no longer than 256 characters, using alpha numeric characters.
`mount_point`	Local directory where the file system should be mounted. This directory must already exist. Enter an absolute path. The maximum number of mount points that you can create is 5 mount points per PDB and 1000 mount points per instance.
`mount_options`	Comma-separated mount options, listed in Table 92-7.

Usage Notes

Table 92-7 Supported Mount Options for the MOUNT_ORACLE_FS procedure

Mount Option	Usage Description
`db_access`	Optimizes the read and write operations performed by database process when you access files managed through OFS or DBFS.
`default_permissions`	Enables permission check and restrict access based on file mode. This option is useful with the `allow_other` mount option.
`allow_other`	Allows other users apart from the operating system user that did the mount can access the files. This will be used in conjunction with permission checks in determining the file access. This option requires setting the `user_allow_other` parameter in the `/etc/fuse.conf` configuration file on Linux.
`max_read`	Maximum size of the read operation. No maximum size is set by default.

Table 92-7 (Cont.) Supported Mount Options for the MOUNT_ORACLE_FS procedure

Mount Option	Usage Description
`max_write`	Maximum write size in a single request. The default is 128K.
`direct_io`	Indicates to the operating system kernel not use file system cache.
`nopersist`	Does not store the mount options for use in next instance start up.
`persist`	Stores the mount entry persistently so that on subsequent instance start up it will be automatically mounted again. This option is supported for both `ofs` and `dbfs`.
`ro`	Mounts the file system in read-only mode. Files cannot be modified.
`rw`	Mounts the file system as read-write. This is the default.
`nosuid`	Specifies that the file system cannot contain set userid files.
`suid`	Specifies that the file system can contain set userid files. This is the default.
`ofs_cache_attr_time`	Specifies the OFS cache attribute timeout in seconds. Default value is 5 seconds. The permissible range is `0` to `UB4MAXVAL`. Oracle recommends that you specify low values when you use RAC. If you specify a larger value, such as 1000, the attributes from the files may not match if the same file system is mounted in another path. This is due to the aggressive caching that is done in the OFS layer.
`exec`	Allows executing a file under the mount path. This option is enabled by default.
`noexec`	Prohibits executing a file under the mount path. Use this option to disallow any file to be executed in the file system.
`atime`	Maintains information about the time when the file was accessed. This option is enabled by default.
`noatime`	Does not maintain information about the time when the was file accessed. When you enable this option, it increases throughput as the access time is not updated for every read operation.
`max_readahead`	Sets the maximum size, in bytes, for the read-ahead operations. The default value depends on the versions of the kernel and FUSE in your operating system.
`sync_read`	Permits only synchronous reads. If you use this option, it disables read-ahead caching for OFS.
`async_read`	Permits asynchronous reads. All read and write operations in the file system as performed asynchronously. This is the default option for FUSE versions 7.6 or later.
`dirsync`	Makes all directory operations synchronous. FUSE utilizes this option.
`big_writes`	Permits writes larger than 4KB. This is the default option for FUSE versions 7.6 or later.
`no_big_writes`	Does not allow writes larger than 4KB. This is not a recommended option for OFS.
`xattr_enabled`	Enables the use of extended attributes. Specify this option only if you want to use extended attributes.

Table 92-7 (Cont.) Supported Mount Options for the MOUNT_ORACLE_FS procedure

Mount Option	Usage Description
`kernel_cache_mode=writeback`	Makes the Linux kernel cache behave like a writeback cache when you enter `writeback` as the value for this option. A writeback cache provides better performance when there are many small input-output requirements. FUSE utilizes this option. Permitted values are: `writeback` and `write-through`. The default value is `write-through`.
`kernel_cache_attr_time=N`	Sets the Linux kernel attribute cache timeout to N seconds. The default value is infinity. Oracle recommends that you set low values for RAC setup.
`wcache_size=N`	Sets the OFS write cache size to N bytes. The default value is 256 MB.
`rcache_size`	Sets the OFS read cache size to N bytes. The default value is 256 MB.
`statfs_ctime`	Sets the time, in seconds, for which `statfs()` gets cached in OFS. The default value is 300 seconds or 5 minutes. Oracle recommends that you use the default value to avoid repeated `statfs()` calls from FUSE.
`no_rbt_cache`	Disallows the use of a red-black tree for caching directory entries. By default, a red-black tree is used to ensure accuracy when there are concurrent updates while listing directories.
`posix_locks`	Implements the POSIX file locks in OFS. This is not supported by OFS. If you want to use the file-locking mechanism, use the `noposix_locks` option.
`noposix_locks`	Implements POSIX file locks in Linux kernel. This is the default option.

Note:

The following options are exclusive options and cannot be used together:

- `nopersist`/`persist`
- `ro`/`rw`
- `nosuid`/`suid`
- `exec`/`noexec`
- `atime`/`noatime`
- `big_writes`/`no_big_writes`
- `sync_read`/`async_read`
- `posix_locks`/`noposix_locks`

Usage Notes

- This procedure makes the files system visible in the local database instance.

- For more information about the file system types, see the `fstype` description in MAKE_ORACLE_FS Procedure.
- You can find information about currently mounted file systems by querying the `V$OFSMOUNT` dynamic view.
- Run the `DBMS_FS.MOUNT_ORACLE_FS` procedure on a file system that has already been created with `DBMS_FS.MAKE_ORACLE_FS` in the local computer node where the Oracle database instance is running. You cannot run this procedure on file systems that were created outside of Oracle Database.
- You cannot update the mount options after mounting the file system. If you want to change the mount options later, you'll have to unmount the file system, and then remount it.

Example 1: Mounts a DBFS file system

Mounts a DBFS file system at `/oracle/dbfs/testfs`.

```
BEGIN
 DBMS_FS.MOUNT_ORACLE_FS (
  fstype          => 'dbfs',
  fsname          => 'dbfs_fs1',
  mount_point     => '/oracle/dbfs/testfs',
  mount_options   => 'default_permissions, allow_other, db_access');
END;
```

Example 2: Persist mount a DBFS file system

Persist mounts a DBFS file system at `/oracle/dbfs/testfs`.

```
BEGIN
 DBMS_FS.MOUNT_ORACLE_FS (
  fstype          => 'dbfs',
  fsname          => 'dbfs_fs1',
  mount_point     => '/oracle/dbfs/testfs',
  mount_options   => 'default_permissions, allow_other, persist, db_access');
END;
```

UNMOUNT_ORACLE_FS Procedure

This procedure unmounts an Oracle file system on the specified mount point.

File systems that are mounted in a PDB are automatically unmounted when the PDB is closed. You can also use `dbms_fs.unmount_oracle_fs()` to explicitly unmount an Oracle file system that you have mounted through OFS.

Syntax

```
DBMS_FS.UNMOUNT_ORACLE_FS (
  fsname              IN VARCHAR2,
  mount_point         IN VARCHAR2,
  unmount_options     IN VARCHAR2);
```

Table 92-8 UNMOUNT_ORACLE_FS Procedure Parameters

Parameter	Description
`fsname`	Name of the file system.
`mount_point`	Local directory where the file system had been mounted. Enter an absolute path.

Table 92-8 (Cont.) UNMOUNT_ORACLE_FS Procedure Parameters

Parameter	Description
`unmount_options`	Optionally, enter `force` or `clean`. Enter `force` to unmount the file system forcibly. This setting prevents new requests from being sent to the file system. All pending requests on the file system are either completed or canceled. If you omit this setting, then attempts to unmount a busy file system cause an `EBUSY` error. Enter `clean` if you don't want the file system to be remounted when the OFSD restarts.

Usage Notes

- Before you unmount the file system, ensure that all applications that use this file system are shut down. Also ensure that no processes reference the mounting file system.
- You can find information about the currently mounted file systems by querying the `V$OFSMOUNT` dynamic view.
- For more information about the file system types, see the `fstype` description in MAKE_ORACLE_FS Procedure.
- After unmounting the file system, the write permissions are removed from the mount point to prevent applications from writing to the underlying file system instead of the OFS supported file systems.
- When an Oracle instance is shut down in normal immediate mode, then all the mounted file systems are automatically unmounted.
- If a file system is mounted with the `MOUNT_ORACLE_FS` procedure with the `persist` option, it will be automatically mounted again when the database instance starts or the PDB is plugged. If this file system is unmounted by executing `DBMS_FS.UNMOUNT_ORACLE_FS`, it will remain unmounted even if the `persist` option was used to mount it.
- If you perform a `SHUTDOWN ABORT`, then the file system may still show as mounted but it may not be accessible. In this case, you can unmount the system manually by calling the `unmount` command at the operating system level or the `fusermount` procedure on Linux systems.
- Do not use `fusermount -u` to unmount a running file system as it causes inconsistency in Oracle views, such as `v$ofsmount`.
- You can export the local mount point of an Oracle file system to point to the remote system, and then NFS mount the file system from the remote system by using the operating system `mount` command. The `DBMS_FS.MOUNT_ORACLE_FS` procedure is similar to `mount` commands that are used for other local file systems.
- For better security, Oracle recommends that you use access control lists (ACLs) and Kerberos to control access to sensitive data.
- Do not attempt to unmount the file system from the operating system level. Doing so can leave the Oracle Database-created file system internal tables in an inconsistent state.

Example

The following sample code unmounts a DBFS mounted file system at `/oracle/dbfs/testfs`.

```
BEGIN
 DBMS_FS.UNMOUNT_ORACLE_FS (
  fsname           => 'dbfs_fs1',
  mount_point      => '/oracle/dbfs/testfs',
  mount_options    => 'force');
END;
```