22

DBFS Content API

You can enable applications to use the Database File System (DBFS) in several different programming environments.

- Overview of DBFS Content API
  You can enable applications to use DBFS using the DBFS Content API (`DBMS_DBFS_CONTENT`), which is a client-side programmatic API package. You can write applications in SQL, PL/SQL, JDBC, OCI, and other programming environments.
- Stores and DBFS Content API
  The DBFS Content API aggregates the path namespace of one or more stores into a single unified namespace.
- Getting Started with DBMS_DBFS_CONTENT Package
  `DBMS_DBFS_CONTENT` is part of the Oracle Database, starting with Oracle Database 11g Release 2, and does not need to be installed.
- Administrative and Query APIs
  Administrative clients and content providers are expected to register content stores with the DBFS Content API. Additionally, administrative clients are expected to mount stores into the top-level namespace of their choice.
- Querying DBFS Content API Space Usage
  You can query file system space usage statistics.
- DBFS Content API Session Defaults
  Normal client access to the DBFS Content API executes with an implicit context that consists of certain objects.
- DBFS Content API Interface Versioning
  To allow for the DBFS Content API itself to evolve, an internal numeric API version increases with each change to the public API.
- DBFS Content API Creation Operations
  You must implement the provider SPI so that when clients invoke the DBFS Content API, it causes the SPI to create directory, file, link, and reference elements (subject to store feature support).
- DBFS Content API Deletion Operations
  You must implement the provider SPI so that when clients invoke the DBFS Content API, it causes the SPI to delete directory, file, link, and reference elements (subject to store feature support).
- DBFS Content API Path Get and Put Operations
  You can query existing path items or update them using simple `GET`*`XXX`*`()` and `PUT`*`XXX`*`()` methods.
- DBFS Content API Rename and Move Operations
  You can rename or move path names, possibly across directory hierarchies and mount points, but only within the same store.
- Directory Listings
  Directory listings are handled several different ways.

- DBFS Content API Directory Navigation and Search
  Clients of the DBFS Content API can list or search the contents of directory path names, with optional modes.
- DBFS Content API Locking Operations
  DBFS Content API clients can apply user-level locks,depending on certain criteria.
- DBFS Content API Access Checks
  The DBFS Content API checks the access of specific path names by operations.
- DBFS Content API Abstract Operations
  All of the operations in the DBFS Content API are represented as abstract `opcodes`.
- DBFS Content API Path Normalization
  There is a process for performing API path normalization.
- DBFS Content API Statistics Support
  DBFS provides support to reduce the expense of collecting DBFS Content API statistics.
- DBFS Content API Tracing Support
  Any DBFS Content API user (both clients and providers) can use DBFS Content API tracing, a generic tracing facility.
- Resource and Property Views
  You can see descriptions of Content API structure and properties in certain views.

22.1 Overview of DBFS Content API

You can enable applications to use DBFS using the DBFS Content API (`DBMS_DBFS_CONTENT`), which is a client-side programmatic API package. You can write applications in SQL, PL/SQL, JDBC, OCI, and other programming environments.

The DBFS Content API is a collection of methods that provide a file system-like abstraction. It is backed by one or more DBFS Store Providers. The *Content* in the DBFS Content interface refers to a file, including metadata, and it can either map to a SecureFiles `LOB` (and other columns) in a table or be dynamically created by user-written plug-ins in Java or PL/SQL that run inside the database. The plug-in form is referred to as a *provider*.

Note:

The DBFS Content API includes the SecureFiles Store Provider, `DBMS_DBFS_SFS`, a default implementation that enables applications that already use LOBs as columns in their schema, to access the `LOB` columns as files.

See Also:

DBFS SecureFiles Store

Examples of possible providers include:

- Packaged applications that want to expose data through files.
- Custom applications developers use to leverage the file system interface, such as an application that stores medical images.

22.2 Stores and DBFS Content API

The DBFS Content API aggregates the path namespace of one or more stores into a single unified namespace.

The first component of the path name is used to disambiguate the namespace and then present it to client applications. This allows clients to access the underlying documents using either a full absolute path name represented by a single string, as shown in the following code snippet:

```
/store-name/store-specific-path-name
```

The DBFS Content API then takes care of correctly dispatching various operations on path names to the appropriate store provider .

Store providers must conform to the store provider interface (SPI) as declared by the package `DBMS_DBFS_CONTENT_SPI`.

- Creating Your Own DBFS Store
- *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_DBFS_CONTENT` package syntax reference

22.3 Getting Started with DBMS_DBFS_CONTENT Package

`DBMS_DBFS_CONTENT` is part of the Oracle Database, starting with Oracle Database 11g Release 2, and does not need to be installed.

- DBFS Content API Role
  Access to the content operational and administrative API (packages, types, tables, and so on) is available through `DBFS_ROLE`.
- Path Name Constants and Types
  Path name constants are modeled after their SecureFiles LOBs store counterparts.
- Path Properties
  Every path name in a store is associated with a set of properties.
- Content IDs
  Content IDs are unique identifiers that represent a path in the store.
- Path Name Types
  Stores can store and provide access to eight types of entities.
- Store Features
  In order to provide a common programmatic interface to as many different types of stores as possible, the DBFS Content API leaves some of the behavior of various operations to individual store providers to define and implement.
- Lock Types
  Stores that support locking should implement three types of locks.
- Standard Properties
  Standard properties are well-defined, mandatory properties associated with all content path names, which all stores must support, in the manner described by the DBFS Content API.

- Optional Properties
  Optional properties are well-defined but non-mandatory properties associated with all content path names that all stores are free to support (but only in the manner described by the DBFS Content API).
- User-Defined Properties
  You can define your own properties for use in your application.
- Property Access Flags
  DBFS Content API methods to get and set properties can use combinations of property access flags to fetch properties from different namespaces in a single API call.
- Exceptions
  DBFS Content API operations can raise any one of the top-level exceptions.
- Property Bundles
  Property bundles are discussed as `property_t` record type and `properties_t`.
- Store Descriptors
  Store descriptors are discussed as `store_t` and `mount_t` records.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information

22.3.1 DBFS Content API Role

Access to the content operational and administrative API (packages, types, tables, and so on) is available through `DBFS_ROLE`.

The `DBFS_ROLE` can be granted to all users as needed.

22.3.2 Path Name Constants and Types

Path name constants are modeled after their SecureFiles LOBs store counterparts.

See Also:

DBMS_DBFS_CONTENT Constants for path name constants and their types

22.3.3 Path Properties

Every path name in a store is associated with a set of properties.

For simplicity and generality, each property is identified by a string name, has a string value (possibly `null` if not set 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.

Coercing property values to strings has the advantage of making the various interfaces uniform and compact (and can even simplify implementation of the underlying stores), but has the potential for information loss during conversions to and from strings.

It is expected that clients and stores use well-defined database conventions for these conversions and use the `typecode` field as appropriate.

PL/SQL types `path_t` and `name_t` are portable aliases for strings that can represent pathnames and component names,

A typecode is a numeric value representing the true type of a string-coerced property value. Simple scalar types (numbers, dates, timestamps, etc.) can be depended on by clients and must be implemented by stores.

Since standard RDBMS typecodes are positive integers, the `DBMS_DBFS_CONTENT` interface allows negative integers to represent client-defined types by negative typecodes. These typecodes do not conflict with standard typecodes, are maintained persistently and returned to the client as needed, but need not be interpreted by the DBFS content API or any particular store. Portable client applications should not use user-defined typecodes as a back door way of passing information to specific stores.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` constants and properties and the `DBMS_DBFS_CONTENT_PROPERTY_T` package

22.3.4 Content IDs

Content IDs are unique identifiers that represent a path in the store.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` Content ID constants and properties

22.3.5 Path Name Types

Stores can store and provide access to eight types of entities.

The entities are:

- `type_file`
- `type_directory`
- `type_link`
- `type_reference`
- `type_scoket`
- `type_character`
- `type_block`
- `type_fifo`

Not all stores must implement all directories, links, or references.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` constants and path name types

22.3.6 Store Features

In order to provide a common programmatic interface to as many different types of stores as possible, the DBFS Content API leaves some of the behavior of various operations to individual store providers to define and implement.

The DBFS Content API remains rich and conducive to portable applications by allowing different store providers (and different stores) to describe themselves as a feature set. A feature set is a bit mask indicating the supported features and the ones that are not supported.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the store features and constants

22.3.7 Lock Types

Stores that support locking should implement three types of locks.

The three types of locks are: `lock_read_only`, `lock_write_only`, `lock_read_write`.

User locks (any of these types) can be associated with user-supplied `lock_data`. The store does not interpret the data, but client applications can use it for their own purposes (for example, the user data could indicate the time at which the lock was placed, and the client application might use this later to control its actions.

In the simplest locking model, a `lock_read_only` prevents all explicit modifications to a path name (but allows implicit modifications and changes to parent/child path names). A `lock_write_only` prevents all explicit reads to the path name, but allows implicit reads and reads to parent/child path names. A `lock_read_write` allows both.

All locks are associated with a principal user who performs the locking operation; stores that support locking are expected to preserve this information and use it to perform read/write lock checking (see `opt_locker`).

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the lock types and constants.

22.3.8 Standard Properties

Standard properties are well-defined, mandatory properties associated with all content path names, which all stores must support, in the manner described by the DBFS Content API.

Stores created against tables with a fixed schema may choose reasonable defaults for as many of these properties as needed, and so on.

All standard properties informally use the `std` namespace. Clients and stores should avoid using this namespace to define their own properties to prevent conflicts in the future.

See Also:

See *Oracle Database PL/SQL Packages and Types Reference* for details of the standard properties and constants

22.3.9 Optional Properties

Optional properties are well-defined but non-mandatory properties associated with all content path names that all stores are free to support (but only in the manner described by the DBFS Content API).

Clients should be prepared to deal with stores that support none of the optional properties.

All optional properties informally use the `opt` namespace. Clients and stores must avoid using this namespace to define their own properties to prevent conflicts in the future.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the optional properties and constants

22.3.10 User-Defined Properties

You can define your own properties for use in your application.

Ensure that the namespace prefixes do not conflict with each other or with the DBFS standard or optional properties.

22.3.11 Property Access Flags

DBFS Content API methods to get and set properties can use combinations of property access flags to fetch properties from different namespaces in a single API call.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the property access flags and constants

22.3.12 Exceptions

DBFS Content API operations can raise any one of the top-level exceptions.

Clients can program against these specific exceptions in their error handlers without worrying about the specific store implementations of the underlying error signalling code.

Store service providers, should try to trap and wrap any internal exceptions into one of the exception types, as appropriate.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the Exceptions

22.3.13 Property Bundles

Property bundles are discussed as `property_t` record type and `properties_t`.

- The `property_t` record type describes a single (value, typecode) property value tuple; the property name is implied.
- `properties_t` 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`.

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.

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.

DBFS Content API provides convenient utility functions to convert between `dbms_dbfs_content_properties_t` and `properties_t`.

The function `DBMS_DBFS_CONTENT.PROPERTIEST2H` converts a `DBMS_DBFS_CONTENT_PROPERTIES_T` value to an equivalent `properties_t` value, and the function `DBMS_DBFS_CONTENT.PROPERTIESH2T` converts a `properties_t` value to an equivalent `DBMS_DBFS_CONTENT_PROPERTIES_T` value.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `PROPERTY_T` record type

22.3.14 Store Descriptors

Store descriptors are discussed as `store_t` and `mount_t` records.

- A `store_t` is a record that describes a store registered with, and managed by the DBFS Content API .
- A `mount_t` is a record that describes a store mount point and its properties.

Clients can query the DBFS Content API for the list of available stores, determine which store handles accesses to a given path name, and determine the feature set for the store.

See Also:

- Administrative and Query APIs
- *Oracle Database PL/SQL Packages and Types Reference* for details of the `STORE_T` record type

22.4 Administrative and Query APIs

Administrative clients and content providers are expected to register content stores with the DBFS Content API. Additionally, administrative clients are expected to mount stores into the top-level namespace of their choice.

The registration and unregistration of a store is separated from the mount and unmount of a store because it is possible for the same store to be mounted multiple times at different mount points (and this is under client control).

- Registering a Content Store
  You can register a new store that is backed by a provider that uses the `provider_package` procedure as the store service provider.
- Unregistering a Content Store
  You can unregister a previously registered store, which invalidates all mount points associated with it.
- Mounting a Registered Store
  You can mount a registered store and bind it to the mount point.
- Unmounting a Previously Mounted Store
  You can unmount a previously mounted store, either by name or by mount point.
- Listing all Available Stores and Their Features
  You can list all the available stores.
- Listing all Available Mount Points
  You can list all available mount points, their backing stores, and the store features.

- Looking Up Specific Stores and Their Features
  You can look up the path name, store name, or mount point of a store.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for the summary of `DBMS_DBFS_CONTENT` package methods

22.4.1 Registering a Content Store

You can register a new store that is backed by a provider that uses the `provider_package` procedure as the store service provider.

The method of registration conforms to the `DBMS_DBFS_CONTENT_SPI` package signature.

- Use the `REGISTERSTORE()` procedure.

This method is designed for use by service providers after they have created a new store. Store names must be unique.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for details of the `REGISTERSTORE()` method

22.4.2 Unregistering a Content Store

You can unregister a previously registered store, which invalidates all mount points associated with it.

Once the store is unregistered, access to the store and its mount points is no longer guaranteed, although a consistent read may provide a temporary illusion of continued access.

- Use the `UNREGISTERSTORE()` procedure.

If the `ignore_unknown` argument is `true`, attempts to unregister unknown stores do not raise an exception.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for details of the `UNREGISTERSTORE()` method

22.4.3 Mounting a Registered Store

You can mount a registered store and bind it to the mount point.

- Use the `MOUNTSTORE()`procedure.

After you mount the store, access to the path names in the form /`store_mount`/`xyz` is redirected to `store_name` and its content provider.

Store mount points must be unique, and a syntactically valid path name component (that is, a `name_t` with no embedded `/`).

If you do not specify a mount point and therefore, it is `null`, the DBFS Content API attempts to use the store name itself as the mount point name (subject to the uniqueness and syntactic constraints).

The same store can be mounted multiple times, obviously at different mount points.

You can use mount properties to specify the DBFS Content API execution environment, that is, the default values of the principal, owner, `ACL`, and `asof`, for a particular mount point. You can also use mount properties to specify a read-only store.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `MOUNTSTORE()` method

22.4.4 Unmounting a Previously Mounted Store

You can unmount a previously mounted store, either by name or by mount point.

Attempting to unmount a store by name unmounts all mount points associated with the store.

- Use the `UNMOUNTSTORE()` procedure.

Once unmounted, access to the store or mount-point is no longer guaranteed to work although a consistent read may provide a temporary illusion of continued access. If the `ignore_unknown` argument is `true`, attempts to unmount unknown stores does not raise an exception.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `UNMOUNTSTORE` method

22.4.5 Listing all Available Stores and Their Features

You can list all the available stores.

The `store_mount` field of the returned records is set to `null` because mount points are separate from stores themselves.

- Use the `LISTSTORES()` function.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `LISTSTORES` Function

22.4.6 Listing all Available Mount Points

You can list all available mount points, their backing stores, and the store features.

A single mount returns a single row, with the `store_mount` field set to `null`.

- Use the `LISTMOUNTS()` function.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `LISTMOUNTS()` method

22.4.7 Looking Up Specific Stores and Their Features

You can look up the path name, store name, or mount point of a store.

- Use `GETSTOREBY`*`XXX`*`()` or `GETFEATUREBY`*`XXX`*`()` functions.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.5 Querying DBFS Content API Space Usage

You can query file system space usage statistics.

Providers are expected to support this method 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.

- Use the `SPACEUSAGE()` method

where:

- `blksize` is the natural tablespace block size that holds the store; if multiple tablespaces with different block sizes are used, any valid block size is acceptable.
- `tbytes` is the total size of the store in bytes, and `fbytes` is the free or unused size of the store in bytes. These values are computed over all segments that comprise the store.

- `nfile`, `ndir`, `nlink`, and `nref` count the number of currently available files, directories, links, and references in the store.

Database objects can grow dynamically, so it is not easy to estimate the division between free space and used space.

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 counted only once.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `SPACEUSAGE()` method

22.6 DBFS Content API Session Defaults

Normal client access to the DBFS Content API executes with an implicit context that consists of certain objects.

- The `principal` invoking the current operation.
- The `owner` for all new elements created (implicitly or explicitly) by the current operation.
- The `ACL` for all new elements created (implicitly or explicitly) by the current operation.
- The `ASOF` timestamp at which the underlying read-only operation (or its read-only sub-components) execute.

All of this information can be passed in explicitly through arguments to the various DBFS Content API method calls, allowing the client fine-grained control over individual operations.

The DBFS Content API also allows clients to set session duration defaults for the context that are automatically inherited by all operations for which the defaults are not explicitly overridden.

All of the context defaults start out as `null` and can be cleared by setting them to `null`.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.7 DBFS Content API Interface Versioning

To allow for the DBFS Content API itself to evolve, an internal numeric API version increases with each change to the public API.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `GETVERSION()` method

22.8 DBFS Content API Creation Operations

You must implement the provider SPI so that when clients invoke the DBFS Content API, it causes the SPI to create directory, file, link, and reference elements (subject to store feature support).

All of the creation methods require a valid path name and can optionally specify properties to be associated with the path name as it is created. It is also possible for clients to fetch back item properties after the creation completes, so that automatically generated properties, such as `std_creation_time`, are immediately available to clients. The exact set of properties fetched back is controlled by the various `prop_`*`xxx`* bit masks in `prop_flags`.

Links and references require an additional path name associated with the primary path name. File path names can optionally specify a `BLOB` value to initially populate the underlying file content, and the provided `BLOB` may be any valid LOB, either temporary or permanent. On creation, the underlying `LOB` is returned to the client if `prop_data` is specified in `prop_flags`.

Non-directory path names require that their parent directory be created first. Directory path names themselves can be recursively created. This means that the path name hierarchy leading up to a directory can be created in one call.

Attempts to create paths that already exist produce an error, except for path names that are soft-deleted. In these cases, the soft-deleted item is implicitly purged, and the new item creation is attempted.

Stores and their providers that support contentID-based access accept an explicit store name and a `NULL` path to create a new content element. The contentID generated for this element is available by means of the `OPT_CONTENT_ID` property. The `PROP_OPT` property in the `prop_flags` parameter automatically implies contentID-based creation.

The newly created element may also have an internally generated path name if the `FEATURE_LAZY_PATH` property is not supported and this path is available by way of the `STD_CANONICAL_PATH` property.

Only file elements are candidates for contentID-based access.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for details of the `DBMS_DBFS_CONTENT()` methods, `DBMS_DBFS_CONTENT()` Constants - Optional Properties, and `DBMS_DBFS_CONTENT` Constants - Standard Properties

22.9 DBFS Content API Deletion Operations

You must implement the provider SPI so that when clients invoke the DBFS Content API, it causes the SPI to delete directory, file, link, and reference elements (subject to store feature support).

By default, the deletions are permanent, and remove successfully deleted items on transaction commit. However, repositories may also support soft-delete features. If requested by the client, soft-deleted items are retained by the store. They are not, however, typically visible in normal listings or searches. Soft-deleted items may be restored or explicitly purged.

Directory path names may be recursively deleted; the path name hierarchy below a directory may be deleted in one call. Non-recursive deletions can be performed only on empty directories. Recursive soft-deletions apply the soft-delete to all of the items being deleted.

Individual path names or all soft-deleted path names under a directory may be restored or purged using the `RESTORE`*`XXX`*`()` and `PURGE`*`XXX`*`()` methods.

Providers that support filtering can use the provider filter to identify subsets of items to delete; this makes most sense for bulk operations such as `deleteDirectory()`, `RESTOREALL()`, and `PURGEALL()`, but all of the deletion-related operations accept a filter argument.

Stores and their providers that support contentID-based access can also allow deleting file items by specifying their contentID.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT()` methods

22.10 DBFS Content API Path Get and Put Operations

You can query existing path items or update them using simple `GET`*`XXX`*`()` and `PUT`*`XXX`*`()` methods.

All path names allow their metadata to be read and modified. On completion of the call, the client can request that specific properties be fetched through `prop_flags`.

File path names allow their data to be read and modified. On completion of the call, the client can request a new `BLOB` locator through the `prop_data` bit masks in `prop_flags`; these may be used to continue data access.

Files can also be read and written without using `BLOB` locators, by explicitly specifying logical offsets, buffer amounts, and a suitably sized buffer.

Update accesses must specify the `forUpdate` flag. Access to link path names may be implicitly and internally dereferenced by stores, subject to feature support, if the `deref` flag is specified. Oracle does not recommend this practice because symbolic links are not guaranteed to resolve.

The read method `GETPATH()` where `forUpdate` is `false` accepts a valid `asof` timestamp parameter that can be used by stores to implement flashback-style queries.

Mutating versions of the `GETPATH()` and the `PUTPATH()` methods do not support `asof` modes of operation.

The DBFS Content API does not have an explicit `COPY()` operation because a copy is easily implemented as a combination of a `GETPATH()` followed by a `CREATE`*`XXX`*`()` with appropriate data or metadata transfer across the calls. This allows copies across stores, while an internalized copy operation cannot provide this facility.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.11 DBFS Content API Rename and Move Operations

You can rename or move path names, possibly across directory hierarchies and mount points, but only within the same store.

Non-directory path names previously accessible by `oldPath` can be renamed as a single item subsequently accessible by `newPath`, assuming that `newPath` does not exist.

If `newPath` exists and is not a directory, the rename implicitly deletes the existing item before renaming `oldPath`. If `newPath` exists and is a directory, `oldPath` is moved into the target directory.

Directory path names previously accessible by `oldPath` can be renamed by moving the directory and all of its children to `newPath` (if it does not exist) or as children of `newPath` (if it exists and is a directory).

Because the semantics of rename and move is complex with respect to non-existent or existent and non-directory or directory targets, clients may choose to implement complex rename and move operations as sequences of simpler moves or copies.

Stores and their providers that support contentID-based access and lazy path name binding also support the *Oracle Database PL/SQL Packages and Types Reference* `SETPATH` procedure that associates an existing contentID with a new "path".

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT.RENAMEPATH()` methods

22.12 Directory Listings

Directory listings are handled several different ways.

- A `list_item_t` is a tuple of path name, component name, and type representing a single element in a directory listing.

- A `path_item_t` is a tuple describing a store, mount qualified path in a content store, with all standard and optional properties associated with it.
- A `prop_item_t` is a tuple describing a store, mount qualified path in a content store, with all user-defined properties associated with it, expanded out into individual tuples of name, value, and type.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of data structures

22.13 DBFS Content API Directory Navigation and Search

Clients of the DBFS Content API can list or search the contents of directory path names, with optional modes.

Optional Modes:

- searching recursively in sub-directories
- seeing soft-deleted items
- using flashback `asof` a provided timestamp
- filtering items in and out within the store based on list or search predicates.

The DBFS Content API currently only returns list items; clients explicitly use one of the `getPath()` methods to access the properties or content associated with an item, as appropriate.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.14 DBFS Content API Locking Operations

DBFS Content API clients can apply user-level locks,depending on certain criteria.

Clients of the DBFS Content API can apply user-level locks to any valid path name, subject to store feature support, associate the lock with user data, and subsequently unlock these path names. The status of locked items is available through various optional properties.

If a store supports user-defined lock checking, it is responsible for ensuring that lock and unlock operations are performed in a consistent manner.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.15 DBFS Content API Access Checks

The DBFS Content API checks the access of specific path names by operations.

Function `CHECKACCESS()` checks if a given path name (`path`, `pathtype`, `store_name`) can be manipulated by an operation, such as the various `op_xxx` opcodes) by `principal`, as described in "DBFS Content API Locking Operations"

This is a convenience function for the client; a store that supports access control still internally performs these checks to guarantee security.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.16 DBFS Content API Abstract Operations

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

Clients can use`opcodes` to directly and explicitly invoke the `CHECKACCESS()` method which verifies if a particular operation can be invoked by a given principal on a particular path name.

An `op_acl()` is an implicit operation invoked during an `op_create()` or `op_put()` call, which 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_delete()` represents the soft-deletion, purge, and restore operations.

The source and destination operations of a rename or move operation are separated, although stores are free to unify these `opcode`s and to also treat a rename as a combination of delete and create.

`op_store` is a catch-all category for miscellaneous store operations that do not fall under any of the other operational APIs.

See Also:

- DBFS Content API Access Checks
- *Oracle Database PL/SQL Packages and Types Reference* for more information about `DBMS_DBFS_CONTENT` Constants - Operation Codes.

22.17 DBFS Content API Path Normalization

There is a process for performing API path normalization.

Function `NORMALIZEPATH()` performs the following steps:

1. Verifies that the path name is absolute (starts with a `/`).
2. Collapses multiple consecutive `/`s into a single `/`.
3. Strips trailing `/`s.
4. Breaks store-specific normalized path names into two components: the parent path name and the trailing component name.
5. Breaks fully qualified normalized path names into three components: store name, parent path name, and trailing component name.

Note that the root path `/` is special: its parent path name is also `/`, and its component name is `null`. In fully qualified mode, it has a `null` store name unless a singleton mount has been created, in which case the appropriate store name is returned.

The return value is always the completely normalized store-specific or fully qualified path name.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT.RENAMEPATH()` methods

22.18 DBFS Content API Statistics Support

DBFS provides support to reduce the expense of collecting DBFS Content API statistics.

DBFS Content API statistics are expensive to collect and maintain persistently. DBFS has support for buffering statistics in memory for a maximum of `flush_time` centiseconds or a maximum of `flush_count` operations, whichever limit is reached first), at which time the buffers are implicitly flushed to disk.

Clients can also explicitly invoke a flush using `flushStats`. An implicit flush also occurs when statistics collection is disabled.

`setStats` is used to enable and disable statistics collection; the client can optionally control the flush settings by specifying non-`null` values for the time and count parameters.

See Also:

Oracle Database PL/SQL Packages and Types Reference for details of the `DBMS_DBFS_CONTENT` methods

22.19 DBFS Content API Tracing Support

Any DBFS Content API user (both clients and providers) can use DBFS Content API tracing, a generic tracing facility.

The DBFS Content API dispatcher itself uses the tracing facility.

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 two components: `severity` and `detail`. These can be thought of as additive bit masks.

The `severity` component 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 the trace level at any severity they choose, although a good rule of thumb would be to use severity 1 for top-level API entry and exit traces, severity 2 for internal operations, and severity 3 or greater for very low-level traces.

The `detail` component controls how much additional information the trace reports with each trace record: timestamps, short-stack, and so on.

See Also:

- Example 22-1 for more information about how to enable tracing using the DBFS Content APIs.
- *Oracle Database PL/SQL Packages and Types Reference* for details of the `DBMS_DBFS_CONTENT` methods

Example 22-1   DBFS Content Tracing

```
function    getTrace
        return  integer;
    procedure   setTrace(
        trclvl      in              integer);
    function    traceEnabled(
        sev         in              integer)
        return  integer;
    procedure   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 '');
```

22.20 Resource and Property Views

You can see descriptions of Content API structure and properties in certain views.

Certain views describe the structure and properties of Content API.

See Also:

- *Oracle Database Reference* for more information about `DBFS_CONTENT` views
- *Oracle Database Reference* for more information about `DBFS_CONTENT_PROPERTIES` views