184
DBMS_SPACE_ADMIN

The `DBMS_SPACE_ADMIN` package provides functionality for locally managed tablespaces.

This chapter contains the following topics:

- Security Model
- Constants
- Operational Notes
- Summary of DBMS_SPACE_ADMIN Subprograms

See Also:

Oracle Database Administrator's Guide for an example and description of using `DBMS_SPACE_ADMIN`.

DBMS_SPACE_ADMIN Security Model

This package runs with `SYS` privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

DBMS_SPACE_ADMIN Constants

The `DBMS_SPACE_ADMIN` package provides constants that can be used for specifying parameter values.

Table 184-1 DBMS_SPACE_ADMIN Constants

Constant	Type	Value	Description
`SEGMENT_VERIFY_EXTENTS`	`POSITIVE`	`1`	Verifies that the space owned by segment is appropriately reflected in the bitmap as used
`SEGMENT_VERIFY_EXTENTS_GLOBAL`	`POSITIVE`	`2`	Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it
`SEGMENT_MARK_CORRUPT`	`POSITIVE`	`3`	Marks a temporary segment as corrupt whereby facilitating its elimination from the dictionary (without space reclamation)

Table 184-1 (Cont.) DBMS_SPACE_ADMIN Constants

Constant	Type	Value	Description
`SEGMENT_MARK_VALID`	`POSITIVE`	`4`	Marks a corrupt temporary segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.
`SEGMENT_DUMP_EXTENT_MAP`	`POSITIVE`	`5`	Dumps the extent map for a given segment
`TABLESPACE_VERIFY_BITMAP`	`POSITIVE`	`6`	Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent
`TABLESPACE_EXTENT_MAKE_FREE`	`POSITIVE`	`7`	Marks the block range (extent) as free in the bitmaps
`TABLESPACE_EXTENT_MAKE_USED`	`POSITIVE`	`8`	Marks the block range (extent) as used in the bitmaps
`SEGMENT_VERIFY_BASIC`	`POSITIVE`	`9`	Performs the basic metadata checks
`SEGMENT_VERIFY_DEEP`	`POSITIVE`	`10`	Performs deep verification
`SEGMENT_VERIFY_SPECIFIC`	`POSITIVE`	`11`	Performs a specific check for the segment
`HWM_CHECK`	`POSITIVE`	`12`	Checks high water mark (HWM)
`BMB_CHECK`	`POSITIVE`	`13`	Checks integrity among L1, L2 and L3 BMBs (Bit Map Blocks)
`SEG_DICT_CHECK`	`POSITIVE`	`14`	Checks consistency of segment header with corresponding `SEG` entry
`EXTENT_TS_BITMAP_CHECK`	`POSITIVE`	`15`	Checks whether the tablespace bitmaps corresponding to the extent map are marked used
`DB_BACKPOINTER_CHECK`	`POSITIVE`	`16`	Checks whether the L1 BMBs, L2 BMBs, L3 BMBs and data blocks point to the same parent segment
`EXTENT_SEGMENT_BITMAP_CHECK`	`POSITIVE`	`17`	Checks whether the bitmap blocks are consistent with the extent map
`BITMAPS_CHECK`	`POSITIVE`	`18`	Checks from the datablocks that the bitmap states representing the blocks are consistent
`TS_VERIFY_BITMAPS`	`POSITIVE`	`19`	Checks whether the tablespace bitmaps are consistent with the extents belonging to that tablespace
`TS_VERIFY_DEEP`	`POSITIVE`	`20`	Performs `TS_VERIFY_BITMAPS` and `TS_VERIFY_SEGMENTS` with `DEEP` option
`TS_VERIFY_SEGMENTS`	`POSITIVE`	`21`	Performs `ASSM_SEGMENT_VERIFY` on all segments in the tablespace, taking either the `BASIC` or the `DEEP` option
`SEGMENTS_DUMP_BITMAP_SUMMARY`	`POSITIVE`	`27`	Dumps only bitmap block summaries

DBMS_SPACE_ADMIN Operational Notes

Before migrating the `SYSTEM` tablespace, certain conditions must be met. These conditions are enforced by the `TABLESPACE_MIGRATE_TO_LOCAL` procedure, except for the cold backup.

- The database must have a default temporary tablespace that is not `SYSTEM`.
- Dictionary-managed tablespaces cannot have any rollback segments.
- A locally managed tablespace must have at least one online rollback segment. If you are using automatic undo management, then an undo tablespace must be online.
- All tablespaces—except the tablespace containing the rollback segment or the undo tablespace—must be read-only.
- You must have a cold backup of the database.
- The system must be in restricted mode.

Summary of DBMS_SPACE_ADMIN Subprograms

This table lists the `DBMS_SPACE_ADMIN` subprograms and briefly describes them.

Table 184-2 DBMS_SPACE_ADMIN Package Subprograms

Subprogram	Description
ASSM_SEGMENT_VERIFY Procedure	Verifies segments created in ASSM (Automatic Segment-Space Management) tablespaces
ASSM_TABLESPACE_VERIFY Procedure	Verifies ASSM tablespaces
DROP_EMPTY_SEGMENTS Procedure	Drops segments from empty tables or table fragments and dependent objects
GET_SEGADV_ATTRIB Procedure	returns the values of attributes of `DBSM_SPACE_ADMIN` package
MATERIALIZE_DEFERRED_SEGMENTS Procedure	Materializes segments for tables and table fragments with deferred segment creation and their dependent objects
SEGMENT_CORRUPT Procedure	Marks the segment corrupt or valid so that appropriate error recovery can be done
SEGMENT_DROP_CORRUPT Procedure	Drops a segment currently marked corrupt (without reclaiming space)
SEGMENT_DUMP Procedure	Dumps the segment header and extent maps of a given segment
SEGMENT_VERIFY Procedure	Verifies the consistency of the extent map of the segment
SET_SEGADV_ATTRIB Procedure	Sets/changes the values of attributes of `DBSM_SPACE_ADMIN` package
TABLESPACE_FIX_BITMAPS Procedure	Marks the appropriate block range (extent) as free or used in bitmap
TABLESPACE_FIX_SEGMENT_STATES Procedure	Fixes the state of the segments in a tablespace in which migration was aborted
TABLESPACE_MIGRATE_FROM_LOCAL Procedure	Migrates a locally managed tablespace to dictionary-managed tablespace
TABLESPACE_MIGRATE_TO_LOCAL Procedure	Migrates a tablespace from dictionary-managed format to locally managed format

Table 184-2 (Cont.) DBMS_SPACE_ADMIN Package Subprograms

Subprogram	Description
TABLESPACE_REBUILD_BITMAPS Procedure	Rebuilds the appropriate bitmaps
TABLESPACE_REBUILD_QUOTAS Procedure	Rebuilds quotas for given tablespace
TABLESPACE_RELOCATE_BITMAPS Procedure	Relocates the bitmaps to the destination specified
TABLESPACE_VERIFY Procedure	Verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized

ASSM_SEGMENT_VERIFY Procedure

Given a segment definition, the procedure verifies the basic consistency of the space metadata blocks as well as consistency between space metadata and segment data blocks. This procedure verifies segments created in Automatic Segment Space Management (ASSM) tablespaces.

There is however a difference between basic verification and deep verification:

- Basic verification involves consistency checks of space metadata, such as integrity among level 1, level 2, level 3 bitmap blocks, consistency of segment extent map and level 1 bitmap ranges.
- Deep verification involves consistency checks between datablocks and space metadata blocks such as whether the datablocks point correctly to the parent level 1 bitmap blocks, and whether the freeness states in the datablocks are consistent with the freeness states of bits in level 1 bitmap blocks corresponding to the datablocks.

Syntax

```
DBMS_SPACE_ADMIN.ASSM_SEGMENT_VERIFY (
   segment_owner    IN VARCHAR2,
   segment_name     IN VARCHAR2,
   segment_type     IN VARCHAR2,
   partition_name   IN VARCHAR2,
   verify_option    IN POSITIVE  DEFAULT SEGMENT_VERIFY_BASIC,
   attrib           IN POSITIVE  DEFAULT NULL);
```

Parameters

Table 184-3 ASSM_SEGMENT_VERIFY Procedure Parameters

Parameter	Description
`segment_owner`	Schema that owns the segment
`segment_name`	Name of the segment to be verified
`segment_type`	Segment namespace is one of `TABLE`, `TABLE PARTITION`, `TABLE SUBPARTITION`, `INDEX`, `INDEX PARTITION`, `INDEX SUBPARTITION`, `LOB`, `LOB PARTITION`, `LOB SUBPARTITION`, `CLUSTER`
`partition_name`	Name of the partition or subpartition

Table 184-3 (Cont.) ASSM_SEGMENT_VERIFY Procedure Parameters

Parameter	Description
verify_option	One of the following options: • SEGMENT_VERIFY_BASIC := 9. Performs the basic metadata checks (Default) • SEGMENT_VERIFY_DEEP := 10. Performs deep verification • SEGMENT_VERIFY_SPECIFIC := 11. Performs a specific check for the segment
attrib	When option SEGMENT_VERIFY_SPECIFIC is specified as option, attrib can be one of the following: • HWM_CHECK := 12. Checks whether high water mark information is accurate • BMB_CHECK := 13. Checks whether space bitmap blocks have correct backpointers to the segment header • SEG_DICT_CHECK := 14. Checks whether dictionary information for segment is accurate • EXTENT_TS_BITMAP_CHECK := 15. Checks whether extent maps are consistent with file level bitmaps • DB_BACKPOINTER_CHECK := 16. Checks whether datablocks have correct backpointers to the space metadata blocks • EXTENT_SEGMENT_BITMAP_CHECK := 17. Checks whether extent map in the segment matches the bitmaps in the segment • BITMAPS_CHECK := 18. Checks whether space bitmap blocks are accurate

Usage Notes

- Using this procedure requires SYSDBA privileges.
- You can determine the relative file # and header block # (header_relative_file and header_block parameters) by querying DBA_SEGMENTS.
- This procedure outputs a dump file named sid_ora_*process_ID*.trc to the location specified in the USER_DUMP_DEST initialization parameter.

ASSM_TABLESPACE_VERIFY Procedure

This procedures verifies all the segments created in an ASSM tablespace. The verification for each segment performs basic consistency checks of the space metadata blocks as well as consistency checks between space metadata and segment data blocks.

Syntax

```
DBMS_SPACE_ADMIN.ASSM_TABLESPACE_VERIFY (
   tablespace_name    IN VARCHAR2,
   ts_option          IN POSITIVE,
   segment_option     IN POSITIVE DEFAULT NULL);
```

Parameters

Table 184-4 ASSM_TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
`tablespace_name`	Name of the tablespace to verify. The tablespace must be an ASSM tablespace.
`ts_option`	• `19`: SecureFiles supports `DBMS_SPACE_ADMIN.ASSM_TABLESPACE_VERIFY` only when you set the value of `TS_OPTION` to `19`. • `TS_VERIFY_BITMAPS` := 19. The bitmaps are verified against the extents. This detects bits that are marked used or free wrongly and detects multiple allocation of extents. The file metadata is validated against `file$` and control file. • `TS_VERIFY_DEEP` := 20. This option is used to verify the file bitmaps as well perform checks on all the segments. • `TS_VERIFY_SEGMENTS` := 21. This option is used to invoke `SEGMENT_VERIFY` on all the segments in the tablespace. Optionally you can write a script that queries all the segments in the tablespace and invoke `SEGMENT_VERIFY`.
`segment_option`	When `TS_VERIFY_SEGMENTS` is specified, `segment_option` can be one of the following: • `SEGMENT_VERIFY_BASIC` := 9 • `SEGMENT_VERIFY_DEEP` := 10 The value of `segment_option` is `NULL` when `TS_VERIFY_DEEP` or `TS_VERIFY_BITMAPS` is specified.

Usage Notes

- Using this procedure requires `SYSDBA` privileges.
- This procedure outputs a dump file named `sid_ora_`*`process_ID`*`.trc` to the location specified in the `USER_DUMP_DEST` initialization parameter.

DROP_EMPTY_SEGMENTS Procedure

This procedures drops segments from empty tables or table fragments and dependent objects.

Syntax

```
DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS (
   schema_name         IN      VARCHAR2   DEFAULT NULL,
   table_name          IN      VARCHAR2   DEFAULT NULL,
   partition_name      IN      VARCHAR2   DEFAULT NULL);
```

Parameters

Table 184-5 DROP_EMPTY_SEGMENTS Procedure Parameters

Parameter	Description
`schema_name`	Name of schema
`table_name`	Name of table

Table 184-5 (Cont.) DROP_EMPTY_SEGMENTS Procedure Parameters

Parameter	Description
`partition_name`	Name of partition

Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the table or any of its fragments are found to be empty, and the table satisfies certain criteria (restrictions being the same as those described in "Restrictions on Deferred Segment Creation"), then the empty table fragment and associated index segments are dropped along with the corresponding LOB data and index segments. A subsequent insert creates segments with the same properties.

Optionally:

- No `schema_name` is specified, in which case tables belonging to all schemas are scanned
- Both `schema_name` and `table_name` are specified to perform the operation on a specified table
- All three arguments are supplied, restricting the operation to the partition and its dependent objects

GET_SEGADV_ATTRIB Procedure

This procedure returns the values of attributes of `DBMS_SPACE_ADMIN` package.

Syntax

```
DBMS_SPACE_ADMIN.GET_SEGADV_ATTRIB(
   attribute   IN  NUMBER,
   value       OUT NUMBER);
```

Parameters

Table 184-6 GET_SEGADV_ATTRIB Procedure Parameters

Parameter	Description
`attribute`	Supported attributes: • `COMP_ADVISOR` — Provides an option to enable or disable Compression Advisor for Automatic Segment Advisor. By default Compression Advisor is enabled for Automatic Segment Advisor. • `COMP_LOB` — Provides an option to enable or disable Compression Advisor for the tables with LOB columns while Automatic Segment Advisor is running. By default Compression Advisor is enabled for tables with LOB columns.
`value`	Supported values: • `ATTR_ENABLE : 1` • `ATTR_DISABLE : 0`

MATERIALIZE_DEFERRED_SEGMENTS Procedure

This procedure materializes segments for tables and table fragments with deferred segment creation and their dependent objects.

Syntax

```
DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS (
   schema_name       IN     VARCHAR2   DEFAULT NULL,
   table_name        IN     VARCHAR2   DEFAULT NULL,
   partition_name    IN     VARCHAR2   DEFAULT NULL);
```

Parameters

Table 184-7 MATERIALIZE_DEFERRED_SEGMENTS Procedure Parameters

Parameter	Description
`schema_name`	Name of schema
`table_name`	Name of table
`partition_name`	Name of partition

Usage Notes

Given a schema name, this procedure scans all tables in the schema. For each table, if the deferred or delayed segment property is set for the table or any of its fragments, then a new segment is created for those fragments and their dependent objects.

Optionally:

- No `schema_name` is specified, in which case tables belonging to all schemas are scanned
- Both `schema_name` and `table_name` are specified to perform the operation on a specified table
- All three arguments are supplied, restricting the operation to the partition and its dependent objects

SEGMENT_CORRUPT Procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be performed.

It cannot be used on the `SYSTEM` tablespace.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    NUMBER,
   corrupt_option          IN    POSITIVE   DEFAULT SEGMENT_MARK_CORRUPT);
```

Parameters

Table 184-8 SEGMENT_CORRUPT Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace in which segment resides
`header_relative_file`	Relative file number of segment header
`header_block`	Block number of segment header
`corrupt_option`	`SEGMENT_MARK_CORRUPT` (default) or `SEGMENT_MARK_VALID`

Usage Noes

You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS`.

Examples

The following example marks the segment as corrupt:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33,
DBMS_SPACE_ADMIN.SEGMENT_MARK_CORRUPT);
```

Alternately, the next example marks a corrupt segment valid:

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33,
DBMS_SPACE_ADMIN.SEGMENT_MARK_VALID);
```

SEGMENT_DROP_CORRUPT Procedure

This procedure drops a segment currently marked corrupt (without reclaiming space).

For this to work, the segment must be marked *temporary*. To mark a corrupt segment as temporary, issue a `DROP` command on the segment.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (
   tablespace_name          IN    VARCHAR2,
   header_relative_file     IN    POSITIVE,
   header_block             IN    NUMBER);
```

Parameters

Table 184-9 SEGMENT_DROP_CORRUPT Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace in which segment resides
`header_relative_file`	Relative file number of segment header
`header_block`	Block number of segment header

Usage Notes

- The space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS Procedure or the TABLESPACE_REBUILD_BITMAPS Procedure.
- The procedure cannot be used on the `SYSTEM` tablespace.
- You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS`.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);
```

SEGMENT_DUMP Procedure

This procedure dumps the segment header and bitmap blocks of a specific segment to the location specified in the `USER_DUMP_DEST` initialization parameter.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    NUMBER,
   dump_option             IN    POSITIVE  DEFAULT SEGMENT_DUMP_EXTENT_MAP);
```

Parameters

Table 184-10 SEGMENT_DUMP Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace in which segment resides
`header_relative_file`	Relative file number of segment header
`header_block`	Block number of segment header
`dump_option`	One of the following options: • `SEGMENT_DUMP_EXTENT_MAP` • `SEGMENT_DUMP_BITMAP_SUMMARY`

Usage Notes

- You can produce a slightly abbreviated dump, which includes the segment header and bitmap block summaries, without percent-free states of each block if you pass `SEGMENT_DUMP_BITMAP_SUMMARY` as the `dump_option` parameter.
- You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS.HEADER_FILE`. If `HEADER_FILE` is greater than 1023 then use `DBA_DATA_FILES.RELATIVE_FNO`.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);
```

SEGMENT_VERIFY Procedure

This procedure checks the consistency of the segment extent map with the tablespace file bitmaps.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    NUMBER,
   verify_option           IN    POSITIVE  DEFAULT SEGMENT_VERIFY_EXTENTS);
```

Parameters

Table 184-11 SEGMENT_VERIFY Procedure Parameters

Parameters	Description
`tablespace_name`	Name of tablespace in which segment resides
`header_relative_file`	Relative file number of segment header
`header_block`	Block number of segment header
`verify_option`	What kind of check to do: `SEGMENT_VERIFY_EXTENTS` or `SEGMENT_VERIFY_EXTENTS_GLOBAL`

Usage Notes

- Anomalies are output as block range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all block ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not considered free, used space considered free, and the same space considered used by multiple segments.
- You can determine the relative file number and block number (`header_relative_file` and `header_block` parameter) of the segment header block by querying `DBA_SEGMENTS`.

Examples

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps synchronized.

```
EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33,
DBMS_SPACE_ADMIN.SEGMENT_VERIFY_EXTENTS);
```

SET_SEGADV_ATTRIB Procedure

This procedure sets the values of attributes of `DBMS_SPACE_ADMIN` package.

Syntax

```
DBMS_SPACE_ADMIN.SET_SEGADV_ATTRIB(
   attribute   IN   NUMBER,
   value       IN   NUMBER);
```

Parameters

Table 184-12 SET_SEGADV_ATTRIB Procedure Parameters

Parameter	Description
`attribute`	Supported attributes: • `COMP_ADVISOR` — Provides an option to enable or disable Compression Advisor for Automatic Segment Advisor. By default Compression Advisor is enabled for Automatic Segment Advisor. • `COMP_LOB` — Provides an option to enable or disable Compression Advisor for the tables with LOB columns while Automatic Segment Advisor is running. By default Compression Advisor is enabled for tables with LOB columns.
`value`	Supported values: • `ATTR_ENABLE : 1` • `ATTR_DISABLE : 0`

TABLESPACE_FIX_BITMAPS Procedure

This procedure marks the appropriate block range (extent) as free or used in bitmap. It cannot be used on the `SYSTEM` tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS (
   tablespace_name          IN    VARCHAR2,
   dbarange_relative_file   IN    POSITIVE,
   dbarange_begin_block     IN    POSITIVE,
   dbarange_end_block       IN    POSITIVE,
   fix_option               IN    POSITIVE);
```

Parameters

Table 184-13 TABLESPACE_FIX_BITMAPS Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace
`dbarange_relative_file`	Relative file number of block range (extent)
`dbarange_begin_block`	Block number of beginning of extent
`dbarange_end_block`	Block number (inclusive) of end of extent
`fix_option`	One of the following options: • `TABLESPACE_EXTENT_MAKE_FREE` • `TABLESPACE_EXTENT_MAKE_USED`

Examples

The following example marks bits for 51 blocks for relative file number 4, beginning at block number 33 and ending at 83, as `USED` in bitmaps.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83,
DBMS_SPACE_ADMIN.EXTENT_MAKE_USED);
```

Alternatively, specifying an option of `TABLESPACE_EXTENT_MAKE_FREE` marks the bits free in bitmaps. The `BEGIN` and `END` blocks must be in extent boundary and be extent multiple; otherwise, an error is raised.

TABLESPACE_FIX_SEGMENT_STATES Procedure

This procedure fixes the state of the segments in a tablespace in which migration was aborted.

During tablespace migration to or from local, the segments are put in a transient state. If migration is aborted, then the segment states are corrected by SMON when event 10906 is set. A database with segments in such a transient state cannot be downgraded. The procedure can be used to fix the state of such segments.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES (
   tablespace_name     IN    VARCHAR);
```

Parameters

Table 184-14   TABLESPACE_FIX_SEGMENT_STATES Procedure Parameters

Parameter Name	Description
`tablespace_name`	Name of the tablespace whose segments must be fixed

Usage Notes

The tablespace must be kept online and read/write when this procedure is called.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_SEGMENT_STATES('TS1')
```

TABLESPACE_MIGRATE_FROM_LOCAL Procedure

This procedure migrates a locally managed tablespace to a dictionary-managed tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
   tablespace_name         IN    VARCHAR2);
```

Parameter

Table 184-15   TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameter

Parameter	Description
`tablespace_name`	Name of tablespace

Usage Notes

The tablespace must be kept online and read/write during migration. Migration of temporary tablespaces and migration of `SYSTEM` tablespaces are not supported.

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');
```

TABLESPACE_MIGRATE_TO_LOCAL Procedure

This procedure migrates the tablespace from a dictionary-managed format to a locally managed format. Tablespaces migrated to locally managed format are user managed.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL (
   tablespace_name      IN     VARCHAR2,
   unit_size            IN     POSITIVE DEFAULT NULL,
   rfno                 IN     POSITIVE DEFAULT NULL);
```

Parameters

Table 184-16 TABLESPACE_MIGRATE_TO_LOCAL Procedure Parameters

Parameter Name	Description
`tablespace_name`	Name of the tablespace to be migrated
`unit_size`	Bitmap unit size (which is the size of the smallest possible chunk of space that can be allocated) in the tablespace specified in number of blocks
`rfno`	Relative File Number of the file where the bitmap blocks are placed

Usage Notes

- Before you migrate the `SYSTEM` tablespace, migrate any dictionary-managed tablespaces that you want to use in read/write mode to locally managed. After the `SYSTEM` tablespace is migrated, you cannot change dictionary-managed tablespaces to read/write.

  See Also:

  Oracle Database Administrator's Guide

- The tablespace must be kept online and read/write during migration. Note that temporary tablespaces cannot be migrated.
- Allocation Unit may be specified optionally. The default is calculated by the system based on the highest common divisor of all extents (used or free) for the tablespace. This number is further trimmed based on the `MINIMUM EXTENT` for the tablespace (5 if `MINIMUM EXTENT` is not specified). Thus, the calculated value will not be larger than the `MINIMUM EXTENT` for the tablespace. The last free extent in every file is ignored for GCD calculation. If you specify the unit size, then it must be a factor of the `unit_size` calculated by the system; otherwise an error message is returned.
- The Relative File Number parameter is used to place the bitmaps in a desired file. If space is not found in the file, then an error is issued. The data file specified must be part of the tablespace being migrated. If the dataflow is not specified, then the system chooses a dataflow in which to place the initial bitmap blocks. If space is not found for the initial bitmaps, then an error is raised.

Examples

To migrate a tablespace 'TS1' in 2KB blocksize with minimum extent size 1MB:

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('TS1', 512, 2);
```

The bitmaps are placed in file with relative file number 2.

TABLESPACE_REBUILD_BITMAPS Procedure

This procedure rebuilds the appropriate bitmaps. If no bitmap block is specified, then it rebuilds all bitmaps for the given tablespace.

The procedure cannot be used on the `SYSTEM` tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   bitmap_relative_file    IN    POSITIVE   DEFAULT NULL,
   bitmap_block            IN    POSITIVE   DEFAULT NULL);
```

Parameters

Table 184-17 TABLESPACE_REBUILD_BITMAPS Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace
`bitmap_relative_file`	Relative file number of bitmap block to rebuild
`bitmap_block`	Block number of bitmap block to rebuild

Usage Notes

Only full rebuild is supported.

Examples

The following example rebuilds bitmaps for all the files in the `USERS` tablespace.

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

TABLESPACE_REBUILD_QUOTAS Procedure

This procedure rebuilds quotas for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (
   tablespace_name         IN    VARCHAR2);
```

Parameters

Table 184-18 TABLESPACE_REBUILD_QUOTAS Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('USERS');
```

TABLESPACE_RELOCATE_BITMAPS Procedure

This procedure relocates the bitmaps to the destination specified.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS (
   tablespace_name    IN      VARCHAR2,
   filno              IN      POSITIVE,
   blkno              IN      POSITIVE);
```

Parameters

Table 184-19 TABLESPACE_RELOCATE_BITMAPS Procedure Parameters

Parameter Name	Description
`tablespace_name`	Name of tablespace
`filno`	Relative File Number of the destination file
`blkno`	Block Number of the destination range

Usage Notes

- Migration of a tablespace from dictionary-managed to locally managed format could result in the creation of `SPACE HEADER` segment that contains the bitmap blocks. The `SPACE HEADER` segment is treated as user data. If you explicitly resize a file at or below the space header segment, then an error is issued. Use the `TABLESPACE_RELOCATE_BITMAPS` command to move the control information to a different destination and then resize the file.
- This procedure cannot be used on the `SYSTEM` tablespace.
- The tablespace must be kept online and read/write during relocation of bitmaps. This can be done only on migrated locally managed tablespaces.

Examples

```
EXECUTE  DBMS_SPACE_ADMIN.TABLESPACE_RELOCATE_BITMAPS('TS1', 3, 4);
```

Moves the bitmaps to file 3, block 4.

Note:

The source and the destination addresses must not overlap. The destination block number is rounded down to the unit boundary. If there is user data in that location, then an error is raised.

TABLESPACE_VERIFY Procedure

This procedure verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (
   tablespace_name         IN     VARCHAR2,
   verify_option           IN     POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

Table 184-20 TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
`tablespace_name`	Name of tablespace
`verify_option`	One option is supported: `TABLESPACE_VERIFY_BITMAP`

Examples

```
EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('USERS');
```