67
DBMS_DBFS_CONTENT

The DBMS_DBFS_CONTENT package provides an interface comprising a file system-like abstraction backed by one or more Store Providers.

This chapter contains the following topics:

> **See Also:**
>
> *Oracle Database SecureFiles and Large Objects Developer's Guide*

DBMS_DBFS_CONTENT Overview

The DBMS_DBFS_CONTENT package provides an interface that connects the Oracle database to the DBFS client-side.

In the server, the DBMS_DBFS_CONTENT package is backed by the DBMS_DBFS_CONTENT_SPI package, which includes descriptions but not actual implementations of DBFS stores.

> **See Also:**
>
> - Oracle Database SecureFiles and Large Objects Developer's Guide for a description of DBMS_DBFS_CONTENT and the DBFS architecture
> - Oracle Database SecureFiles and Large Objects Developer's Guide for conceptual information about the DBMS_DBFS_CONTENT package

DBMS_DBFS_CONTENT Security Model

The DBMS_DBFS_CONTENT package runs under AUTHID CURRENT_USER.

DBMS_DBFS_CONTENT Constants

The DBMS_DBFS_CONTENT package uses various types of constants.

These are shown in the following tables:

Path Name Constants and Types

The following constants are useful for declaring paths and item names. Paths are limited to 1024 characters and item names are limited to 256 characters.

Table 67-1 DBMS_DBFS_CONTENT Constants - Path Names

Constant	Type	Value	Description
NAME_MAX	PLS_INTEGER	256	Maximum length of an absolute path name visible to clients
NAME_T	VARCHAR2(256)	NAME_MAX	Portable alias for string that can represent component names
PATH_MAX	PLS_INTEGER	1024	Maximum length of any individual component of an absolute path name visible to clients
PATH_T	VARCHAR2(1024)	PATH_MAX	Portable alias for string that can represent path names

ContentID Constants

Stores may expose to the user a unique identifier that represents a particular path item in the store. These identifiers are limited to 128 characters.

Table 67-2 DBMS_DBFS_CONTENT Constants - ContentID

Constant	Type	Value	Description
CONTENT_ID_MAX	PLS_INTEGER	128	Maximum length of a store-specific provider-generated contentID that identifies a file-type content item
CONTENT_ID_T	RAW(128)	CONTENT_ID_MAX	Portable alias for raw buffers that can represent contentID values

Path Properties Constants

Every path name in a store is associated with a set of properties. Each property is identified by a string "name", has a string "value" (which might be NULL if unset or undefined or unsupported by a specific store implementation) and a value "typecode" (a numeric discriminant for the actual type of value held in the "value" string.)

Table 67-3 DBMS_DBFS_CONTENT Constants - Properties

Constant	Type	Value	Description
PROPNAME_MAX	PLS_INTEGER	32	Maximum length of a property name
PROPNAME_T	VARCHAR2(32)	PROPNAME_MAX	Portable alias for string that can represent property names
PROPVAL_MAX	PLS_INTEGER	1024	Maximum length of the string value of a property
PROPVAL_T	VARCHAR2(1024)	PATH_MAX	Portable alias for string that can represent property values

Path Name Type Constants

Path items in a store have a item type associated with them. These types represent the kind of entry the item represents in the store.

Table 67-4 DBMS_DBFS_CONTENT Constants - Path Name Types

Constant	Type	Value	Description
TYPE_FILE	PLS_INTEGER	1	A regular file storing content (a logically linear sequence of bytes accessed as a BLOB
TYPE_DIRECTORY	PLS_INTEGER	2	A container of other path name types, including file types
TYPE_LINK	PLS_INTEGER	3	A symbolic link (that is, an uninterpreted string value associated with a path name). Since symbolic links may represent path names that fall outside the scope of any given store (or even the entire aggregation of stores managed by the DBMS_DBFS_CONTENT interface), or may not even represent path names, clients must be careful in creating symbolic links, and stores must be careful in trying to resolve these links internally.
TYPE_REFERENCE	PLS_INTEGER	4	A hard link which is always a valid path name alias to content
TYPE_SOCKET	PLS_INTEGER	5	UNIX domain socket created using socket interface defined as in socket.h with domain defined as AF_UNIX or AF_LOCAL.

Store Feature Constants

The DBFS content API allows different store providers (and different stores) to describe themselves through a *feature set* (a bitmask indicating which features they support and which ones they do not).

Table 67-5 DBMS_DBFS_CONTENT Constants - Store Features

Constant	Type	Value	Description
FEATURE_FOLDERS	PLS_INTEGER	1	Set if the store supports folders (or directories) as part of hierarchical path names
FEATURE_FOIAT	PLS_INTEGER	2	Set if implicit folder operations within the store (performed as part of a client-requested operation) runs inside autonomous transactions.In general, the use of autonomous transactions is a compromise between (a) simplicity in the implementation and client-controlled transaction scope for all operations, at the cost of greatly reduced concurrency (FEATURE_FOIAT not set), versus (b) more complex implementation and smaller client-controlled transaction scope, at the benefit of greatly increased concurrency (FEATURE_FOIAT set).
FEATURE_NOWAIT	PLS_INTEGER	4	Set if the store allows nowait gets of path elements. The default behavior is to wait for row locks; if nowait gets are implemented, the get operation raises an ORA-54 exception if the path element is already locked by another transaction.
FEATURE_ACLS	PLS_INTEGER	8	Set if the store supports Access Control Lists (ACLs) and internal authorization or checking based on these ACLs. ACLs are standard properties but a store may do nothing more than store and retrieve the ACLs without interpreting them in any way.
FEATURE_LINKS	PLS_INTEGER	16	Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself
FEATURE_LINK_DEREF	PLS_INTEGER	32	Set if the store supports symbolic links, and if certain types of symbolic links (specifically non-absolute path names) can be internally resolved by the store itself
FEATURE_REFERENCES	PLS_INTEGER	64	Set if the store supports hard links

Table 67-5 (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

Constant	Type	Value	Description
FEATURE_LOCKING	PLS_INTEGER	128	Set if the store supports user-level locks (read-only, write-only, read-write) that can be applied on various items of the store, and if the store uses these lock settings to control various types of access to the locked items. User-level locks are orthogonal to transaction locks and persist beyond the scope of any specific transaction, session, or connection — this implies that the store itself may not be able to clean up after dangling locks, and client-applications need to perform any garbage collection.
FEATURE_LOCK_HIERARCHY	PLS_INTEGER	256	Set if the store allows a user-lock to control access to the entire sub-tree under the locked path name.
FEATURE_LOCK_CONVERT	PLS_INTEGER	512	Set if the store supports upgrade or downgrade of locks from one mode to another
FEATURE_VERSIONING	PLS_INTEGER	1024	Set if the store supports at least a linear versioning and version management. Different versions of the same path name are identified by monotonic version numbers, with a version-nonqualified path name representing the latest version.
FEATURE_VERSION_PATH	PLS_INTEGER	2048	Set if the store supports a hierarchical namespace for different versions of a path name
FEATURE_SOFT_DELETES	PLS_INTEGER	4096	Set if the store supports a "soft-delete", that is, the ability to delete a path name and make it invisible to normal operations, but retain the ability to restore the path name later (as long as it has not been overwritten by a new create operation). The store also supports purging soft-deleted path names (making them truly deleted), and navigation modes that show soft-deleted items.
FEATURE_HASHING	PLS_INTEGER	8192	Set if the store automatically computes and maintains some type of a secure hash of the contents of a path name (typically a TYPE_FILE path).
FEATURE_HASH_LOOKUP	PLS_INTEGER	16384	Set if the store allows "content-based addressing", that is, the ability to locate a content item based, not on its path name, but on its content hash.

Table 67-5 (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

Constant	Type	Value	Description

FEATURE_FILTERING	PLS_INTEGER	32768	Set if the store allows clients to pass a filter function (a PL/SQL function conforming to the signature below) that returns a logical boolean indicating if a given store item satisfies a selection predicate. Stores that support filtering may be able to more efficiently perform item listing, directory navigation, and deletions by embedding the filtering logic inside their implementation. If filtering is not supported, clients can retrieve more items than necessary and perform the filtering checks themselves, albeit less efficiently.
			A filter predicate is a function with the following signature:
			`function filterFunction(`
			` path IN VARCHAR2,`
			` store_name IN VARCHAR2,`
			` opcode IN INTEGER,`
			` item_type IN INTEGER,`
			` properties IN DBMS_DBFS_CONTENT_PROPERTIES_T,`
			` content IN BLOB)`
			`  RETURN INTEGER;`
			Any PL/SQL function conforming to this signature can examine the contents and properties of a store item, and determine if the item satisfies the selection criterion for the current operation. Any nonzero return value results in the DBMS_DBFS_CONTENT interface processing the item as part of the current operation; a return value that is zero or NULL results in the item being skipped from processing.

FEATURE_SEARCHING	PLS_INTEGER	65536	Set if the store allows clients to pass a text-search filter query to locate type_file path names based on their content. Stores that support searching may use indexes to accelerate such searches; otherwise, clients need to build their own indexes, or else search a potentially larger set of items to locate the ones of interest for the current search.
FEATURE_ASOF	PLS_INTEGER	131072	Set if the store allows clients to use a flashback timestamp in query operations (non-mutating GETPATH Procedures and LIST Function).

Table 67-5 (Cont.) DBMS_DBFS_CONTENT Constants - Store Features

Constant	Type	Value	Description
FEATURE_PROVIDER_PROPS	PLS_INTEGER	262144	Set if the store allows per-operation properties (that control the behavior of the store with regard to the current operation, as opposed to properties associated with individual items).
FEATURE_SNAPSHOTS	PLS_INTEGER	524288	Set if the store allows the use of named, read-only snapshots of its contents. It is up to the provider to implement snapshots using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between snapshots and its parent content view.
FEATURE_CLONES	PLS_INTEGER	1048576	Set if the store allows the use of named, writable clones of its contents. It is up to the provider to implement clones using any suitable means (including creating immediate copies of the content, or using copy-on-write) and managing dependencies between clones and its parent content view.
FEATURE_LOCATOR	PLS_INTEGER	2097152	Set if the store allows direct access to file contents through a LOB locator. Stores that internally manipulate the file contents, perhaps by shredding or reassembling them in separate pieces, performing other transformations, and so on, cannot transparently give out a LOB locator to clients. The file contents of these stores should be accessed using the buffer-based interfaces.
FEATURE_CONTENT_ID	PLS_INTEGER	4194304	Set if the store allows a "pathless", contentID-based access to files (there is no notion of a directory, link, or reference in this model)
FEATURE_LAZY_PATH	PLS_INTEGER	8388608	Set if the store allows a lazy binding of a path name to file content elements that are otherwise identified by a contentID; this feature makes sense only in conjunction with FEATURE_CONTENT_ID

Lock Type Constants

Stores that support locking should implement 3 types of locks: LOCK_READ_ONLY, LOCK_WRITE_ONLY, and LOCK_READ_WRITE.

Table 67-6 DBMS_DBFS_CONTENT Constants - Lock Types

Constant	Type	Value	Description
LOCK_READ_ONLY	PLS_INTEGER	1	Locks as read-only

Table 67-6 (Cont.) DBMS_DBFS_CONTENT Constants - Lock Types

Constant	Type	Value	Description
LOCK_WRITE_ONLY	PLS_INTEGER	2	Locks as write-only
LOCK_READ_WRITE	PLS_INTEGER	3	Locks as read-write

Standard Property Constants

Standard properties are well-defined, mandatory properties associated with all content path names that all stores should support (in the manner described by the content interface), with some exceptions. For example, a read-only store need not implement a `modification_time` or `creation_time`.

All standard properties informally use the `STD` namespace, which clients and stores should avoid using.

Table 67-7 DBMS_DBFS_CONTENT Constants - Standard Properties

Constant	Type	Value	Description
STD_ACCESS_TIME	VARCHAR2(32)	'std:access_time'	TYPECODE_TIMESTAMP in UTC: The time of last access of a path name's contents
STD_ACL	VARCHAR2(32)	'std:acl'	TYPECODE_VARCHAR2: The access control list (in standard ACL syntax) associated with the path name
STD_CANONICAL_PATH	VARCHAR2(32)	'std:canonical_path'	TYPECODE_VARCHAR2: The canonical store-specific path name of an item.
STD_CHANGE_TIME	VARCHAR2(32)	'std:change_time'	TYPECODE_TIMESTAMP in UTC: The time of last change to the metadata of a path name
STD_CHILDREN	VARCHAR2(32)	'std:children'	TYPECODE_NUMBER: The number of child directories/folders a directory/folder path has (this property should be available in providers that support the FEATURE_FOLDERS feature)
STD_CONTENT_TYPE	VARCHAR2 (32)	'std:content_type'	TYPECODE_NUMBER: The client-supplied mime-type(s) (in standard RFC syntax) describing the (typically type_file) path name. The content type is not necessarily interpreted by the store.
STD_CREATION_TIME	VARCHAR2(32)	'std:creation_time'	TYPECODE_TIMESTAMP in UTC: The time at which the item was created (once set, this value never changes for the lifetime of the path name)
STD_DELETED	VARCHAR2(32)	'std:deleted'	TYPECODE_NUMBER as a BOOLEAN: Set to a nonzero number if the path name has been soft-deleted but not yet purged.

Table 67-7 (Cont.) DBMS_DBFS_CONTENT Constants - Standard Properties

Constant	Type	Value	Description
STD_GUID	VARCHAR2(32)	'std:guid'	TYPECODE_NUMBER: A store-specific unique identifier for a path name. Clients must not depend on the GUID being unique across different stores, but a given (*store-name*, *store-specific-path name*) has a stable and unique GUID for its lifetime.
STD_LENGTH	VARCHAR2(32)	'std:length'	TYPECODE_NUMBER: The length of the content (BLOB) of a TYPE_FILE/ TYPE_REFERENCE path, or the length of the referent of a TYPE_LINK symbolic link. Directories do not have a well-defined length and stores are free to set this property to zero, NULL, or any other value.
STD_MODIFICATION_TIME	VARCHAR2(32)	'std:modification_time'	TYPECODE_TIMESTAMP in UTC: The time of last change to the data associated with a path name. Change to the content of a TYPE_FILE/ TYPE_REFERENCE path, the referent of the TYPE_LINK path, and addition or deletion of immediate children in a TYPE_DIRECTORY path, all constitute data changes.
STD_OWNER	VARCHAR2(32)	'std:owner'	TYPECODE_VARCHAR2: A client-supplied (or implicit) owner name for the path name. The owner name may be used (along with the current "principal") for access checks by stores that support ACLs, locking, or both.
STD_PARENT_GUID	VARCHAR2(32)	'std:parent_guid'	TYPECODE_NUMBER: A store-specific unique identifier for the parent of a path name. Clients must not depend on the GUID being unique across different stores, but a given (*store-name*, *store-specific-path name*) has a stable and unique GUID for its lifetime. STD_PARENT_GUID(path name) == STD_GUID(parent(path name))
STD_REFERENT	VARCHAR2(32)	'std:referent'	TYPECODE_VARCHAR2: The content of the symbolic link of a TYPE_LINK path; NULL otherwise. As mentioned, the STD_REFERENT can be an arbitrary string and must not necessarily be interpreted as path name by clients (or such interpretation should be done with great care).

Optional Property Constants

Optional properties are well-defined properties (not mandatory) associated with all content path names that all stores are free to support (but only in the manner described by the DBFS content API).

All optional properties informally use the `opt:` namespace, which clients and stores should avoid using.

Table 67-8 DBMS_DBFS_CONTENT Constants - Optional Properties

Constant	Type	Value	Description
OPT_HASH_TYPE	VARCHAR2(32)	'opt:hash_type'	TYPECODE_NUMBER: The type of hash provided in the opt_hash_value property; see DBMS_CRYPTO for possible options.
OPT_HASH_VALUE	VARCHAR2(32)	'opt:hash_value'	TYPECODE_NUMBER: The hash value of type OPT_HASH_TYPE describing the content of the path name.
OPT_LOCK_COUNT	VARCHAR2(32)	'opt:lock_count'	TYPECODE_NUMBER: The number of (compatible) locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the opt_locker must specify all lockers (with repeats so that lock counts can be correctly maintained).
OPT_LOCK_DATA	VARCHAR2(32)	'opt:lock_data'	TYPECODE_NUMBER: The client-supplied user-data associated with a user-lock, uninterpreted by the store.
OPT_LOCKER	VARCHAR2(32)	'opt:locker'	TYPECODE_NUMBER: One or more implicit or client-specified principals that applied a user-lock on a path name.
OPT_LOCK_STATUS	VARCHAR2(32)	'opt:lock_status'	TYPECODE_NUMBER: One of the LOCK_READ_ONLY, LOCK_WRITE_ONLY, LOCK_READ_WRITE values describing the type of lock currently applied on a path name.
OPT_VERSION	VARCHAR2(32)	'opt:version'	TYPECODE_NUMBER: A sequence number for linear versioning of a path name.
OPT_VERSION_PATH	VARCHAR2(32)	'opt:version_path'	TYPECODE_NUMBER: A version-path name for hierarchical versioning of a path name.
OPT_CONTENT_ID	VARCHAR2(32)	'opt:content_id'	TYPECODE_NUMBER: A provider-generated store-specific unique contentID in the form of a string for a file content element (that may optionally not be associated with a path; see FEATURE_CONTENT_ID and FEATURE_LAZY_PATH).

Property Access Flag Constants

Content interface methods to get or set properties can use combinations of property access flags to fetch properties from different name spaces in a single interface call.

Table 67-9 DBMS_DBFS_CONTENT Constants - Property Access Flags

Constant	Type	Value	Description
PROP_NONE	PLS_INTEGER	0	None: used when the client is not interested in any properties, and is invoking the content access method for other reasons (path name existence or lockability validation, data access, and so on)
PROP_STD	PLS_INTEGER	1	Mandatory: used when the client is interested in the standard properties; all standard properties are retrieved if this flag is specified.
PROP_OPT	PLS_INTEGER	2	Optional: used when the client is interested in the optional properties; all optional properties are retrieved if this flag is specified.
PROP_USR	PLS_INTEGER	3	User-defined: used when the client is interested in the user-defined properties; all user-defined properties are retrieved if this flag is specified.
PROP_ALL	PLS_INTEGER	PROP_STD + PROP_OPT + PROP_USR ;	All: an alias for the combination of all standard, optional, and user-defined properties
PROP_DATA	PLS_INTEGER	8	Content: used when the client is interested only in data access, and does not care about properties
PROP_SPC	PLS_INTEGER	16	Specific: used when the client is interested in a mix-and-match of different subsets of various property name spaces; the names of the specific properties to fetch are passed into the content interface method call as arguments, and only these property values are fetched and returned to the client. This is useful in cases where there are a very large number of properties potentially accessible, but the client is interested in only a small number of them (and knows the names of these "interesting" properties beforehand). PROP_SPC is applicable only to the various GETPATH operations. Other operations that specify properties ignore PROP_SPC specifications.

Operation Code Constants

All of the operations in the DBFS content API are represented as abstract opcodes.

Clients can use these opcodes to directly and explicitly by invoking the CHECKACCESS Function to verify if a particular operation can be invoked by a given principal on a particular path name.

Table 67-10 DBMS_DBFS_CONTENT Constants - Operation Codes

Constant	Type	Value	Description
OP_CREATE	PLS_INTEGER	1	Create a path item
OP_CREATEFILE	PLS_INTEGER	OP_CREATE	Create a file
OP_CREATELINK	PLS_INTEGER	OP_CREATE	Create a soft link
OP_CREATEREFERENCE	PLS_INTEGER	OP_CREATE	Create a reference (hard link)
OP_DELETE	PLS_INTEGER	2	Soft-deletion, purge, and restore operations are all represented by OP_DELETE
OP_DELETEFILE	PLS_INTEGER	OP_DELETE	Delete a file
OP_DELETEDIRECTORY	PLS_INTEGER	OP_DELETE	Delete a directory
OP_RESTORE	PLS_INTEGER	OP_DELETE	Restore a soft-deleted path item
OP_PURGE	PLS_INTEGER	OP_DELETE	Purge a soft-deleted path item
OP_READ	PLS_INTEGER	3	Read from a path item
OP_GET	PLS_INTEGER	OP_READ	Get a path item for either read or update operations
OP_WRITE	PLS_INTEGER	4	Write a path item
OP_PUT	PLS_INTEGER	OP_WRITE	Put (write) to a path item
OP_RENAME	PLS_INTEGER	5	Rename a path item
OP_RENAMEFROM	PLS_INTEGER	OP_RENAME	Operations performed on the source of a rename
OP_RENAMETO	PLS_INTEGER	OP_RENAME	Operations performed on the destination of a rename
OP_SETPATH	PLS_INTEGER	OP_RENAME	Set a path item name
OP_LIST	PLS_INTEGER	6	Perform a path listing
OP_SEARCH	PLS_INTEGER	7	Perform a search
OP_LOCK	PLS_INTEGER	8	Lock a path item
OP_UNLOCK	PLS_INTEGER	9	Unlock a path item

Table 67-10 (Cont.) DBMS_DBFS_CONTENT Constants - Operation Codes

Constant	Type	Value	Description
OP_ACL	PLS_INTEGER	10	An implicit operation invoked during an OP_CREATE or OP_PUT that specifies a STD_ACL property; the operation tests to see if the principal is allowed to set or change the ACL of a store item
OP_STORE	PLS_INTEGER	11	A catch-all category for miscellaneous store operations that do not fall under any of the other operational interfaces

Exceptions

DBFS content API operations can raise any one of these top-level exceptions.

Table 67-11 DBMS_DBFS_CONTENT Exceptions

Exception	Code	Description
PATH_EXISTS	64000	A specified path name already exists
INVALID_PARENT	64001	Parent of a specified path name does not exist
INVALID_PATH	64002	Specified path name does not exist, or is not valid
UNSUPPORTED_OPERATION	64003	An operation unsupported by a store was invoked
INVALID_ARGUMENTS	64004	An operation was invoked with invalid arguments
INVALID_ACCESS	64005	Access control checks failed for the current operation
LOCK_CONFLICT	64006	Current operation failed lock conflict check
INVALID_STORE	64007	An invalid store name was specified
INVALID_MOUNT	64008	An invalid mount point was specified
INVALID_PROVIDER	64009	An invalid provider-package was specified
READONLY_PATH	64010	A mutating operation was invoked on a read-only mount or store

DBMS_DBFS_CONTENT Operational Notes

This topic lists operational notes for DBMS_DBFS_CONTENT implementation, path names, and other operations.

- Implementation
- Path Names
- Other DBMS_DBFS_CONTENT Operations

Implementation

Since the interconnection of the DBMS_DBFS_CONTENT interface and the provider SPI is a 1-to-many pluggable architecture, the interface uses dynamic SQL to invoke methods in the provider SPI, this can lead to runtime errors.

There are no explicit `INIT` or `FINI` methods to indicate when the `DBMS_DBFS_CONTENT` interface plugs or unplugs a particular provider SPI. Provider SPIs must be willing to auto-initialize themselves at any SPI entry-point.

All operations performed by a store provider are "stateless" in that they are complete operations unto themselves. If state is necessary to be maintained for some reason, then the state must be maintained in data structures such as auxiliary tables that can be queried as needed.

Path Names

All path names used in the provider SPI are store-qualified in pair form (`store_name`, `pathname`) where the path name is rooted within the store namespace.

Stores and their providers that support contentID-based access (see `FEATURE_CONTENT_ID` in Table 67-5) also support a form of addressing that is not based on path names. Content items are identified by an explicit store name, a `NULL` path name, and possibly a contentID specified as a parameter or by way of the `OPT_CONTENT_ID` (see Table 67-8) property.

Not all operations are supported with contentID-based access, and applications should depend only on the simplest create or delete functionality being available.

This table lists other operations and provides links to related discussions.

Other DBMS_DBFS_CONTENT Operations

Table 67-12 Other DBMS_DBFS_CONTENT Operations

Other Operations	See ...
Creation	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on creation operations
Deletion	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on deletion operations
Get (Retrieve) and Put (Insert)	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on Get and Put operations
Rename and Move	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on Rename and Move operations
Directory Navigation and Search	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on Navigation and Search operations
Locking	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on Locking operations
Access Check	*Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on Access Check operations

DBMS_DBFS_CONTENT Data Structures

The `DBMS_DBFS_CONTENT` package defines `RECORD` types and `TABLE` types.

RECORD Types

- FEATURE_T Record Type
- MOUNT_T Record Type
- PATH_ITEM_T Record Type

- PROP_ITEM_T Record Type
- PROPERTY_T Record Type
- STORE_T Record Type

TABLE Types

- FEATURES_T Table Type
- MOUNTS_T Table Type
- PATH_ITEMS_T Table Type
- PROP_ITEMS_T Table Type
- PROPERTIES_T Table Type
- STORES_T Table Type

Usage Notes

There is an approximate correspondence between `DBMS_DBFS_CONTENT_PROPERTY_T` and `PROPERTY_T` — the former is a SQL object type that describes the full property tuple, while the latter is a PL/SQL record type that describes only the property value component.

Likewise, there is an approximate correspondence between `DBMS_DBFS_CONTENT_PROPERTIES_T` and `PROPERTIES_T` — the former is a SQL nested table type, while the latter is a PL/SQL hash table type.

Dynamic SQL calling conventions force the use of SQL types, but PL/SQL code may be implemented more conveniently in terms of the hash-table types.

The `DBMS_DBFS_CONTENT` interface provides convenient utility functions to convert between `DBMS_DBFS_CONTENT_PROPERTIES_T` and `PROPERTIES_T` (see propertiesT2H and propertiesH2T).

Clients can query the `DBMS_DBFS_CONTENT` interface for the list of available stores, determine which store is to handle access to a given path name, and determine the feature set for the store.

DBMS_DBFS_CONTENT FEATURE_T Record Type

This type describes a store mount point and its properties.

Syntax

```
TYPE feature_t IS RECORD (
   feature_name    VARCHAR2(32),
   feature_mask    INTEGER,
   feature_state   VARCHAR2(3));
```

Fields

Table 67-13 MOUNT_T Fields

Field	Description
`feature_name`	Name of feature
`feature_mask`	Value used to mask off all other bits other than this feature in the feature value

Table 67-13 (Cont.) MOUNT_T Fields

Field	Description
feature_state	'YES' or 'NO' depending on whether the feature is supported on this store

DBMS_DBFS_CONTENT MOUNT_T Record Type

This type describes a store mount point and its properties.

Syntax

```
TYPE mount_t IS RECORD (
   store_name          VARCHAR2(32),
   store_id            NUMBER,
   provider_name       VARCHAR2(32),
   provider_pkg        VARCHAR2(32),
   provider_id         NUMBER,
   provider_version    VARCHAR2(32),
   store_features      INTEGER,
   store_guid          NUMBER,
   store_mount         NAME_T,
   mount_properties    DBMS_DBFS_CONTENT_PROPERTIES_T);
```

Fields

Table 67-14 MOUNT_T Fields

Field	Description
store_name	Name of store
store_id	ID of store
provider_name	Name of the content store
provider_pkg	PL/SQL package name for the content store
provider_id	Unique identifier for the content store
provider_version	Version number for the content store
respos_features	Features supported by this content store
store_guid	Unique ID for this instance of the store
store_mount	Location at which this store instance is mounted
mount_properties	Properties for this mount point (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)

DBMS_DBFS_CONTENT PATH_ITEM_T Record Type

A PATH_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

Syntax

```
TYPE path_item_t IS RECORD (
   store                    NAME_T,
```

```
  mount                   NAME_T,
  pathname                PATH_T,
  pathtype                VARCHAR2(32),
  filedata                BLOB,
  std_access_time         TIMESTAMP,
  std_acl                 VARCHAR2(1024),
  std_change_time         TIMESTAMP,
  std_children            NUMBER,
  std_content_type        VARCHAR2(1024),
  std_creation_time       TIMESTAMP,
  std_deleted             INTEGER,
  std_guid                INTEGER,
  std_modification_time   TIMESTAMP,
  std_owner               VARCHAR2(32),
  std_parent_guid         INTEGER,
  std_referent            VARCHAR2(1024),
  opt_hash_type           VARCHAR2(32),
  opt_hash_value          VARCHAR2(128),
  opt_lock_count          INTEGER,
  opt_lock_data           VARCHAR2(128),
  opt_locker              VARCHAR2(128),
  opt_lock_status         INTEGER,
  opt_version             INTEGER,
  opt_version_path        PATH_T,
  opt_content_id          CONTENT_ID_T);
```

Fields

Table 67-15 PATH_ITEM_T Fields

Field	Description
`store`	Name of store
`mount`	Location at which instance of store is mounted
`pathname`	Name of path to item
`pathtype`	Type of object path (see Table 67-4)
`filedata`	`BLOB` locator `that can be` used to access data in the path item
`std_access_time`	Time of last access of a pathname's contents
`std_acl`	Access Control List (in standard ACL syntax)
`std_change_time`	Time of last change to the metadata of a path name
`std_children`	Number of child directories or folders a directory or folder path (this property should be available in providers that support the `feature_folders` feature).
`std_content_type`	One or more client-supplied mime-types (in standard RFC syntax) describing the path name which is typically of `type_file`. The content type s not necessarily interpreted by the store.
`std_creation_time`	Time at which the item was created. Once set, this value remains the same for the lifetime of the path name.
`std_deleted`	Set to a nonzero number if the path name has been soft-deleted but not yet purged (see Table 67-5)
`std_guid`	Store-specific unique identifier for a path name. Clients must not depend on the GUID being unique across different stores, but a given *store-name*, *store-specific-pathname* has a stable and unique GUID for its lifetime.

Table 67-15 (Cont.) PATH_ITEM_T Fields

Field	Description
std_modification_time	Time of last change to the data associated with a path name. Changes to the content of a type_file or type_reference path, the referent of the type_link path, and addition or deletion of immediate children in a type_directory path, all constitute data changes.
std_owner	Client-supplied (or implicit) owner name for the path name
std_parent_guid	Store-specific unique identifier for the parent of a path name. Clients must not depend on the GUID being unique across different stores, but a given *store-name*, *store-specific-pathname* has a stable and unique GUID for its lifetime. std_parent_guid(pathname) == std_guid(parent(pathname))
std_referent	Content of the symbolic link of a type_link path, otherwise NULL. As mentioned before, the std_referent can be an arbitrary string and must not necessarily be interpreted as pathname by clients (or such interpretation should be done with great care).
opt_hash_type	Type of hash provided in the opt_hash_value property (see DBMS_CRYPTO for possible options)
opt_hash_value	Hash value of type opt_hash_type describing the content of the path name
opt_lock_count	Number of compatible locks placed on a path name. If different principals are allowed to place compatible (read) locks on a path, the opt_locker must specify all lockers with repeats so that lock counts can be correctly maintained.
opt_lock_data	Client-supplied user-data associated with a user-lock, uninterpreted by the store
opt_locker	One or more implicit or client-specified principals that applied a user-lock on a path name
opt_lock_status	One of the lock_read_only, lock_write_only, lock_read_write values describing the type of lock currently applied on a path name
opt_version	Sequence number for linear versioning of a path name
opt_version_path	Version path name for hierarchical versioning of a path name
opt_content_id	Stringified provider-generated store-specific unique contentID for a file element (that may optionally not be associated with a path (see FEATURE_CONTENT_ID and FEATURE_LAZY_PATH in Table 67-5)

DBMS_DBFS_CONTENT PROP_ITEM_T Record Type

A PROP_ITEM_T is a tuple describing a (store, mount) qualified path in a store, with all user-defined properties associated with it, expanded out into individual (name, value, type) tuples.

Syntax

```
TYPE prop_item_t IS RECORD (
   store               NAME_T,
   mount                    NAME_T,
   pathname                 PATH_T,
   property_name            PROPNAME_T,
```

```
   property_value          PROPVAL_T,
   property_type           INTEGER);
```

Fields

Table 67-16 PROP_ITEM_T Fields

Field	Description
store	Name of store
mount	Location at which instance of store is mounted
pathname	Name of path to item
property_name	Name of the property
property_value	Value of the property
property_type	PL/SQL typecode for the property value

PROPERTY_T Record Type

This type describes a single (value, typecode) property value tuple; the property name is implied.

See PROPERTIES_T Table Type for more information.

Syntax

```
TYPE property_t  IS RECORD (
   propvalue    PROPVAL_T,
   typecode     INTEGER);
```

Fields

Table 67-17 PROPERTY_T Fields

Field	Description
propvalue	Value of property
typecode	Typecode

DBMS_DBFS_CONTENT STORE_T Record Type

This type describes a store registered with and managed by the DBMS_DBFS_CONTENT interface.

Syntax

```
TYPE store_t IS RECORD (
   store_name          VARCHAR2(32),
   store_id            NUMBER,
   provider_name       VARCHAR2(32),
   provider_pkg        VARCHAR2(32),
   provider_id         NUMBER,
   provider_version    VARCHAR2(32),
   store_features      INTEGER,
   store_guid          NUMBER);
```

Fields

Table 67-18 STORET_T Fields

Field	Description
store_name	Name of store
store_name	ID of store
provider_name	Name of the content store
provider_pkg	PL/SQL package name for the content store
provider_id	Unique identifier for the content store
provider_version	Version number for the content store
respos_features	Features supported by this content store
store_guid	Unique ID for this instance of the store

DBMS_DBFS_CONTENT FEATURES_T Table Type

A table type of FEATURE_T Record Type.

Syntax

```
TYPE features_t IS TABLE OF feature_t;
```

Related Topics

- DBMS_DBFS_CONTENT FEATURE_T Record Type
  This type describes a store mount point and its properties.

MOUNTS_T Table Type

A table type of MOUNT_T Record Type.

Syntax

```
TYPE mounts_t IS TABLE OF mount_t;
```

Related Topics

- DBMS_DBFS_CONTENT MOUNT_T Record Type
  This type describes a store mount point and its properties.

DBMS_DBFS_CONTENT PATH_ITEMS_T Table Type

A table type of PATH_ITEM_T Record Type

Syntax

```
TYPE path_items_t IS TABLE OF path_item_t;
```

Related Topics

- DBMS_DBFS_CONTENT PATH_ITEM_T Record Type
  A `PATH_ITEM_T` is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

DBMS_DBFS_CONTENT PROP_ITEMS_T Table Type

A table type of PATH_ITEM_T Record Type.

Syntax

```
TYPE prop_items_t IS TABLE OF prop_item_t;
```

Related Topics

- DBMS_DBFS_CONTENT PATH_ITEM_T Record Type
  A `PATH_ITEM_T` is a tuple describing a (store, mount) qualified path in a store, with all standard and optional properties associated with it.

DBMS_DBFS_CONTENT PROPERTIES_T Table Type

This is a name-indexed hash table of property tuples. The implicit hash-table association between the index and the value allows the client to build up the full `DBMS_DBFS_CONTENT_PROPERTY_T` tuples for a `PROPERTIES_T`.

Syntax

```
TYPE properties_t IS TABLE OF property_t INDEX BY propname_t;
```

STORES_T Table Type

This type describes a store registered with and managed by the `DBMS_DBFS_CONTENT` interface.

Syntax

```
TYPE stores_t IS TABLE OF store_t;
```

Summary of DBMS_DBFS_CONTENT Subprograms

This table lists and describes the subprograms used in the `DBMS_DBFS_CONTENT` Package.

Table 67-19 DBMS_DBFS_CONTENT Package Subprograms

Subprogram	Description
CHECKACCESS Function	Reports if the user (`principal`) can perform the specified operation on the given path
CHECKSPI Functions and Procedures	Checks if a user-provided package implements all of the `DBMS_DBFS_CONTENT_SPI` subprograms with the proper signatures, and reports on the conformance.
CREATEDIRECTORY Procedures	Creates a directory
CREATEFILE Procedures	Creates a file
CREATELINK Procedures	Creates a new reference to the source file system element
CREATEREFERENCE Procedures	Creates a physical link to an already existing file system element
DECODEFEATURES Function	Given a feature bit set integer value, returns a `FEATURES_T` table of the feature bits as `FEATURE_T` records
DELETECONTENT Procedure	Deletes the file specified by the given contentID

Table 67-19 (Cont.) DBMS_DBFS_CONTENT Package Subprograms

Subprogram	Description
DELETEDIRECTORY Procedure	Deletes a directory
DELETEFILE Procedure	Deletes a file
FEATURENAME Function	Given a feature bit, returns a `VARCHAR2` of that feature's name
FLUSHSTATS Function	Flushes `DBMS_DBFS_CONTENT` statistics to disk
GETDEFAULTACL Procedure	Returns the ACL parameter of the default context
GETDEFAULTASOF Procedure	Returns the `asof` parameter of the default context
GETTDEFAULTCONTEXT Procedure	Returns the default context
GETDEFAULTOWNER Procedure	Returns the `owner` parameter of the default context
GETDEFAULTPRINCIPAL Procedure	Returns the `principal` parameter of the default context
GETFEATURESBYMOUNT Function	Returns features of a store by mount point
GETFEATURESBYNAME Function	Returns features of a store by store name
GETFEATURESBYPATH Function	Returns features of a store by path
GETPATHBYMOUNTID Function	Returns the full absolute path name
GETPATH Procedures	Returns existing path items (such as files and directories)
GETPATHBYSTOREID Function	If the underlying GUID is found in the underlying store, returns the store-qualified path name
GETPATHNOWAIT Procedures	Implies that the operation is for an update, and, if implemented, allows providers to return an exception (`ORA-00054`) rather than wait for row locks.
GETSTOREBYMOUNT Function	Returns a store by way of its mount point
GETSTOREBYNAME Function	Returns a store by way of its name
GETSTOREBYPATH Function	Returns a store by way of its path
GETSTATS Procedure	Returns information about `DBMS_DBFS_CONTENT` statistics collection
GETTRACE Function	Returns whether or not `DBMS_DBFS_CONTENT` tracing is turned on
GETVERSION Function	Returns the version of the `DBMS_DBFS_CONTENT` interface in a standardized format associated with a store
LIST Function	Lists the path items in the specified path meeting the specified filter and other criteria
LISTALLCONTENT Function	Lists all path items in all mounts
LISTALLPROPERTIES Function	Returns a table of all properties for all path items in all mounts
LISTMOUNTS Function	Lists all available mount points, their backing stores, and the store features
LISTSTORES Function	Lists all available stores and their features
LOCKPATH Procedure	Applies user-level locks to the given valid path name
MOUNTSTORE Procedure	Mounts a previously registered store and binds it to the mount point
NORMALIZEPATH Functions	Converts a store-specific or full-absolute path name into normalized form

Table 67-19 (Cont.) DBMS_DBFS_CONTENT Package Subprograms

Subprogram	Description
PROPANY Functions	Provides constructors that take one of a variety of types and return a PROPERTY_T
PROPERTIESH2T Function	Converts a `PROPERTY_T` hash to a `DBMS_DBFS_CONTENT_PROPERTIES_T` table
PROPERTIEST2H Function	Converts a `DBMS_DBFS_CONTENT_PROPERTIES_T` table to a `PROPERTY_T` hash
PROPNUMBER Function	Is a constructor that takes a `NUMBER` and returns a `PROPERTY_T`
PROPRAW Function	Is a constructor that takes a `RAW` and returns a `PROPERTY_T`
PROPTIMESTAMP Function	Is a constructor that takes a `TIMESTAMP` and returns a `PROPERTY_T`
PROPVARCHAR2 Function	Is a constructor that takes a `VARCAHR2` and returns a `PROPERTY_T`
PURGEALL Procedure	Purges all soft-deleted entries matching the path and optional filter criteria
PURGEPATH Procedure	Purges any soft-deleted versions of the given path item
PUTPATH Procedures	Creates a new path item
REGISTERSTORE Procedure	Registers a new store
RENAMEPATH Procedures	Renames or moves a path
RESTOREALL Procedure	Restores all soft-deleted path items meeting the path and filter criteria
RESTOREPATH Procedure	Restores all soft-deleted path items that match the given path and filter criteria
SETDEFAULTACL Procedure	Sets the ACL parameter of the default context
SETDEFAULTASOF Procedure	Sets the "as of" parameter of the default context
SETDEFAULTCONTEXT Procedure	Sets the default context
SETDEFAULTOWNER Procedure	Sets the "owner" parameter of the default context
SETDEFAULTPRINCIPAL Procedure	Sets the "principal" parameter of the default context
SETPATH Procedures	Assigns a path name to a path item represented by contentID
SETSTATS Procedure	Enables and disables statistics collection
SETTRACE Procedure	Sets `DBMS_DBFS_CONTENT` tracing on or off
SPACEUSAGE Procedure	Queries file system space usage statistics
TRACE Procedure	Returns a `CLOB` that contains the evaluation results
TRACEENABLED Function	Determines if the current trace "severity" set by the SETTRACE Procedure is at least as high as the given trace level
UNLOCKPATH Procedure	Unlocks path items that were previously locked with the LOCKPATH Procedure
UNMOUNTSTORE Procedure	Unmounts a registered store
UNREGISTERSTORE Procedure	Unregisters a store

CHECKACCESS Function

This function reports if the user (`principal`) can perform the specified operation on the given path. This enables verifying the validity of an operation without attempting to perform the operation. If `CHECKACCESS` returns 0, then the subprogram invoked to implement that operation should fail with an error.

Syntax

```
DBMS_DBFS_CONTENT.CHECKACCESS (
   path           IN     VARCHAR2,
   pathtype       IN     INTEGER,
   operation      IN     VARCHAR2,
   principal      IN     VARCHAR2,
   store_name     IN     VARCHAR2  DEFAULT NULL)
  RETURN  BOOLEAN;
```

Parameters

Table 67-20 CHECKACCESS Procedure Parameters

Parameter	Description
`path`	Name of path to check for access
`pathtype`	Type of object `path` represents (see Table 67-4)
`operation`	Operation to be checked (see Table 67-8)
`principal`	File system user for whom the access check is made
`store_name`	Name of store

Usage Notes

Whether or not the user invokes this function, a store that supports access control internally performs these checks to guarantee security.

CHECKSPI Functions and Procedures

Given the name of a putative `DBMS_DBFS_CONTENT_SPI` conforming package, this function or procedure checks whether the package implements all of the provider subprograms with the proper signatures, and reports on the conformance.

Syntax

```
DBMS_DBFS_CONTENT.CHECKSPI (
   package_name        IN              VARCHAR2)
  RETURN  CLOB;

DBMS_DBFS_CONTENT.CHECKSPI (
   schema_name         IN              VARCHAR2,
   package_name        IN              VARCHAR2)
  return  clob;

DBMS_DBFS_CONTENT.CHECKSPI (
   package_name        IN              VARCHAR2,
   chk                 IN OUT NOCOPY   CLOB);
```

```
DBMS_DBFS_CONTENT.CHECKSPI (
   schema_name         in              VARCHAR2,
   package_name        in              VARCHAR2,
   chk                 in out nocopy   CLOB);
```

Parameters

Table 67-21 CHECKSPI Procedure Parameters

Parameter	Description
package_name	Name of package
schema_name	Name of schema
chk	CLOB that contains the evaluation results

Usage Notes

- The functional form returns a cached temporary LOB of session duration with the results of the analysis. The caller is expected to manage the lifetime of this LOB, as needed.
- The procedural form generates the results of the analysis into the chk LOB parameter; if the value passed in is NULL, the results are written to the foreground trace file provided that DBMS_DBFS_CONTENT interface tracing is enabled. If neither tracing is enabled nor a valid LOB passed in, the checker does not provide any useful indication of the analysis (other than raise exceptions if it encounters a serious error).
- If schema_name is NULL, standard name resolution rules (current schema, private synonym, public synonym) are used to try and locate a suitable package to analyze.

CREATEDIRECTORY Procedures

This procedure creates a directory.

Syntax

```
DBMS_DBFS_CONTENT.CREATEDIRECTORY (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   recurse     IN              BOOLEAN     DEFAULT FALSE,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATEDIRECTORY (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   recurse     IN              BOOLEAN     DEFAULT FALSE,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-22 CREATEDIRECTORY Procedure Parameters

Parameter	Description
path	Name of path to the directory

Table 67-22 (Cont.) CREATEDIRECTORY Procedure Parameters

Parameter	Description
properties	One or more properties and their values to be set, returned, or both, depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting PROP_SPC (see Table 67-9), and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
recurse	If 0, do not execute recursively; otherwise, recursively create the directories above the given directory
store_name	Name of store
principal	File system user for whom the access check is made

CREATEFILE Procedures

This procedure creates a file.

Syntax

```
DBMS_DBFS_CONTENT.CREATEFILE (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   content     IN OUT NOCOPY   BLOB,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD + PROP_DATA),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATEFILE (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   content     IN OUT NOCOPY   BLOB,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD + PROP_DATA),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-23 CREATEFILE Procedure Parameters

Parameter	Description
path	Name of path to the file
properties	One or more properties and their values to be set, returned, or both, depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
content	BLOB holding data with which to populate the file (optional)

Table 67-23 (Cont.) CREATEFILE Procedure Parameters

Parameter	Description
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
store_name	Name of store
principal	File system user for whom the access check is made

CREATELINK Procedures

This procedure creates a new link element srcPath with the value of dstPath. The value of dstPath is not validated or interpreted in any way by this procedure. This is analogous to a UNIX file system symbolic link.

Syntax

```
DBMS_DBFS_CONTENT.CREATELINK (
   srcPath     IN              VARCHAR2,
   dstPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATELINK (
   srcPath     IN              VARCHAR2,
   dstPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-24 CREATELINK Procedure Parameters

Parameter	Description
srcPath	File system entry to create.
dstPath	Value to associate with srcPath.
properties	One or more properties and their values to be set, returned depending, or both, on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
store_name	Name of store
principal	File system user for whom the access check is made

CREATEREFERENCE Procedures

This procedure creates a physical link, `srcPath`, to an already existing file system element, `dstPath` (such as file or directory). The resulting entry shares the same metadata structures as the value of the `dstPath` parameter, and so is similar to incrementing a reference count on the file system element. This is analogous to a UNIX file system hard link.

Syntax

```
DBMS_DBFS_CONTENT.CREATEREFERENCE (
   srcPath     IN              VARCHAR2,
   dstPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.CREATEREFERENCE (
   srcPath     IN              VARCHAR2,
   dstPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   prop_flags  IN              INTEGER     DEFAULT PROP_STD,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-25 CREATEREFERENCE Procedure Parameters

Parameter	Description
`srcPath`	File system entry to create.
`dstPath`	Path that is the reference to `srcPath`.
`properties`	One or more properties and their values to be set, returned, or both, depending on `prop_flags` (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
`prop_flags`	Determines which properties are set, returned. Default is `PROP_STD`. Specify properties to be returned by setting `prop_spec`, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
`store_name`	Name of store
`principal`	File system user for whom the access check is made

DECODEFEATURES Function

Given a feature bit set integer value, this function returns a `FEATURES_T` table of the feature bits as `FEATURE_T` records.

Syntax

```
DBMS_DBFS_CONTENT.DECODEFEATURES (
   featureSet          IN      INTEGER)
  RETURN FEATURES_T DETERMINISTIC PIPELINED;
```

Parameters

Table 67-26 DECODEFEATURES Function Parameters

Parameter	Description
featureSet	Feature set

Return Values

FEATURES_T Table Type

DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

Syntax

```
DBMS_DBFS_CONTENT.DELETECONTENT (
   store_name     IN     VARCHAR2    DEFAULT NULL,
   contentID      IN     RAW,
   filter         IN     VARCHAR2    DEFAULT NULL,
   soft_delete    IN     BOOLEAN     DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-27 DELETECONTENT Procedure Parameters

Parameter	Description
store_name	Name of store
contentID	Unique identifier for the file to be deleted
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see *Oracle Database SecureFiles and Large Objects Developer's Guide*, Deletion Operations).
principal	File system user for whom the access check is made

DELETEDIRECTORY Procedure

This procedure deletes a directory.

If recurse is nonzero, it recursively deletes all elements of the directory. A filter, if supplied, determines which elements of the directory are deleted.

Syntax

```
DBMS_DBFS_CONTENT.DELETEDIRECTORY (
   path           IN     VARCHAR2,
   filter         IN     VARCHAR2    DEFAULT NULL,
   soft_delete    IN     BOOLEAN     DEFAULT NULL,
   recurse        IN     BOOLEAN     DEFAULT FALSE,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-28 DELETEDIRECTORY Procedure Parameters

Parameter	Description
path	Name of path to the directory
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete see *Oracle Database SecureFiles and Large Objects Developer's Guide*, Deletion Operations.
recurse	If 0, do not execute recursively. Otherwise, recursively delete the directories and files below the given directory.
store_name	Name of store
principal	File system user for whom the access check is made

DELETEFILE Procedure

This procedure deletes the specified file.

Syntax

```
DBMS_DBFS_CONTENT.DELETEFILE (
   path           IN     VARCHAR2,
   filter         IN     VARCHAR2    DEFAULT NULL,
   soft_delete    IN     BOOLEAN     DEFAULT NULL,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-29 DELETEFILE Procedure Parameters

Parameter	Description
path	Name of path to the file
filter	A filter, if any, to be applied
soft_delete	If 0, execute a hard (permanent) delete. For any value other than 0, perform a soft delete (see *Oracle Database SecureFiles and Large Objects Developer's Guide*, Deletion Operations).
store_name	Name of store
principal	File system user for whom the access check is made

FEATURENAME Function

Given a feature bit, this function returns a `VARCHAR2` of that feature's name.

Syntax

```
DBMS_DBFS_CONTENT.FEATURENAME (
   featureBit          IN      INTEGER)
  RETURN VARCHAR2 DETERMINISTIC;
```

Parameters

Table 67-30 FEATURENAME Function Parameters

Parameter	Description
featureBit	Bit representation of the feature (see Table 67-5)

Return Values

Name of the feature

FLUSHSTATS Function

This procedure flushes DBMS_DBFS_CONTENT statistics to disk.

Syntax

```
DBMS_DBFS_CONTENT.FLUSHSTATS;
```

GETDEFAULTACL Procedure

This procedure returns the ACL parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTACL (
   acl    OUT NOCOPY     VARCHAR2);
```

Parameters

Table 67-31 GETDEFAULTACL Procedure Parameters

Parameter	Description
acl	ACL for all new elements created (implicitly or explicitly) by the current operation

GETDEFAULTASOF Procedure

This procedure returns the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTASOF (
  asof    OUT NOCOPY     TIMESTAMP);
```

Parameters

Table 67-32 GETDEFAULTASOF Procedure Parameters

Parameter	Description
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

GETDEFAULTCONTEXT Procedure

This procedure returns the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTCONTEXT (
   principal    OUT NOCOPY     VARCHAR2,
   owner        OUT NOCOPY     VARCHAR2,
   acl          OUT NOCOPY     VARCHAR2,
   asof         OUT NOCOPY     TIMESTAMP);
```

Parameters

Table 67-33 GETDEFAULTCONTEXT Procedure Parameters

Parameter	Description
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

GETDEFAULTOWNER Procedure

This procedure returns the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTOWNER (
   principal    IN     VARCHAR2);
```

Parameters

Table 67-34 GETDEFAULTOWNER Procedure Parameters

Parameter	Description
owner	Owner for new elements created (implicitly or explicitly) by the current operation

GETDEFAULTPRINCIPAL Procedure

This procedure returns the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.GETDEFAULTPRINCIPAL (
   principal    OUT NOCOPY     VARCHAR2);
```

Parameters

Table 67-35 GETDEFAULTPRINCIPAL Procedure Parameters

Parameter	Description
principal	Agent (principal) invoking the current operation

GETFEATURESBYMOUNT Function

This function returns features of a store by mount point.

Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYMOUNT (
   store_mount       IN      VARCHAR2)
  RETURN INTEGER;
```

Parameters

Table 67-36 GETFEATURESBYMOUNT Function Parameters

Parameter	Description
store_mount	Mount point

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

GETFEATURESBYNAME Function

This function returns features of a store by store name.

Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYNAME (
   store_name       IN      VARCHAR2)
  RETURN INTEGER;
```

Parameters

Table 67-37 GETFEATURESBYNAME Function Parameters

Parameter	Description
store_name	Name of store

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

GETFEATURESBYPATH Function

This function returns features of a store by path.

Syntax

```
DBMS_DBFS_CONTENT.GETFEATURESBYPATH (
   path       IN      PATH_T)
  RETURN INTEGER;
```

Parameters

Table 67-38 GETFEATURESBYPATH Function Parameters

Parameter	Description
path	PATH_T

Return Values

A bit mask of supported features (see FEATURES_T Table Type)

GETPATH Procedures

This procedure returns existing path items (such as files and directories). This includes both data and metadata (properties).

The client can request (using `prop_flags`) that specific properties be returned. File path names can be read either by specifying a `BLOB` locator using the `prop_data` bitmask in `prop_flags` (see Table 67-9) or by passing one or more `RAW` buffers.

When `forUpdate` is 0, this procedure also accepts a valid `asof` timestamp parameter as part of `ctx` that can be used by stores to implement "as of" style flashback queries. Mutating versions of the `GETPATH` Procedures do not support these modes of operation.

Syntax

```
DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
        properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
        content     OUT    NOCOPY   BLOB,
        item_type   OUT             INTEGER,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        forUpdate   IN              BOOLEAN     DEFAULT FALSE,
        deref       IN              BOOLEAN     DEFAULT FALSE,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
        properties  IN OUT NOCOPY   PROPERTIES_T,
        content     OUT    NOCOPY   BLOB,
        item_type   OUT             INTEGER,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        forUpdate   IN              BOOLEAN     DEFAULT FALSE,
        deref       IN              BOOLEAN     DEFAULT FALSE,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
        properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
        amount      IN OUT          NUMBER,
        offset      IN              NUMBER,
        buffers     OUT    NOCOPY   RAW,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
        properties  IN OUT NOCOPY   PROPERTIES_T,
        amount      IN OUT          NUMBER,
        offset      IN              NUMBER,
        buffers     OUT    NOCOPY   RAW,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
```

```
        properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
        amount      IN OUT          NUMBER,
        offset      IN              NUMBER,
        buffers     OUT    NOCOPY   DBMS_DBFS_CONTENT_RAW_T,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATH (
        path        IN              VARCHAR2,
        properties  IN OUT NOCOPY   PROPERTIES_T,
        amount      IN OUT          NUMBER,
        offset      IN              NUMBER,
        buffers     OUT    NOCOPY   DBMS_DBFS_CONTENT_RAW_T,
        prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                         PROP_OPT +
                                                         PROP_DATA),
        asof        IN              TIMESTAMP   DEFAULT NULL,
        store_name  IN              VARCHAR2    DEFAULT NULL,
        principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-39 GETPATH Procedure Parameters

Parameter	Description
path	Name of path to path items
properties	One or more properties and their values to be returned depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see Table 67-4)
amount	On input, number of bytes to be read. On output, number of bytes read
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write
prop_flags	Determines which properties are set, returned, or both. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
forUpdate	Specifies that a lock should be taken to signify exclusive write access to the path item
deref	If nonzero, attempts to resolve the given path item to actual data provided it is a reference
store_name	Name of store
principal	Agent (principal) invoking the current operation

GETPATHBYMOUNTID Function

If the underlying GUID is found in the underlying store, this function returns the full absolute path name.

Syntax

```
DBMS_DBFS_CONTENT.GETPATHBYMOUNTID (
   store_mount          IN      VARCHAR2,
   guid                 IN      INTEGER)
  RETURN VARCHAR2;
```

Parameters

Table 67-40 GETPATHBYMOUNTID Function Parameters

Parameter	Description
`store_mount`	Mount point in which the path item with `guid` resides
`guid`	Unique ID for the path item

Usage Notes

If the GUID is unknown, a `NULL` value is returned. Clients are expected to handle this as appropriate.

Return Values

Path of the path item represented by GUID in `store_mount`

GETPATHBYSTOREID Function

If the underlying GUID is found in the underlying store, this function returns the store-qualified path name.

Syntax

```
DBMS_DBFS_CONTENT.GETPATHBYSTOREID (
   store_name           IN      VARCHAR2,
   guid                 IN      INTEGER)
  RETURN VARCHAR2;
```

Parameters

Table 67-41 GETPATHBYSTOREID Function Parameters

Parameter	Description
`store_name`	Name of store
`guid`	Unique ID representing the desired path item

Usage Notes

If the GUID is unknown, a `NULL` value is returned. Clients are expected to handle this as appropriate.

Return Values

Store-qualified path name represented by the GUID

GETPATHNOWAIT Procedures

This procedure implies that the operation is for an update, and, if implemented, allows providers to return an exception (`ORA-00054`) rather than wait for row locks.

See `FEATURE_NOWAIT` in Table 67-5 for more information.

Syntax

```
DBMS_DBFS_CONTENT.GETPATHNOWAIT (
       path        IN              VARCHAR2,
       properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
       content     OUT    NOCOPY   BLOB,
       item_type   OUT             INTEGER,
       prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                        PROP_OPT +
                                                        PROP_DATA),
       deref       IN              BOOLEAN     DEFAULT FALSE,
       store_name  IN              VARCHAR2    DEFAULT NULL,
       principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.GETPATHNOWAIT (
       path        IN              VARCHAR2,
       properties  IN OUT NOCOPY   PROPERTIES_T,
       content     OUT    NOCOPY   BLOB,
       item_type   OUT             INTEGER,
       prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                        PROP_OPT +
                                                        PROP_DATA),
       deref       IN              BOOLEAN     DEFAULT FALSE,
       store_name  IN              VARCHAR2    DEFAULT NULL,
       principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-42 GETPATHNOWAIT Procedure Parameters

Parameter	Description
`path`	Name of path to path items
`properties`	One or more properties and their values to be returned depending on `prop_flags` (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
`content`	`BLOB` holding data which populates the file (optional)
`item_type`	Type of the path item specified (see Table 67-4)
`prop_flags`	Determines which properties are returned. Default is `PROP_STD`. Specify properties to be returned by setting `prop_spec`, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
`asof`	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

Table 67-42 (Cont.) GETPATHNOWAIT Procedure Parameters

Parameter	Description
deref	If nonzero, attempts to resolve the given path item to actual data provided it is a reference
store_name	Name of store
principal	Agent (principal) invoking the current operation

GETSTOREBYMOUNT Function

This function returns a store by way of its name.

Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYMOUNT (
   store_mount       IN      VARCHAR2)
  RETURN STORE_T;
```

Parameters

Table 67-43 GETSTOREBYMOUNT Function Parameters

Parameter	Description
store_mount	Location at which the store instance is mounted

Return Values

STORE_T Record Type

GETSTOREBYNAME Function

This function returns a store by way of its name.

Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYNAME (
   store_name       IN      VARCHAR2)
  RETURN STORE_T;
```

Parameters

Table 67-44 GETSTOREBYNAME Function Parameters

Parameter	Description
store_name	Name of store

Return Values

STORE_T Record Type

GETSTOREBYPATH Function

This function returns a store by way of its path.

Syntax

```
DBMS_DBFS_CONTENT.GETSTOREBYPATH (
   path       IN      PATH_T)
  RETURN STORE_T;
```

Parameters

Table 67-45 GETSTOREBYPATH Function Parameters

Parameter	Description
path	PATH_T s

Return Values

STORE_T Record Type

GETSTATS Procedure

This procedure returns information about DBMS_DBFS_CONTENT statistics collection.

Syntax

```
DBMS_DBFS_CONTENT.GETSTATS (
   enabled         OUT    BOOLEAN,
   flush_time      OUT    INTEGER,
   flush_count     OUT    INTEGER);
```

Parameters

Table 67-46 GETSTATS Procedure Parameters

Parameter	Description
enabled	Whether statistics collection is enabled
flush_time	How often to flush the statistics to disk in centiseconds
flush_count	Number of operations to allow between statistics flushes

GETTRACE Function

This function returns whether DBMS_DBFS_CONTENT tracing is turned on or not.

Syntax

```
DBMS_DBFS_CONTENT.GETTRACE
  RETURN INTEGER.
```

Return Values

Returns zero if tracing is off, non-zero if tracing is on.

GETVERSION Function

This function marks each version of the `DBMS_DBFS_CONTENT` interface.

Syntax

```
DBMS_DBFS_CONTENT.GETVERSION (
  RETURN VARCHAR2;
```

Return Values

A string enumerating the version of the `DBMS_DBFS_CONTENT` interface in standard naming convention: string: *a.b.c* corresponding to *major*, *minor*, and *patch* components.

ISPATHLOCKED Procedure

This procedure checks if any user-level locks are applied on a given path.

Syntax

```
DBMS_DBFS_CONTENT.ISPATHLOCKED (
   path           IN     VARCHAR2,
   who            IN     VARCHAR2,
   lock_type      IN OUT INTEGER,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-47 ISPATHLOCKED Procedure Parameters

Parameter	Description
`path`	Name of path to file items
`who`	Transaction identifier that has locked the path
`lock_type`	One of the available lock types (see Table 67-6)
`store_name`	Name of store
`principal`	Agent (principal) invoking the current operation

LIST Function

This function lists the path items in the specified path meeting the specified filter and other criteria.

Syntax

```
DBMS_DBFS_CONTENT.LIST (
   path           IN     VARCHAR2,
   filter         IN     VARCHAR2    DEFAULT NULL,
   recurse        IN     INTEGER     DEFAULT 0,
   asof           IN     TIMESTAMP   DEFAULT NULL,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL)
  RETURN  DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

Parameters

Table 67-48 LIST Function Parameters

Parameter	Description
path	Name of path to directories
filter	A filter, if any, to be applied
recurse	If 0, do not execute recursively. Otherwise, recursively list the contents of directories and files below the given directory.
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
store_name	Name of repository
principal	Agent (principal) invoking the current operation

Return Values

DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type

LISTALLPROPERTIES Function

This function returns a table of all properties for all path items in all mounts.

Syntax

```
DBMS_DBFS_CONTENT.LISTALLPROPERTIES
  RETURN  PROP_ITEMS_T PIPELINED;
```

Return Values

PROP_ITEMS_T Table Type

LISTALLCONTENT Function

This function lists all path items in all mounts.

Syntax

```
DBMS_DBFS_CONTENT.LISTALLCONTENT
  RETURN  PATH_ITEMS_T PIPELINED;
```

Return Values

PATH_ITEMS_T Table Type

LISTMOUNTS Function

This function lists all available mount points, their backing stores, and the store features.

Syntax

```
DBMS_DBFS_CONTENT.LISTMOUNTS
  RETURN MOUNTS_T PIPELINED;
```

Return Values

MOUNTS_T Table Type

Usage Notes

A single mount results in a single returned row, with its `store_mount` field of the returned records set to `NULL`.

LISTSTORES Function

This function lists all available stores and their features.

Syntax

```
DBMS_DBFS_CONTENT.LISTSTORES
  RETURN STORES_T PIPELINED;
```

Return Values

STORES_T Table Type

Usage Notes

The `store_mount` field of the returned records is set to `NULL` (since mount-points are separate from stores themselves).

LOCKPATH Procedure

This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

Syntax

```
DBMS_DBFS_CONTENT.LOCKPATH (
   path           IN     VARCHAR2,
   who            IN     VARCHAR2,
   lock_type      IN     INTEGER,
   waitForRowLock IN     INTEGER     DEFAULT 1,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-49 LOCKPATH Procedure Parameters

Parameter	Description
`path`	Name of path to file items
`who`	Transaction identifier that has locked the path
`lock_type`	One of the available lock types (see Table 67-6)
`waitForRowLock`	Determines if a row is locked by a transaction or not
`store_name`	Name of store
`principal`	Agent (principal) invoking the current operation

MOUNTSTORE Procedure

This procedure mounts a previously registered store and binds it to the mount point.

Syntax

```
DBMS_DBFS_CONTENT.MOUNTSTORE (
   store_mount     in      VARCHAR2   DEFAULT NULL,
   singleton       in      BOOLEAN    DEFAULT FALSE,
   principal       in      VARCHAR2   DEFAULT NULL,
   owner           in      VARCHAR2   DEFAULT NULL,
   acl             in      VARCHAR2   DEFAULT NULL,
   asof            in      TIMESTAMP  DEFAULT NULL,
   read_only       in      BOOLEAN    DEFAULT FALSE);
```

Parameters

Table 67-50 MOUNTSTORE Procedure Parameters

Parameter	Description
store_mount	Path name to use to mount this store
singleton	Whether the mount is a single backend store on the system
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes
read_only	Whether the mount is read-only

Usage Notes

> **See Also:**
>
> *Oracle Database SecureFiles and Large Objects Developer's Guide* for information on mounting a registered store

NORMALIZEPATH Functions

This function converts a store-specific or full-absolute path name into normalized form.

It does the following:

- verifies that the path name is absolute, and so starts with "/"
- collapses multiple consecutive "/" into a single "/"
- strips trailing "/"
- breaks up a store-specific normalized path name into 2 components - parent pathname, trailing component name

- breaks up a full-absolute normalized path name into 3 components - store name, parent pathname, trailing component name

Syntax

```
DBMS_DBFS_CONTENT.NORMALIZEPATH (
   path        IN              VARCHAR2,
   parent      OUT NOCOPY      VARCHAR2,
   tpath       OUT NOCOPY      VARCHAR2)
  RETURN VARCHAR2;

DBMS_DBFS_CONTENT.NORMALIZEPATH (
   path        IN              VARCHAR2,
   store_name  OUT NOCOPY      VARCHAR2,
   parent      OUT NOCOPY      VARCHAR2,
   tpath       OUT NOCOPY      VARCHAR2)
  RETURN VARCHAR2;
```

Parameters

Table 67-51 NORMALIZEPATH Function Parameters

Parameter	Description
path	Name of path to file items
store_name	Name of store
parent	Parent path name
tpath	Name of trailing path item

Return Values

The completely normalized store-specific or full-absolute path name

PROPANY Functions

This function provides constructors that take one of a variety of types and return a PROPERTY_T.

Syntax

```
DBMS_DBFS_CONTENT.PROPANY (
   val      IN      NUMBER)
RETURN PROPERTY_T;

DBMS_DBFS_CONTENT.PROPANY (
   val      IN      VARCHAR2)
RETURN PROPERTY_T;

DBMS_DBFS_CONTENT.PROPANY (
   val      IN      TIMESTAMP)
RETURN PROPERTY_T;

DBMS_DBFS_CONTENT.PROPANY (
   val      IN      RAW)
RETURN PROPERTY_T;
```

Parameters

Table 67-52 PROPANY Function Parameters

Parameter	Description
val	Value

Return Values

PROPERTY_T Record Type

PROPERTIESH2T Function

This function converts a PROPERTY_T hash to a DBMS_DBFS_CONTENT_PROPERTIES_T table.

Syntax

```
DBMS_DBFS_CONTENT.PROPERTIEST2H (
   pprops      IN      PROPERTIES_T)
RETURN DBMS_DBFS_CONTENT_PROPERTIES_T;
```

Parameters

Table 67-53 PROPERTIEST2H Function Parameters

Parameter	Description
sprops	A PROPERTIES_T hash

Return Values

DBMS_DBFS_CONTENT_PROPERTIES_T Table Type

PROPERTIEST2H Function

This function converts a DBMS_DBFS_CONTENT_PROPERTIES_T table to a PROPERTY_T hash.

Syntax

```
DBMS_DBFS_CONTENT.PROPERTIEST2H (
   sprops      IN      DBMS_DBFS_CONTENT_PROPERTIES_T)
RETURN properties_t;
```

Parameters

Table 67-54 PROPERTIEST2H Function Parameters

Parameter	Description
sprops	A DBMS_DBFS_CONTENT_PROPERTIES_T table

Return Values

PROPERTIES_T Table Type

PROPNUMBER Function

This function is a constructor that takes a number and returns a `PROPERTY_T`.

Syntax

```
DBMS_DBFS_CONTENT.PROPNUMBER (
   val      IN      NUMBER)
RETURN PROPERTY_T;
```

Parameters

Table 67-55 PROPNUMBER Function Parameters

Parameter	Description
`val`	Value

Return Values

PROPERTY_T Record Type

PROPRAW Function

This function is a constructor that takes a `RAW` and returns a `PROPERTY_T`.

Syntax

```
DBMS_DBFS_CONTENT.PROPRAW (
   val      IN      RAW)
RETURN PROPERTY_T;
```

Parameters

Table 67-56 PROPRAW Function Parameters

Parameter	Description
`val`	Value

Return Values

PROPERTY_T Record Type

PROPTIMESTAMP Function

This function is a constructor that takes a `TIMESTAMP` and returns a `PROPERTY_T`.

Syntax

```
DBMS_DBFS_CONTENT.PROPTIMESTAMP (
   val      IN      TIMESTAMP)
RETURN PROPERTY_T;
```

Parameters

Table 67-57 PROPTIMESTAMP Function Parameters

Parameter	Description
val	Value

Return Values

PROPERTY_T Record Type

PROPVARCHAR2 Function

This function is a constructor that takes a VARCHAR2 and returns a PROPERTY_T.

Syntax

```
DBMS_DBFS_CONTENT.PROPVARCHAR2 (
   val      IN      VARCHAR2)
RETURN PROPERTY_T;
```

Parameters

Table 67-58 PROPNUMBER Function Parameters

Parameter	Description
val	Value

Return Values

PROPERTY_T Record Type

PURGEALL Procedure

This procedure purges all soft-deleted entries matching the path and optional filter criteria.

Syntax

```
DBMS_DBFS_CONTENT.PURGEALL (
   path           IN      VARCHAR2,
   filter         IN      VARCHAR2    DEFAULT NULL,
   store_name     IN      VARCHAR2    DEFAULT NULL,
   principal      IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-59 PURGEALL Procedure Parameters

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied based on specified criteria
store_name	Name of store

Table 67-59 (Cont.) PURGEALL Procedure Parameters

Parameter	Description
principal	Agent (principal) invoking the current operation

PURGEPATH Procedure

This procedure purges any soft-deleted versions of the given path item.

Syntax

```
DBMS_DBFS_CONTENT.PURGEPATH (
   path           IN      VARCHAR2,
   filter         IN      VARCHAR2    DEFAULT NULL,
   store_name     IN      VARCHAR2    DEFAULT NULL,
   principal      IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-60 PURGEPATH Procedure Parameters

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
store_name	Name of store
principal	Agent (principal) invoking the current operation

PUTPATH Procedures

This procedure creates a new path item.

Syntax

```
DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   content     IN OUT NOCOPY   BLOB,
   item_type   OUT             INTEGER,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT +
                                                    PROP_DATA),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   content     IN OUT NOCOPY   BLOB,
   item_type   OUT             INTEGER,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT +
                                                    PROP_DATA),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

```
DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   amount      IN              NUMBER,
   offset      IN              NUMBER,
   buffer      IN              RAW,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   amount      IN              NUMBER,
   offset      IN              NUMBER,
   buffer      IN              RAW,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   written     OUT             NUMBER,
   offset      IN              NUMBER,
   buffers     IN             DBMS_DBFS_CONTENT_RAW_T,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.PUTPATH (
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   written     OUT             NUMBER,
   offset      IN              NUMBER,
   buffers     IN              DBMS_DBFS_CONTENT_RAW_T,
   prop_flags  IN              INTEGER     DEFAULT (PROP_STD +
                                                    PROP_OPT),
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-61 PUTPATH Procedure Parameters

Parameter	Description
path	Name of path to file items
properties	One or more properties and their values to be set depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
content	BLOB holding data which populates the file (optional)
item_type	Type of the path item specified (see Table 67-4)
amount	Number of bytes to be read

Table 67-61 (Cont.) PUTPATH Procedure Parameters

Parameter	Description
offset	Byte offset from which to begin reading
buffer	Buffer to which to write
buffers	Buffers to which to write
prop_flags	Determines which properties are set. Default is PROP_STD. Specify properties to be returned by setting prop_spec, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
store_name	Name of store
principal	Agent (principal) invoking the current operation

REGISTERSTORE Procedure

This procedure registers a new store backed by a provider that uses a store provider (conforming to the DBMS_DBFS_CONTENT_SPI package signature).

This method is to be used primarily by store providers after they have created a new store.

Syntax

```
DBMS_DBFS_CONTENT.REGISTERSTORE (
   store_name          IN      VARCHAR2,
   provider_name       IN      VARCHAR2,
   provider_package    IN      VARCHAR2);
```

Parameters

Table 67-62 REGISTERSTORE Procedure Parameters

Parameter	Description
store_name	Name of store, must be unique
provider_name	Name of provider
provider_package	Store provider

RENAMEPATH Procedures

This procedure renames or moves a path. This operation can be performed across directory hierarchies and mount-points as long as it is within the same store.

> **Note:**
>
> See *Oracle Database SecureFiles and Large Objects Developer's Guide* for Rename and Move operations

Syntax

```
DBMS_DBFS_CONTENT.RENAMEPATH (
   oldPath     IN              VARCHAR2,
   newPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.RENAMEPATH (
   oldPath     IN              VARCHAR2,
   newPath     IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   store_name  IN              VARCHAR2    DEFAULT NULL,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-63 RENAMEPATH Procedure Parameters

Parameter	Description
oldPath	Name of path prior to renaming
newPath	Name of path after renaming
properties	One or more properties and their values to be set depending on prop_flags (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
store_name	Name of store, must be unique
principal	Agent (principal) invoking the current operation

RESTOREALL Procedure

This procedure restores all soft-deleted path items meeting the path and optional filter criteria.

Syntax

```
DBMS_DBFS_CONTENT.RESTOREALL (
   path           IN      VARCHAR2,
   filter         IN      VARCHAR2    DEFAULT NULL,
   store_name     IN      VARCHAR2    DEFAULT NULL,
   principal      IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-64 RESTOREALL Procedure Parameters

Parameter	Description
path	Name of path to file items
filter	A filter, if any, to be applied
store_name	Name of store
principal	Agent (principal) invoking the current operation

RESTOREPATH Procedure

This procedure restores all soft-deleted path items that match the given path and optional filter criteria.

Syntax

```
DBMS_DBFS_CONTENT.RESTOREPATH (
   path           IN      VARCHAR2,
   filter         IN      VARCHAR2    DEFAULT NULL,
   store_name     IN      VARCHAR2    DEFAULT NULL,
   principal      IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-65 RESTOREPATH Procedure Parameters

Parameter	Description
`path`	Name of path to file items
`filter`	A filter, if any, to be applied
`store_name`	Name of store
`principal`	Agent (principal) invoking the current operation

SETDEFAULTACL Procedure

This procedure sets the ACL parameter of the default context.

This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTACL (
   acl    IN     VARCHAR2);
```

Parameters

Table 67-66 SETDEFAULTACL Procedure Parameters

Parameter	Description
`acl`	ACL for all new elements created (implicitly or explicitly) by the current operation

Usage Notes

- `NULL` by default, this parameter be can be cleared by setting it to `NULL`.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

SETDEFAULTASOF Procedure

This procedure sets the "as of" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTASOF (
  asof     IN     TIMESTAMP);
```

Parameters

Table 67-67 SETDEFAULTASOF Procedure Parameters

Parameter	Description
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

Usage Notes

- NULL by default, this parameter be can be cleared by setting it to NULL.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

SETDEFAULTCONTEXT Procedure

This procedure sets the default context. The information contained in the context can be inserted explicitly by way of arguments to the various method calls, allowing for fine-grained control over individual operations.

Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTCONTEXT (
   principal     IN     VARCHAR2,
   owner         IN     VARCHAR2,
   acl           IN     VARCHAR2,
   asof          IN     TIMESTAMP);
```

Parameters

Table 67-68 SETDEFAULTCONTEXT Procedure Parameters

Parameter	Description
principal	Agent (principal) invoking the current operation
owner	Owner for new elements created (implicitly or explicitly) by the current operation
acl	ACL for all new elements created (implicitly or explicitly) by the current operation
asof	The "as of" timestamp at which the underlying read-only operation (or its read-only sub-components) executes

Usage Notes

- All of the context parameters are `NULL` by default, and be can be cleared by setting them to `NULL`.
- The context parameters, once set, remain as defaults for the duration of the session, and are inherited by all operations for which the defaults are not explicitly overridden.

SETDEFAULTOWNER Procedure

This procedure sets the "owner" parameter of the default context. This information can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTOWNER (
   principal    IN     VARCHAR2);
```

Parameters

Table 67-69 SETDEFAULTOWNER Procedure Parameters

Parameter	Description
`owner`	Owner for new elements created (implicitly or explicitly) by the current operation

Usage Notes

- `NULL` by default, this parameter be can be cleared by setting it to `NULL`.
- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

SETDEFAULTPRINCIPAL Procedure

This procedure sets the "principal" parameter of the default context. This information contained can be inserted explicitly by way of argument into other method calls, allowing for a more fine-grained control.

Syntax

```
DBMS_DBFS_CONTENT.SETDEFAULTPRINCIPAL (
   principal    IN     VARCHAR2);
```

Parameters

Table 67-70 SETDEFAULTPRINCIPAL Procedure Parameters

Parameter	Description
`principal`	Agent (principal) invoking the current operation

Usage Notes

- `NULL` by default, this parameter be can be cleared by setting it to `NULL`.

- The parameters, once set, remain as a default for the duration of the session, and is inherited by all operations for which the default is not explicitly overridden.

SETPATH Procedures

This procedure assigns a path name to a path item represented by contentID.

Stores and their providers that support contentID-based access and lazy path name binding also support the `SETPATH Procedure` that associates an existing `contentID` with a new path.

> **Note:**
>
> See *Oracle Database SecureFiles and Large Objects Developer's Guide* for Rename and Move operations

Syntax

```
DBMS_DBFS_CONTENT.SETPATH (
   store_name  IN              VARCHAR2,
   contentID   IN              RAW,
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   principal   IN              VARCHAR2    DEFAULT NULL);

DBMS_DBFS_CONTENT.SETPATH (
   store_name  IN              VARCHAR2,
   contentID   IN              RAW,
   path        IN              VARCHAR2,
   properties  IN OUT NOCOPY   PROPERTIES_T,
   principal   IN              VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-71 SETPATH Procedure Parameters

Parameter	Description
`store_name`	Name of the store
`contentID`	Unique identifier for the item to be associated
`path`	Name of path to path item
`properties`	One or more properties and their values to be set depending on `prop_flags` (see DBMS_DBFS_CONTENT_PROPERTIES_T Table Type)
`principal`	Agent (principal) invoking the current operation

SETSTATS Procedure

This procedure enables and disables statistics collection.

The client can optionally control the flush settings by specifying non-`NULL` values for the time, count or both parameters.

Syntax

```
DBMS_DBFS_CONTENT.SETSTATS (
   enable          IN    BOOLEAN,
   flush_time      IN    INTEGER,
   flush_count     IN    INTEGER);
```

Parameters

Table 67-72 SETSTATS Procedure Parameters

Parameter	Description
`enable`	If `TRUE`, enable statistics collection. If `FALSE`, disable statistics collection.
`flush_time`	How often to flush the statistics to disk in centiseconds
`flush_count`	Number of operations to allow between statistics flushes

Usage Notes

The `SETSTATS Procedure` buffers statistics in-memory for a maximum of `flush_time` centiseconds or a maximum of `flush_count` operations (whichever limit is reached first), or both, at which time the buffers are implicitly flushed to disk.

SETTRACE Procedure

This procedure sets the `DBMS_DBFS_CONTENT` tracing severity to the given level, `0` being "off".

Syntax

```
DBMS_DBFS_CONTENT.SETTRACE
     trclvl      IN         INTEGER);
```

Parameters

Table 67-73 SETTRACE Procedure Parameters

Parameter	Description
`trclvl`	Level of the tracing, higher values implying more tracing

SPACEUSAGE Procedure

This procedure queries file system space usage statistics.

Providers are expected to support this subprogram for their stores (and to make a best effort determination of space usage, especially if the store consists of multiple tables, indexes, LOBs, and so on).

Syntax

```
DBMS_DBFS_CONTENT.SPACEUSAGE (
   path          IN        VARCHAR2,
   blksize       OUT       INTEGER,
   tbytes        OUT       INTEGER,
```

```
   fbytes        OUT       INTEGER,
   nfile         OUT       INTEGER,
   ndir          OUT       INTEGER,
   nlink         OUT       INTEGER,
   nref          OUT       INTEGER,
   store_name    IN        VARCHAR2 DEFAULT NULL);
```

Parameters

Table 67-74 SPACEUSAGE Procedure Parameters

Parameter	Description
path	Name of path to file items
blksize	Natural tablespace blocksize that holds the store. If multiple tablespaces with different blocksizes are used, any valid blocksize is acceptable.
tbytes	Total size of the store in bytes computed over all segments that comprise the store
fbytes	Free or unused size of the store in bytes computed over all segments that comprise the store
nfile	Number of currently available files in the store
ndir	Number of currently available directories in the store
nlink	Number of currently available links in the store
nref	Number of currently available references in the store
store_name	Name of store

Usage Notes

- A space usage query on the top-level root directory returns a combined summary of the space usage of all available distinct stores under it (if the same store is mounted multiple times, is still counted only once).
- Since database objects are dynamically expandable, it is not easy to estimate the division between "free" space and "used" space.

TRACE Procedure

This procedure outputs tracing to the current foreground trace file.

Syntax

```
DBMS_DBFS_CONTENT.TRACE
   sev           IN              INTEGER,
   msg0          IN              VARCHAR2,
   msg1          IN              VARCHAR     DEFAULT '',
   msg2          IN              VARCHAR     DEFAULT '',
   msg3          IN              VARCHAR     DEFAULT '',
   msg4          IN              VARCHAR     DEFAULT '',
   msg5          IN              VARCHAR     DEFAULT '',
   msg6          IN              VARCHAR     DEFAULT '',
   msg7          IN              VARCHAR     DEFAULT '',
   msg8          IN              VARCHAR     DEFAULT '',
   msg9          IN              VARCHAR     DEFAULT '',
   msg10         IN              VARCHAR     DEFAULT '');
```

Parameters

Table 67-75 TRACE Procedure Parameters

Parameter	Description
sev	Severity at which trace message is output
msg*	One or more message strings to be output. If more than one message is specified, all are output.

Usage Notes

- Trace information is written to the foreground trace file, with varying levels of detail as specified by the trace level arguments.
- The global trace level consists of 2 components: "severity" and "detail". These can be thought of as additive bitmasks.

  The "severity" allows the separation of top level as compared to low-level tracing of different components, and allows the amount of tracing to be increased as needed. There are no semantics associated with different levels, and users are free to set trace at any severity they choose, although a good rule of thumb would use severity "1" for top level API entry and exit traces, "2" for internal operations, and "3" or greater for very low-level traces.

  The "detail" controls how much additional information: timestamps, short-stack, etc. is dumped along with each trace record.

TRACEENABLED Function

This function determines if the current trace "severity" set by the SETTRACE Procedure is at least as high as the given trace level.

Syntax

```
DBMS_DBFS_CONTENT.TRACEENABLED(
   sev          IN               INTEGER)
  RETURN  INTEGER;
```

Parameters

Table 67-76 TRACEENABLED Procedure Parameters

Parameter	Description
sev	Severity at which trace message is output

Return Values

Returns 0 if the requested severity level is lower than the currently set trace severity level; 1 otherwise.

Related Topics

- SETTRACE Procedure
  This procedure sets the DBMS_DBFS_CONTENT tracing severity to the given level, 0 being "off".

UNLOCKPATH Procedure

This procedure unlocks path items that were previously locked with the LOCKPATH Procedure.

Syntax

```
DBMS_DBFS_CONTENT.UNLOCKPATH (
   path           IN     VARCHAR2,
   who            IN     VARCHAR2,
   waitForRowLock IN     INTEGER     DEFAULT 1,
   store_name     IN     VARCHAR2    DEFAULT NULL,
   principal      IN     VARCHAR2    DEFAULT NULL);
```

Parameters

Table 67-77 UNLOCKPATH Procedure Parameters

Parameter	Description
`path`	Name of path to file items
`who`	Transaction identifier that has locked the path
`waitForRowLock`	Determines if a row is locked by a transaction or not
`store_name`	Name of store
`principal`	Agent (principal) invoking the current operation

Related Topics

- LOCKPATH Procedure
  This procedure applies user-level locks to the given valid path name (subject to store feature support), and optionally associates user-data with the lock.

UNMOUNTSTORE Procedure

This procedure unmounts a registered store, either by name or by mount point.

Syntax

```
DBMS_DBFS_CONTENT.UNMOUNTSTORE (
   store_name       IN      VARCHAR2   DEFAULT NULL,
   store_mount      IN      VARCHAR2   DEFAULT NULL,
   ignore_unknown   IN      BOOLEAN    DEFAULT FALSE);
```

Parameters

Table 67-78 UNMOUNTSTORE Procedure Parameters

Parameter	Description
`store_name`	Name of store
`store_mount`	Location at which the store instance is mounted
`ignore_unknown`	If `TRUE`, attempts to unregister unknown stores will not raise an exception.

Usage Notes

> **See Also:**
>
> *Oracle Database SecureFiles and Large Objects Developer's Guide* for further information on unmounting a previously unmounted store

UNREGISTERSTORE Procedure

This procedure unregisters a previously registered store (invalidating all mount points associated with it).

Syntax

```
DBMS_DBFS_CONTENT.UNREGISTERSTORE (
   store_name          IN      VARCHAR2,
   ignore_unknown      IN      BOOLEAN DEFAULT FALSE);
```

Parameters

Table 67-79 UNREGISTERSTORE Procedure Parameters

Parameter	Description
`store_name`	Name of store
`ignore_unknown`	If `TRUE`, attempts to unregister unknown stores will not raise an exception.

Usage Notes

- Once unregistered all access to the store (and its mount points) are not guaranteed to work
- If the `ignore_unknown` argument is `TRUE`, attempts to unregister unknown stores do not raise an exception.