68
DBMS_DBFS_CONTENT_SPI

The `DBMS_DBFS_CONTENT_SPI` package is a specification for DBMS_DBFS_CONTENT store providers, which must be implemented. Application designers can create PL/SQL packages conforming to this specification to extend `DBMS_DBFS_CONTENT` to use custom store providers.

This chapter contains the following topics:

- Overview
- Security Model
- Operational Notes
- Summary of DBMS_DBFS_CONTENT_SPI Subprograms

Related Topics

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

See Also:

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

ODBMS_DBFS_CONTENT_SPI Overview

The `DBMS_DBFS_CONTENT_SPI` package describes an internal contract between the implementation of the DBMS_DBFS_CONTENT interface and individual store providers, and whichever package contains their code.

Since PL/SQL does not allow a compile-time, declarative type-conformation between package signatures, store providers should informally conform to the SPI, which is to say, they should implement the SPI by means of a package that contains all of the methods specified in package `DBMS_DBFS_CONTENT_SPI`, with the same method signatures and semantics.

Obviously, these provider packages can implement other methods and expose other interfaces, however, these interfaces are not to be used by the `DBMS_DBFS_CONTENT` interface itself.

Since the provider SPI is merely a contract specification, there is no package body for `DBMS_DBFS_CONTENT_SPI`, and it is not possible to actually invoke any methods using this package.

The SPI references various elements (constants, types, exceptions) defined by the `DBMS_DBFS_CONTENT` interface.

Additionally, there is an almost one-to-one correspondence between the client API exported by the DBMS_DBFS_CONTENT interface and the provider interface that the DBMS_DBFS_CONTENT interface itself expects to work against.

The main distinction in the method naming conventions is that all path name references are always store-qualified. That is, the notion of mount-points and full-absolute path names have been normalized and converted to store-qualified path names by the DBMS_DBFS_CONTENT interface before it invokes any of the provider SPI methods.

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

Related Topics

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

DBMS_DBFS_CONTENT_SPI Security Model

Implementations of the DBMS_DBFS_CONTENT_SPI package should be created as AUTHID CURRENT_USER.

DBMS_DBFS_CONTENT_SPI Operational Notes

This topic lists operational notes for DBMS_DBFS_CONTENT_SPI 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` (seeTable 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.

Other DBMS_DBFS_CONTENT Operations

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

Table 68-1 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

Summary of DBMS_DBFS_CONTENT_SPI Subprograms

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

Table 68-2 DBMS_DBFS_CONTENT_SPI Package Subprograms

Subprogram	Description
CHECKACCESS Function	Reports if the user (`principal`) can perform the specified operation on the given path
CREATEDIRECTORY Procedure	Creates a directory
CREATEFILE Procedure	Creates a file
CREATELINK Procedure	Creates a physical link to an already existing file system element
CREATEREFERENCE Procedure	Creates a new reference to the source file system element
DELETECONTENT Procedure	Deletes the file specified by the given contentID
DELETEDIRECTORY Procedure	Deletes a directory
DELETEFILE Procedure	Deletes a file
GETFEATURES Function	Returns the features of a store
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

Table 68-2 (Cont.) DBMS_DBFS_CONTENT_SPI Package Subprograms

Subprogram	Description
GETPATHNOWAIT 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.
GETSTOREID Function	Returns the ID of a store
GETVERSION Function	Returns the version associated with a store
LIST Function	Lists the contents of a directory path name
LOCKPATH Procedure	Applies user-level locks to the given valid path name
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
RENAMEPATH Procedure	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
SEARCH Function	Searches for path items matching the given path and filter criteria
SETPATH Procedure	Assigns a path name to a path item represented by contentID
SPACEUSAGE Procedure	Queries file system space usage statistics
UNLOCKPATH Procedure	Unlocks path items that were previously locked with the LOCKPATH Procedure

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

Parameters

Table 68-3 CHECKACCESS Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to check for access

Table 68-3 (Cont.) CHECKACCESS Procedure Parameters

Parameter	Description
`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

Usage Notes

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

CREATEDIRECTORY Procedure

This procedure creates a directory.

Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEDIRECTORY (
   store_name   IN              VARCHAR2,
   path         IN              VARCHAR2,
   properties   IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags   IN              INTEGER,
   recurse      IN              INTEGER,
   ctx          IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-4 CREATEDIRECTORY Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to the directory
`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
`ctx`	Context with which to create the directory (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

CREATEFILE Procedure

This procedure creates a file.

Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEFILE (
   store_name     IN              VARCHAR2,
   path           IN              VARCHAR2,
   properties     IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   content        IN OUT NOCOPY   BLOB,
   prop_flags     IN              INTEGER,
   ctx            IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-5 CREATEFILE Procedure Parameters

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

CREATELINK Procedure

This procedure creates a physical link to an already existing file system element (such as file or directory). The resulting entry shares the same metadata structures as the value of the `srcPath` 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_SPI.CREATELINK (
   store_name     IN              VARCHAR2,
   srcPath        IN              VARCHAR2,
   dstPath        IN              VARCHAR2,
   properties     IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags     IN              INTEGER,
   ctx            IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-6 CREATELINK Procedure Parameters

Parameter	Description
`store_name`	Name of store
`srcPath`	File system entry with which to link
`dstPath`	Path of the new link element to be created
`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_spec`, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.
`ctx`	Context with which to create the link (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

CREATEREFERENCE Procedure

This procedure creates a new reference to the source file system element (such as a file, or directory). The resulting reference points to the source element but does not directly share metadata with the source element. This is analogous to a UNIX file system symbolic link.

Syntax

```
DBMS_DBFS_CONTENT_SPI.CREATEREFERENCE (
   srcPath       IN               VARCHAR2,
   dstPath       IN               VARCHAR2,
   properties    IN OUT NOCOPY    DBMS_DBFS_CONTENT_PROPERTIES_T,
   prop_flags    IN               INTEGER,
   store_name    IN               VARCHAR2,
   ctx           IN               DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-7 CREATEREFERENCE Procedure Parameters

Parameter	Description
`store_name`	Name of store
`srcPath`	File system entry with which to link
`dstPath`	Path of the new link element to be created
`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_spec`, and providing an instance of the DBMS_DBFS_CONTENT_PROPERTIES_T Table Type with properties whose values are of interest.

Table 68-7 (Cont.) CREATEREFERENCE Procedure Parameters

Parameter	Description
`ctx`	Context with which to create the reference (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

DELETECONTENT Procedure

This procedure deletes the file specified by the given contentID.

Syntax

```
DBMS_DBFS_CONTENT_SPI.DELETECONTENT (
   store_name     IN     VARCHAR2,
   contentID      IN     RAW,
   filter         IN     VARCHAR2,
   soft_delete    IN     INTEGER,
   ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-8 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).
`ctx`	Context with which to delete the file (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

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_SPI.DELETEDIRECTORY (
   store_name     IN     VARCHAR2,
   path           IN     VARCHAR2,
   filter         IN     VARCHAR2,
   soft_delete    IN     INTEGER,
   recurse        IN     INTEGER,
   ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-9 DELETEDIRECTORY Procedure Parameters

Parameter	Description
`store_name`	Name of store
`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.
`ctx`	Context with which to delete the directory (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

DELETEFILE Procedure

This procedure deletes the specified file.

Syntax

```
DBMS_DBFS_CONTENT_SPI.DELETEFILE (
   store_name     IN     VARCHAR2,
   path           IN     VARCHAR2,
   filter         IN     VARCHAR2,
   soft_delete    IN     BOOLEAN,
   ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-10 DELETEFILE Procedure Parameters

Parameter	Description
`store_name`	Name of store
`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).
`ctx`	Context with which to delete the file (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

GETFEATURES Function

This function returns the features of a store.

Syntax

```
DBMS_DBFS_CONTENT_SPI.GETFEATURES (
   store_name          IN      VARCHAR2)
  RETURN  INTEGER;
```

Parameters

Table 68-11 GETFEATURES Function Parameters

Parameter	Description
`store_name`	Name of store

Return Values

`DBMS_DBFS_CONTENT.FEATURE_*` features supported by the Store Provider

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 "as of" 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_SPI.GETPATH (
   store_name  IN             VARCHAR2,
   path        IN             VARCHAR2,
   properties  IN OUT NOCOPY  DBMS_DBFS_CONTENT_PROPERTIES_T,
   content     OUT    NOCOPY  BLOB,
   item_type   OUT            INTEGER,
   prop_flags  IN             INTEGER,
   forUpdate   IN             INTEGER,
   deref       IN             INTEGER,
   ctx         IN             DBMS_DBFS_CONTENT_CONTEXT_T);

DBMS_DBFS_CONTENT_SPI.GETPATH (
   store_name  IN             VARCHAR2,
   path        IN             VARCHAR2,
   properties  IN OUT NOCOPY  DBMS_DBFS_CONTENT_PROPERTIES_T,
   amount      IN OUT         NUMBER,
   offset      IN             NUMBER,
   buffer      OUT    NOCOPY  RAW,
   prop_flags  IN             INTEGER,
   ctx         IN             DBMS_DBFS_CONTENT_CONTEXT_T);
```

```
DBMS_DBFS_CONTENT_SPI.GETPATH (
   store_name  IN              VARCHAR2,
   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,
   ctx         IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-12 GETPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`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.
`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 (symbolic link)
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

GETPATHBYSTOREID Function

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

Syntax

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

Parameters

Table 68-13 GETPATHBYSTOREID Function Parameters

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

Return Values

Store-qualified path name represented by the GUID

Usage Notes

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

GETPATHNOWAIT Procedure

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_SPI.GETPATHNOWAIT (
   store_name   IN             VARCHAR2,
   path         IN             VARCHAR2,
   properties   IN OUT NOCOPY  DBMS_DBFS_CONTENT_PROPERTIES_T,
   content      OUT    NOCOPY  BLOB,
   item_type    OUT            INTEGER,
   prop_flags   IN             INTEGER,
   deref        IN             INTEGER,
   ctx          IN             DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-14 GETPATHNOWAIT Procedure Parameters

Parameter	Description
`store_name`	Name of store
`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.

Table 68-14 (Cont.) GETPATHNOWAIT Procedure Parameters

Parameter	Description
`deref`	If nonzero, attempts to resolve the given path item to actual data provided it is a reference (symbolic link)
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

GETSTOREID Function

This function returns the ID of a store.

Syntax

```
DBMS_DBFS_CONTENT_SPI.GETSTOREID (
   store_name          IN      VARCHAR2)
  RETURN  NUMBER;
```

Parameters

Table 68-15 GETSTOREID Function Parameters

Parameter	Description
`store_name`	Name of store

Return Values

ID of the Store

Usage Notes

A store ID identifies a provider-specific store, across registrations and mounts, but independent of changes to the store contents. For this reason, changes to the store table or tables should be reflected in the store ID, but re-initialization of the same store table or tables should preserve the store ID.

GETVERSION Function

This function returns the version associated with a store.

Syntax

```
DBMS_DBFS_CONTENT_SPI.GETVERSION (
   store_name          IN      VARCHAR2)
  RETURN  VARCHAR2;
```

Parameters

Table 68-16 GETVERSION Function Parameters

Parameter	Description
`store_name`	Name of store

Return Values

A "version" (either specific to a provider package, or to an individual store) based on a standard *a.b.c* naming convention (for *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 (
   store_name     IN     VARCHAR2,
   path           IN     VARCHAR2,
   who            IN     VARCHAR2,
   lock_type      IN OUT INTEGER,
   ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-17 ISPATHLOCKED Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Path name of items to be locked
`who`	Transaction identifier that has locked the path
`lock_type`	One of the available lock types (see Table 67-6)
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

LIST Function

This function lists the contents of a directory path name.

The invoker of the subprogram has the option to investigate recursively into sub-directories, to make soft-deleted items visible, to use a flashback "as of" a specified timestamp, and to filter items within the store based on list predicates.

Syntax

```
DBMS_DBFS_CONTENT_SPI.LIST (
   store_name    IN     VARCHAR2,
   path          IN     VARCHAR2,
   filter        IN     VARCHAR2,
   recurse       IN     INTEGER,
   ctx           IN     DBMS_DBFS_CONTENT_CONTEXT_T)
  RETURN  DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

Parameters

Table 68-18 LIST Function Parameters

Parameter	Description
`store_name`	Name of repository
`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.
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

Return Values

Path items found that match the path, filter and criteria for executing recursively (see DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type)

Usage Notes

This function returns only list items; the client is expected to explicitly use one of the GETPATH Procedures to access the properties or content associated with an item.

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_SPI.LOCKPATH (
   store_name      IN     VARCHAR2,
   path            IN     VARCHAR2,
   who             IN     VARCHAR2,
   lock_type       IN     INTEGER,
   waitForRowLock  IN     INTEGER,
   ctx             IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-19 LOCKPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Path name of items to be locked
`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
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

Usage Notes

- It is the responsibility of the store and its providers (assuming it supports user-defined lock checking) to ensure that lock and unlock operations are performed in a consistent manner.
- The status of locked items is available by means of various optional properties (see `OPT_LOCK`* in Table 67-8).

PURGEALL Procedure

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

Syntax

```
DBMS_DBFS_CONTENT_SPI.PURGEALL (
   store_name      IN      VARCHAR2,
   path            IN      VARCHAR2,
   filter          IN      VARCHAR2,
   ctx             IN      DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-20 PURGEALL Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to file items
`filter`	A filter, if any, to be applied based on specified criteria
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

PURGEPATH Procedure

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

Syntax

```
DBMS_DBFS_CONTENT_SPI.PURGEPATH (
   path            IN      VARCHAR2,
   filter          IN      VARCHAR2,
   store_name      IN      VARCHAR2,
   ctx             IN      DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-21 PURGEPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to file items
`filter`	A filter, if any, to be applied

Table 68-21 (Cont.) PURGEPATH Procedure Parameters

Parameter	Description
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

PUTPATH Procedures

This procedure creates a new path item.

Syntax

```
DBMS_DBFS_CONTENT_SPI.PUTPATH (
   store_name     IN               VARCHAR2,
   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,
   ctx            IN               DBMS_DBFS_CONTENT_CONTEXT_T);

DBMS_DBFS_CONTENT_SPI.PUTPATH (
   store_name     IN               VARCHAR2,
   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,
   ctx            IN               DBMS_DBFS_CONTENT_CONTEXT_T);

DBMS_DBFS_CONTENT_SPI.PUTPATH (
   store_name     IN               VARCHAR2,
   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,
   ctx            IN               DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-22 PUTPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Path name of item to be put
`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 68-22 (Cont.) PUTPATH Procedure Parameters

Parameter	Description
`written`	Number of bytes written
`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.
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

Usage Notes

- All path names allow their metadata (properties) to be read and modified. On completion of the call, the client can access specific properties using `prop_flags` (see Table 67-9).
- On completion of the call, the client can request a new `BLOB` locator that can be used to continue data access using the `prop_data` bitmask in `prop_flags` (see Table 67-9).
- Files can also be written without using `BLOB` locators, by explicitly specifying logical offsets or buffer-amounts, and a suitably sized buffer.

RENAMEPATH Procedure

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 further information on Rename and Move operations

Syntax

```
DBMS_DBFS_CONTENT_SPI.RENAMEPATH (
   store_name     IN              VARCHAR2,
   oldPath        IN              VARCHAR2,
   newPath        IN              VARCHAR2,
   properties     IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   ctx            IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-23 RENAMEPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store, must be unique

Table 68-23 (Cont.) 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)
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

RESTOREALL Procedure

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

Syntax

```
DBMS_DBFS_CONTENT_SPI.RESTOREALL (
   store_name      IN      VARCHAR2,
   path            IN      VARCHAR2,
   filter          IN      VARCHAR2,
   ctx             IN      DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-24 RESTOREALL Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to path items
`filter`	A filter, if any, to be applied
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

RESTOREPATH Procedure

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

Syntax

```
DBMS_DBFS_CONTENT_SPI.RESTOREPATH (
   store_name      IN      VARCHAR2,
   path            IN      VARCHAR2,
   filter          IN      VARCHAR2,
   ctx             IN      DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-25 RESTOREPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to path items
`filter`	A filter, if any, to be applied
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

SEARCH Function

This function searches for path items matching the given path and filter criteria.

Syntax

```
DBMS_DBFS_CONTENT_SPI.SEARCH (
   store_name      IN      VARCHAR2,
   path            IN      VARCHAR2,
   filter          IN      VARCHAR2,
   recurse         IN      INTEGER,
   ctx             IN      DBMS_DBFS_CONTENT_CONTEXT_T)
 RETURN  DBMS_DBFS_CONTENT_LIST_ITEMS_T PIPELINED;
```

Parameters

Table 68-26 LIST Function Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to the path items
`filter`	A filter, if any, to be applied
`recurse`	If 0, do not execute recursively. Otherwise, recursively search the contents of directories and files below the given directory.
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

Return Values

Path items matching the given path and filter criteria (see DBMS_DBFS_CONTENT_LIST_ITEMS_T Table Type)

SETPATH Procedure

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 further information on Rename and Move operations

Syntax

```
DBMS_DBFS_CONTENT_SPI.SETPATH (
   store_name     IN              VARCHAR2,
   contentID      IN              RAW,
   path           IN              VARCHAR2,
   properties     IN OUT NOCOPY   DBMS_DBFS_CONTENT_PROPERTIES_T,
   ctx            IN              DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-27 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)
`ctx`	Context with which to access the path items (seE DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

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_SPI.SPACEUSAGE (
   store_name     IN        VARCHAR2,
   blksize        OUT       INTEGER,
   tbytes         OUT       INTEGER,
   fbytes         OUT       INTEGER,
   nfile          OUT       INTEGER,
   ndir           OUT       INTEGER,
   nlink          OUT       INTEGER,
   nref           OUT       INTEGER);
```

Parameters

Table 68-28 SPACEUSAGE Procedure Parameters

Parameter	Description
`store_name`	Name of store
`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

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, it 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.

UNLOCKPATH Procedure

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

Syntax

```
DBMS_DBFS_CONTENT_SPI.UNLOCKPATH (
   store_name     IN     VARCHAR2,
   path           IN     VARCHAR2,
   who            IN     VARCHAR2,
   waitForRowLock IN     INTEGER,
   ctx            IN     DBMS_DBFS_CONTENT_CONTEXT_T);
```

Parameters

Table 68-29 UNLOCKPATH Procedure Parameters

Parameter	Description
`store_name`	Name of store
`path`	Name of path to the path items
`who`	Transaction identifier that has locked the path
`waitForRowLock`	Determines if a row is locked by a transaction or not
`ctx`	Context with which to access the path items (see DBMS_DBFS_CONTENT_CONTEXT_T Object Type)

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.