69

# DBMS_DBFS_HS

The Oracle Database File System Hierarchical Store is implemented in the `DBMS_DBFS_HS` package. This package provides users the ability to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management for their database tables.

This chapter contains the following topics:

**See Also:**

- *Oracle Database SecureFiles and Large Objects Developer's Guide*

## DBMS_DBFS_HS Overview

The `DBMS_DBFS_HS` package is a service provider underneath the `DBMS_DBFS_CONTENT` package that enables use of tape or Amazon S3 Web service as storage for data.

The data on tape or Amazon S3 Web service is part of the Oracle Database and can be accessed through all standard interfaces, but only through the database. The package allows users to use tape or Amazon S3 Web service as a storage tier when doing Information Lifecycle Management of their content.

The package initially stores all content files in level-1 cache. As the level-1 cache fills up, content files are moved to level-2 cache and then to an external storage device using bulk writes.

## DBMS_DBFS_HS Security Model

The `DBMS_DBFS_HS` package runs with invoker's rights.

## DBMS_DBFS_HS Constants

The `DBMS_DBFS_HS` package uses the constants shown in the tables in this topic.

**Table 69-1 DBMS_DBFS_HS Constants - Used by the CREATESTORE Procedure**

| Constant | Type | Value | Description |
|---|---|---|---|
| `STORETYPE_TAPE` | `VARCHAR2(50)` | `'HS_TAPE'` | Use tape as a storage tier |
| `STORETYPE_AMAZONS3` | `VARCHAR2(50)` | `'HS_S3'` | Use Amazon S3 Web service as a storage tier |

**Table 69-2 DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| `PROPNAME_BUCKET` | `VARCHAR2(50)` | `'BUCKET'` | Specifies the AWS bucket to be used as a storage tier by the Hierarchical Store.<br>Restrictions on bucket name are:<br>1) Bucket names can only contain lowercase letters, numbers, periods (`.`) and dashes(-). Note that underscores (_) are invalid.<br>2) Bucket names must start with a number or letter.<br>3) Bucket names cannot be in an IP address style (`192.168.5.4`).<br>4) Bucket names must be between 3 and 63 characters long.<br>5) Bucket names should not end with a dash.<br>6) Dashes cannot appear next to periods. For example, `my-.bucket.com` is invalid. |
| `PROPNAME_CACHESIZE` | `VARCHAR2(50)` | `'CACHE_SIZE'` | Specifies the cumulative cache size used for the Hierarchical Store. This property is set by the CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure. It cannot be modified by the SETSTOREPROPERTY Procedure, though its value can be queried by the GETSTOREPROPERTY Function. |
| `PROPNAME_COMPRESSLEVEL` | `VARCHAR2(50)` | `'COMPRESSION_LEVEL'` | Use to enable compression of files stored in the DBFS hierarchical store. It specifies the compression level to be used for compressing the files |

**Table 69-2 (Cont.) DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| PROPNAME_ENABLECLEANUPONDELETE | VARCHAR2(50) | 'ENABLE_CLEANUP_ON_DELETE' | If this property is set to 'TRUE', whenever the user invokes the DELETEFILE Procedure in the DBMS_DBFS_CONTENT interface on a file residing in the DBMS_DBFS_HS store, the DBMS_DBFS_HS removes the file on the external storage that contains this user file, provided that the file has no other useful data.By default, the property is set to 'TRUE' for STORETYPE_AMAZONS3 and 'FALSE' for STORETYPE_TAPE. |
| PROPNAME_HTTPPROXY | VARCHAR2(50) | 'HTTP_PROXY' | Specifies the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service |
| PROPNAME_LICENSEID | VARCHAR2(50) | 'LICENSE_ID' | Specifies the license ID associated with the library libosbws11.so. |
| PROPNAME_LOBCACHE_QUOTA | VARCHAR2(50) | 'LOBCACHE_QUOTA' | Specifies fraction of the cache_size which is allocated for level 1 cache. The default value of this parameter is NULL which means that 0.8 (= 80%) of the cache_size is used for level 1 cache.<br>This property cannot be modified by the SETSTOREPROPERTY Procedurethough its value can be queried by the GETSTOREPROPERTY Function. Its value is set by CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure. |
| PROPNAME_MEDIAPOOL | VARCHAR2(50) | 'MEDIA_POOL' | Specifies the media pool number to use for storing the content |
| PROPVAL_COMPLVL_NONE | VARCHAR2(50) | 'NONE' | Indicates no compression |
| PROPVAL_COMPLVL_LOW | VARCHAR2(50) | 'LOW' | Use to set the compression level to LOW. This is expected to have the best performance while still providing a good compression ratio. |
| PROPVAL_COMPLVL_MEDIUM | VARCHAR2(50) | 'MEDIUM' | Use to set the compression level to MEDIUM. This compression level is expected to provide better compression ratio than LOW but the time required for compression will be higher than compression level LOW. |
| PROPVAL_COMPLVL_HIGH | VARCHAR2(50) | 'HIGH' | Use to set the compression level to HIGH. This compression level is expected to provide the best compression ratio but compression time will in general be highest among the 3 compression levels. |

**Table 69-2 (Cont.) DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| PROPNAME_OPTTARBALLSIZE | VARCHAR2(50) | 'OPTIMAL_TARBALL_SIZE' | Specifies optimal_tarball_size as the maximum possible size of an archive file.<br>Multiple content files are bundled together into one archive file and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation.<br>This property cannot be modified by the SETSTOREPROPERTY Procedurethough its value can be queried by the GETSTOREPROPERTY Function. Its value is set by CREATESTORE Procedure and can be modified by the RECONFIGCACHE Procedure. |
| PROPNAME_READCHUNKSIZE | VARCHAR2(50) | 'READ_CHUNK_SIZE' | Specifies the size used by the SBT protocol to transfer data from tape or S3.This chunk is allocated in memory per transaction for retrieval of content files from an archive store, so the value of this property should be conservative. The default size of 1MB is typically good for most users. |
| PROPNAME_S3HOST | VARCHAR2(50) | 'S3_HOST' | Specifies the HOST name of the Amazon S3 storage service. It must be s3.amazonaws.com. |
| PROPNAME_SBT_LIBRARY | VARCHAR2(50) | 'SBT_LIBRARY' | Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named libosbws11.so and is available in rdbms/lib directory. |
| PROPNAME_STREAMABLE | VARCHAR2(50) | 'STREAMABLE' | Indicates whether buffer-based PUT or GET should be done on this store. Valid values for are TRUE and FALSE. The default value of this property is TRUE. |

**Table 69-2 (Cont.) DBMS_DBFS_HS Constants - Used by the SETSTOREPROPERTY Procedure and the GETSTOREPROPERTY Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| `PROPNAME_WALLET` | `VARCHAR2(50)` | `'WALLET'` | The value of this property should be of the form:<br>`LOCATION=file:`*`filename`* `CREDENTIAL_ALIAS=`*`access/secret_alias`*<br>`PROXY_AUTH_ALIAS=`*`proxyusername/password alias`*<br>Defines the Oracle Wallet which contains the credentials of the Amazon S3 account associated with the store under consideration.<br>`LOCATION`: The directory path that contains the Oracle wallet. The format is `file:`*`directory-path`*<br>The format of `wallet_path` in Windows is, for example:<br>`file:c:\WINNT\Profiles\`*`username`*`\WALLETS`<br>In UNIX or Linux it is, for example:<br>`file:/home/`*`username`*`/wallets`<br>When the package is executed in the Oracle database server, the wallet is accessed from the database server.<br>`PASSWORD`: Defines the wallet password. If `auto-login` is enabled in wallet, this parameter does not have to be specified. By default, the `mkstore` utility enables `auto-login`.<br>`CREDENTIAL_ALIAS`: Defines the credential alias for `ACCESS_KEY` and `SECRET_KEY` |
| `PROPNAME_WRITECHUNKSIZ` | `VARCHAR2(50)` | `'WRITE_CHUNK_SIZE'` | Specifies the size used by the SBT protocol to transfer data to tape or S3.<br>This chunk is allocated in memory per transaction for `PUT` of Content Files to an archive store so the value should be conservative.<br>The default size of 1MB is typically good for most users. |

**Table 69-3 DBMS_DBFS_HS Constants - Used by the REGISTERSTORECOMMAND Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| `BEFORE_PUT` | `NUMBER` | `'1'` | Specified operation must be performed before writing a `SECUREFILE` to the remote store |

**Table 69-3 (Cont.) DBMS_DBFS_HS Constants - Used by the REGISTERSTORECOMMAND Function**

| Constant | Type | Value | Description |
|---|---|---|---|
| `BEFORE_GET` | `NUMBER` | `'2'` | Specified operation must be performed before a retrieval operation such as reading a `SECUREFILE` from the remote device |

**Table 69-4 DBMS_DBFS_HS Constants - Failure/Success/Error**

| Constant | Type | Value | Description |
|---|---|---|---|
| `FAIL` | `NUMBER` | `'0'` | Procedure or function did not execute successfully |
| `SUCCESS` | `NUMBER` | `'1'` | Procedure or function completed successfully |
| `ERROR` | `NUMBER` | `'2'` | Procedure or function returned an error |

## DBMS_DBFS_HS Operational Notes

When the `DBMS_DBFS_HS` package is executed in the Oracle database server, the wallet is accessed from the database server.

## Summary of DBMS_DBFS_HS Subprograms

This table lists and describes the subprograms in the `DBMS_DBFS_HS` package.

**Table 69-5 DBMS_DBFS_HS Package Subprograms**

| Subprogram | Description |
|---|---|
| CLEANUPUNUSEDBACKUPFILES Procedure | Removes files created on the external storage device that hold no currently used data |
| CREATEBUCKET Procedure | Creates an AWS bucket, associated with a store of type `STORETYPE_AMAZONS3` into which the Hierarchical Store can then move data |
| CREATESTORE Procedure | Creates a new hierarchical store |
| DEREGSTORECOMMAND Function | Removes a command that had been previously associated with a store through the RECONFIGCACHE Procedure |
| DROPSTORE Procedure | Deletes a previously created hierarchical store |
| FLUSHCACHE Procedure | Flushes (writes out) dirty contents from the level-1 cache. |
| GETSTOREPROPERTY Function | Retrieves the values of a property of a store |
| RECONFIGCACHE Procedure | Reconfigures the parameters of the database cache used by the store |
| REGISTERSTORECOMMAND Procedure | Registers commands for a store with the Hierarchical Store to be sent to the Media Manager for the external storage device associated with the store |

**Table 69-5 (Cont.) DBMS_DBFS_HS Package Subprograms**

| Subprogram | Description |
|---|---|
| SENDCOMMAND Procedures | Sends a command to be executed on the external storage device's Media Manager |
| SETSTOREPROPERTY Procedure | Stores properties of a store in the database |
| STOREPUSH Procedure | Pushes locally staged data to the remote storage |

## CLEANUPUNUSEDBACKUPFILES Procedure

This procedure removes files created on the external storage device that hold no currently used data in them.

### Syntax

```
DBMS_DBFS_HS.CLEANUPUNUSEDBACKUPFILES (
   store_name     IN     VARCHAR2);
```

### Parameters

**Table 69-6 CLEANUPUNUSEDBACKUPFILES Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |

### Usage Notes

- The action of removing files from external storage device can not be rolled back.
- This method can be executed periodically to clear space on the external storage device. Asynchronously deleting content from the external storage device is useful because it has minimal impact on the OLTP performance. The periodic scheduling can be accomplished using the DBMS_SCHEDULER package.

## CREATEBUCKET Procedure

This procedure creates an AWS bucket, associated with a store of type `STORETYPE_AMAZONS3` into which the Hierarchical Store can then move data.

### Syntax

```
DBMS_DBFS_HS.CREATEBUCKET  (
   store_name    IN     VARCHAR2);
```

### Parameters

**Table 69-7 CREATEBUCKET Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |

### Usage Notes

- The PROPNAME_BUCKET property of the store should be set before this subprogram is called.
- Once this procedure has successfully created a bucket in Amazon S3, the bucket can only be deleted using out-of-band methods, such as logging-in to S3 and deleting data (directories, files, and other items) for the bucket.

## CREATESTORE Procedure

This procedure creates a new hierarchical store store_name of type STORE_TYPE (STORETYPE_TAPE or STORETYPE_AMAZONS3) in schema schema_name (defaulting to the current schema) under the ownership of the invoking session user.

### Syntax

```
DBMS_DBFS_HS.CREATESTORE  (
   store_name            IN     VARCHAR2,
   store_type            IN     VARCHAR2,
   tbl_name              IN     VARCHAR2,
   tbs_name              IN     VARCHAR2,
   cache_size            IN     NUMBER,
   lob_cache_quota       IN     NUMBER DEFAULT NULL,
   optimal_tarball_size  IN     NUMBER DEFAULT NULL,
   schema_name           IN     VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 69-8 CREATESTORE Procedure Parameters**

| Parameter | Description |
|---|---|
| store_name | Name of store |
| store_type | STORETYPE_TAPE or STORETYPE_AMAZONS3 |
| tbl_name | Table for store entries |
| tbs_name | Tablespace for the store |
| cache_size | Amount of space used by the store to cache content in given tablespace |
| lob_cache_quota | Fraction of the cache_size which is allocated for level 1 cache. The default value of this parameter is NULL which means that 0.8 (= 80%) of the cache_size is used for level 1 cache. |
| optimal_tarball_size | Maximum possible size of the archive file.<br>Multiple content files are bundled together into one archive file, and then the archive file is transferred to tape or Amazon S3. This is because creating one file on tape or Amazon S3 for every content file in the store is a prohibitively expensive operation.<br>The value of is set by default to 10GB for tape and to 100MB for Amazon S3. |
| schema_name | Schema for the store |

### Usage Notes

CREATESTORE() sets certain properties of the store to default values. The user can use the methods SETSTOREPROPERTY() and RECONFIGCACHE() to appropriately change the property values and to set other properties of the store.

- Store names must be unique for an owner. The same store names can be used for different stores owned by different owners.
- Once a table space has been specified to store the store's content in a database, it cannot be changed later.
- This subprogram will execute like a DDL statement, performing an automatic `COMMIT` before and after execution.
- Stores using `DBMS_DBFS_HS` must not use singleton mount. This means that the singleton parameter should be `FALSE` and the `store_mount` parameter should have a non-`NULL` value in a call to the `DBMS_DBFS_CONTENT.MOUNTSTORE` procedure.

## DEREGSTORECOMMAND Function

This procedure removes a command that had been previously associated with a store through the REGISTERSTORECOMMAND Procedure.

### Syntax

```
DBMS_DBFS_HS.DEREGSTORECOMMAND (
   store_name       IN     VARCHAR2,
   message          IN     VARCHAR2);
```

### Parameters

**Table 69-9 DEREGSTORECOMMAND Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `message` | Message to be deregistered |

### Usage Notes

If this subprogram successfully executes, its actions cannot be rolled back by the user. If the user wants to restore the previous state, the user must call the REGISTERSTORECOMMAND Procedure.

### Related Topics

- REGISTERSTORECOMMAND Procedure
  This procedure registers commands for a store with the Hierarchical Store. These commands are sent to the Media Manager for the external storage device associated with the store.

## DROPSTORE Procedure

This procedure deletes a previously created hierarchical store specified by name and owned by the invoking session user.

### Syntax

```
DBMS_DBFS_HS.DROPSTORE  (
   store_name    IN     VARCHAR2,
   opt_flags     IN     INTEGER DEFAULT 0);
```

### Parameters

**Table 69-10 DROPSTORE Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store owned by the invoking session user |
| `opt_flags` | User can specify optional flags. If `DISABLE_CLEANUPBACKUPFILES` is specified as one of the optional flags, the call to the CLEANUPUNUSEDBACKUPFILES Procedure is not issued. By default, when this flag is not set, the procedure implicitly cleans-up all unused backup files. |

### Usage Notes

- The procedure executes like a DDL in that it auto-commits before and after its execution.
- If `CLEANUPBACKUPFILES` is disabled during the procedure, the user must resort to out-of-band techniques to cleanup unused backup files. No further invocations of `CLEANUPBACKFILES` for a dropped store are possible through hierarchical store.
- This subprogram will un-register the store from `DBMS_DBFS_CONTENT` package. All files in the given store are deleted from the store (Tape or Amazon S3 Web service). The database table holding the store's entries in the database, is also dropped by this subprogram.

## FLUSHCACHE Procedure

This procedure flushes out dirty contents from level-1 cache, which can be locked, to level-2 cache, thereby freeing-up space in level 1 cache.

### Syntax

```
DBMS_DBFS_HS.FLUSHCACHE  (
   store_name    IN     VARCHAR2);
```

### Parameters

**Table 69-11 FLUSHCACHE Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |

## GETSTOREPROPERTY Function

This function retrieves the values of a property.

### Syntax

```
DBMS_DBFS_HS.GETSTOREPROPERTY  (
   store_name       IN     VARCHAR2,
   property_name    IN     VARCHAR2,
   noexcp           IN     BOOLEAN DEFAULT FALSE) RETURN VARCHAR2;
```

### Parameters

**Table 69-12 GETSTOREPROPERTY Function Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `property_name` | Name of property |
| `noexcp` | If set to `FALSE`, raises an exception if the property does not exist in the database. If `noexcp` is set to `TRUE`, returns `NULL` if the property does not exist. |

### Return Values

The values of a property.

### Usage Notes

The specified store must already have been created.

## RECONFIGCACHE Procedure

This procedure reconfigures the parameters of the database cache being used by the store.

### Syntax

```
DBMS_DBFS_HS.RECONFIGCACHE (
   store_name            IN    VARCHAR2,
   cache_size            IN    NUMBER DEFAULT NULL,
   lobcache_quota        IN    NUMBER DEFAULT NULL,
   optimal_tarball_size  IN    NUMBER DEFAULT NULL);
```

### Parameters

**Table 69-13 RECONFIGCACHE Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `cache_size` | Cumulative cache size used for the Hierarchical Store |
| `lobcache_quota` | Fraction of the cache size that are assigned to level 1 cache |
| `optimal_tarball_size` | Maximum possible size of an archive file. Since creating one file for every content file in the store is a prohibitively expensive operation, multiple content files are bundled together into one archive file for transfer to tape or Amazon S3. |

### Usage Notes

- The specified store must already have been created before reconfiguration.
- The Hierarchical Store uses a level 1 cache and a level 2 cache. The level 1 cache subsumes most of the working set and the level 2 cache is used to perform bulk writes to the backend device.

- If any of the last 3 parameters is `NULL`, its value specified during store creation is used. If the parameter was `NULL` when the call to the CREATESTORE Procedure was issued, the `DBMS_DBFS_HS` package assigns a default value.

  The `DBMS_DBFS_HS` package optimistically tries to allocate more than 1 tarball's worth of size for level 2 cache to facilitate concurrency, though a minimum of 1 tarball size is necessary for level 2 cache.

  The values for cumulative cache size and LOB cache quota decide allocation of space for the two caches. If values are not provided, a user might see an `INSUFFICIENT_CACHE` exception. In that case, it is better to revise the cache parameters in order to have a working store.
- If this subprogram successfully executes, its actions cannot be rolled back by the user. In that case, the user should call `RECONFIGCACHE` again with new or modified parameters.

## REGISTERSTORECOMMAND Procedure

This procedure registers commands for a store with the Hierarchical Store. These commands are sent to the Media Manager for the external storage device associated with the store.

### Syntax

```
DBMS_DBFS_HS.REGISTERSTORECOMMAND (
   store_name       IN     VARCHAR2,
   message          IN     VARCHAR2,
   flags            IN     NUMBER);
```

### Parameters

**Table 69-14 REGISTERSTORECOMMAND Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `message` | Message to be sent to the Media Manager of the external store |
| `flags` | Valid values:<br>• `BEFORE_PUT CONSTANT NUMBER := 1;`<br>• `BEFORE_GET CONSTANT NUMBER := 2;` |

### Usage Notes

- These commands are sent before the next read or write of content. When the Hierarchical Store wants to push (or get) data to (or from) the storage device, it begins a session (to communicate with the device). After beginning the session, it sends all registered commands for the to the relevant device before writing (or getting) any data.
- If this method successfully executes, its actions cannot be rolled back by the user. To restore the previous state the user must call the DEREGSTORECOMMAND Function.

## SENDCOMMAND Procedure

This procedure sends a command to be executed on the external storage device's Media Manager.

### Syntax

```
DBMS_DBFS_HS.SENDCOMMAND (
   store_name     IN       VARCHAR2,
   message        IN       VARCHAR2);
```

### Parameters

**Table 69-15    SENDCOMMAND Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `message` | Message string to be executed |

## SETSTOREPROPERTY Procedure

This procedure stores properties of a store in the database as name-value pairs.

### Syntax

```
DBMS_DBFS_HS.SETSTOREPROPERTY  (
   store_name      IN     VARCHAR2,
   property_name   IN     VARCHAR2,
   property_value  IN     VARCHAR2);
```

### Parameters

**Table 69-16    SETSTOREPROPERTY Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store |
| `property_name` | For a store using Tape device, there are three properties whose values must be set by the user, and four properties that have default values. Stores of type `STORETYPE_AMAZONS3` have properties with default values. The various options for both types of stores are detailed under `property_value`. |

**Table 69-16 (Cont.) SETSTOREPROPERTY Procedure Parameters**

| Parameter | Description |
|---|---|
| `property_value` | **Stores using a Tape Device**<br>The values for the following properties must be set by the user:<br>• `PROPNAME_SBTLIBRARY` - This should point to the shared library used by RMAN to communicate with the external tape device. It is usually named `libobk.so`.<br>• `PROPNAME_MEDIAPOOL` - Media pool number for storing content<br>• `PROPNAME_CACHE_SIZE` - Amount of space, in bytes, used for the cache of this store<br>The following properties, which have default values assigned to them when a store is created, benefit from tuning:<br>• `PROPNAME_READCHUNKSIZE` and `PROPNAME_WRITECHUNKSIZE` - These are the sizes used by the SBT protocol to transfer data to and from the tapes. These chunks are allocated in memory per transaction, so the values should be conservative. The default size is 1MB.<br>• `PROPNAME_STREAMABLE` – Indicates whether `DBFS_LINK`s can perform read operations (for example `SELECT` or `DBMS_LOB.READ`) directly from the store, or if the data must be copied back into the database before it can be read<br>• `PROPNAME_ENABLECLEANUPONDELETE` - Indicates if `DBMS_DBFS_HS` should delete unused files on the external storage. Valid values for this property are '`FALSE`' for `STORETYPE_TAPE`.<br>• `PROPNAME_COMPRESSLEVEL` - Describes how files written to Tape should be compressed. It can be set to `PROPVAL_COMPLVL_NONE`, `PROPVAL_COMPLVL_LOW`, `PROPVAL_COMPLVL_MEDIUM` or `PROPVAL_COMPLVL_HIGH`. By default it is set to `PROPVAL_COMPLVL_NONE`. |

**Table 69-16 (Cont.) SETSTOREPROPERTY Procedure Parameters**

| Parameter | Description |
|---|---|
| (cont) property_value | **Stores of type STORETYPE_AMAZONS3**<br>It is mandatory that the following properties have assigned values, and default values are provided:<br>• PROPNAME_SBTLIBRARY - Specifies the path of the shared library used by RMAN to communicate with Amazon S3. It is named libosbws11.so and is available in rdbms/lib directory.<br>• PROPNAME_S3HOST - Defines the HOST name of the Amazon S3 storage service. It must be s3.amazonaws.com.<br>• PROPNAME_BUCKET - Defines the AWS bucket used as a storage tier by the Hierarchical Store. Restrictions on bucket names are:<br>-- Bucket names can only contain lowercase letters, numbers, periods (.) and dashes (-). Use of an underscore (_) is invalid.<br>-- Bucket names must start with a number or letter<br>-- Bucket names cannot be in an IP address style ("192.168.5.4")<br>-- Bucket names must be between 3 and 63 characters in length<br>-- Bucket names should not end with a dash<br>-- Dashes cannot appear next to periods. For example, "my-.bucket.com" is invalid.<br>• PROPNAME_LICENSEID - Specifies the license ID associated with the library libosbws11.so.<br>• PROPNAME_WALLET - Has the form: 'LOCATION=file:<filename> CREDENTIAL_ALIAS=<access/secret_alias> PROXY_AUTH_ALIAS=<proxyusername/password alias>'<br>-- LOCATION - Directory path that contains the Oracle wallet. The format is file:*directory-path*. See Examples for variations in format.<br>-- PASSWORD - Defines the wallet password. If auto-login is enabled in the wallet (this can be changed using the user's own utility), and does not have to be specified. By default, the mkstore utility enables auto-login.<br>-- CREDENTIAL_ALIAS - Defines the credential alias for ACCESS_KEY and SECRET_KEY<br>-- PROXY_AUTH_ALIAS - Defines authentication credentials for the proxy server, if applicable. |

**Table 69-16 (Cont.) SETSTOREPROPERTY Procedure Parameters**

| Parameter | Description |
|---|---|
| `(property_value` (contd.) | The following properties are optional:<br>• `PROPNAME_HTTPPROXY` - Defines the DNS name of the HTTP proxy, if any, that is needed to access the Amazon S3 storage service.<br>• `PROPNAME_STREAMABLE` – Indicates whether buffer-based `PUT` or `GET` operation should be done on this store. Valid values for this property are `TRUE` (default) and `FALSE`.<br>• `PROPNAME_ENABLECLEANUPONDELETE` - Indicates if `DBMS_DBFS_HS` should delete unused files on the external storage device. Default values for this property are `FALSE` for `STORETYPE_TAPE` and `TRUE` for `STORETYPE_AMAZONS3`.<br>• `PROPNAME_COMPRESSLEVEL` - Describes how files written to tape should be compressed. It can be set to `PROPVAL_COMPLVL_NONE`, `PROPVAL_COMPLVL_LOW`, `PROPVAL_COMPLVL_MEDIUM` or `PROPVAL_COMPLVL_HIGH`. By default it is set to `PROPVAL_COMPLVL_NONE`. |

### Usage Notes

- The specified store must already have been created.
- If this subprogram successfully executes, its actions cannot be rolled back by the user.
- The same property can be set multiple times to the same or different values using this subprogram
- Regarding `PROPNAME_ENABLECLEANUPONDELETE` behavior, a job is created for each store by the `DBMS_DBFS_HS` to remove the unused files from the external storage. By default, the job is enabled for `STORETYPE_AMAZONS3` and is disabled for `STORETYPE_TAPE`. If the `ENABLECLEANUPONDELETE` property is set to `TRUE`, the job is enabled; if the property is set to `FALSE`, the job is disabled. If enabled, the job runs at an interval of one hour by default. The `DBMS_SCHEDULER` package can be used to modify the schedule. The name of the job can be obtained by querying `USER_DBFS_HS_FIXED_PROPERTIES` for `prop_name = 'DELJOB_NAME'`.

### Examples

Format

The format of `wallet_path` in Windows is, for example:

```
file:c:\WINNT\Profiles\<username>\WALLETS
```

The format of `wallet_path` in UNIX or Linux is, for example:

```
file:/home/username/wallets
```

## STOREPUSH Procedure

This procedure pushes locally staged data to the remote storage.

### Syntax

```
DBMS_DBFS_HS.STOREPUSH (
   store_name    IN         VARCHAR2,
   path          IN         VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 69-17 STOREPUSH Procedure Parameters**

| Parameter | Description |
|---|---|
| `store_name` | Name of store whose content the client writes from local cache to the external store |
| `path` | A non-mount qualified (without mount point) path within the store. By default, its value is `NULL` which corresponds to the root path of the store. |

### Usage Notes

- The Hierarchical Store caches the content files locally in database tables. When enough content is amassed in the cache to make it efficient to write to the external storage device (or the cache is completely filled), the Hierarchical Store creates a tarball out of the local content and writes these tarballs as files on the external device. The size of the tarball created by the Hierarchical Store is controlled by the store property `PROPNAME_OPTTARBALLSIZE`.
- When the amount of free space in the cache is such that the caching of a content file will push the space used above `cache_size`, the Hierarchical Store will internally call `STOREPUSH`. The `STOREPUSH Procedure` creates tarball(s) out of the existing dirty or modified content files in the cache and writes them out to the external device. A `STOREPUSH` call is not guaranteed to write all the dirty content from local cache to the external storage, since some files may be locked by other sessions.
- `STOREPUSH` has a built-in ability feature allowing it to automatically resume operation. If a `STOREPUSH` call is interrupted (say by a network outage) after it has transferred some tarballs to the external device, it can be restarted after the outage and will then resume transferring data from the point it was interrupted. In other words, work done before the outage is not lost. `STOREPUSH` can safely be restarted and the effect is such as if the outage never occurred.
- If this method successfully executes, its actions cannot be rolled back by the user.
- By default, when `path` is `NULL`, all files in the store are candidates for `STOREPUSH`. If `path` has a valid input value, all files which are under the namespace of given path are written from the local cache to the external store. If a given path is an existing file, it is pushed out again to the remote store.