

Moving Oracle Databases Across Platforms without Export/Import

by Roger Schrag¹
Database Specialists, Inc.²

Introduction

Prior to Oracle 10g, one of the only supported ways to move an Oracle database across platforms was to export the data from the existing database and import it into a new database on the new server. This works pretty well if your database is small, but can require an unreasonable amount of down time if your database is large. In Oracle 10g, the transportable tablespace feature has been enhanced in a way that makes it possible to move large databases (or portions of them) across platforms much more quickly and simply than the export/import method. We will begin this paper with a high-level look at Oracle 10g's cross-platform support for transportable tablespaces and possible uses for this feature. Then we'll walk through a real-life case study of moving an Oracle database from a Sun Enterprise 450 server running the Solaris operating system on SPARC processors to a Dell server running Red Hat Enterprise Linux on Intel processors. We'll look at each of the steps in detail and show all of the commands used and resulting output from Oracle utilities. We will wrap up with a discussion of pitfalls, limitations, and things to keep in mind when preparing to move Oracle data across platforms.

Cross-platform Transportable Tablespaces: An Overview

The transportable tablespace feature, introduced in Oracle 8i, allows you to copy one or more tablespaces from one database to another. The export and import utilities are used to copy the metadata for the objects residing in the transported tablespaces, and ordinary file copy commands like FTP or SCP are used to copy the actual data files. This feature made sharing information

¹ Roger Schrag has been an Oracle DBA and application architect for over fifteen years. He started out at Oracle Corporation on the Oracle Financials development team and moved into the roles of production DBA and database architect at various companies in the San Francisco Bay Area. In 1995, Roger founded Database Specialists, Inc. which provides remote DBA services and onsite database support for mission-critical Oracle systems. Roger is a frequent speaker at Oracle Open World and the IOUG Live! conferences. He is also Director of Conference Programming for the Northern California Oracle Users Group.

² Used in the MSIA by kind permission of Database Specialists. Original URL for HTML version of this paper is < http://www.dbspecialists.com/files/presentations/changing_platforms.html >. A PowerPoint slide show (not narrated) is also available for download. < http://www.dbspecialists.com/files/presentations/changing_platforms.ppt >

Moving Oracle Databases Across Platforms without Export/Import

between databases, or flowing data from one database to another, fast and efficient. Transporting tablespaces is fast because Oracle only needs to write the metadata—or schema object definitions—into the export file. This saves the time-consuming and resource intensive steps of exporting data into a dump file at the source database, loading the data from the dump file into the target database, and rebuilding indexes in the target database from scratch.

In Oracle 8i and Oracle 9i, tablespaces could only be transported into databases that ran on the same hardware platform and operating system. So if your OLTP system ran on HP-UX and your data warehouse on Linux, you could not use transportable tablespaces to copy data efficiently between the databases. And if you had a database running on a Windows server and you wanted to permanently move it to a Sun server running Solaris, you couldn't use transportable tablespaces either.

Beginning in Oracle 10g release 1, cross-platform support for transportable tablespaces is available for several of the most commonly used platforms. The process is similar to transporting tablespaces in previous Oracle releases, except there are a few possible extra steps, and there are more limitations and restrictions. Oracle 10g release 2 goes one step further and offers the ability to transport an entire database across platforms in one step. But the limitations here are even stricter.

At a high level, transporting tablespaces across platforms works like this: First you make sure your tablespaces qualify for using the cross-platform transportable tablespaces feature. Next you determine if file conversion will be required. (This depends on what platform you are moving from and to.) Next you make the tablespaces read-only on the source database and you export the metadata. If file conversion is required, next you invoke RMAN to perform the file conversion. Next you copy the data files to the target database server and import the metadata. If you want to make the tablespaces read-write on the target database, you can do so.

This paper focuses on using the cross-platform transportable tablespace feature for moving a database permanently from one platform to another as quickly and efficiently as possible. But this feature could also be used on a regular basis for information sharing between databases running on different platforms. In some cases it may even be possible for multiple databases on different platforms to share the same physical data files in read-only mode. Because you can transport tablespaces into a database running a higher Oracle version, it is theoretically possible that you could upgrade an Oracle 10g release 1 database to Oracle 10g release 2 and move it to a new platform all in one step.

Many companies today run Oracle databases on multiple platforms and would like to share information easily and efficiently between systems. Other companies are looking to move their Oracle databases to newer hardware platforms or different operating systems than they are presently using. The introduction of cross-platform support for the transportable tablespace feature can be a great help in these situations.

A Case Study

With the increased acceptance of Linux by the business community, many companies are moving IT systems to commodity hardware running Linux. We are seeing many organizations that want to move Oracle databases from expensive servers running AIX, HP-UX, or Solaris operating systems to less expensive hardware running Linux. In this section of the paper we will discuss in great detail one such project, walking through each of the steps of the process thoroughly.

The company had an Oracle 10g release 2 database running on a Sun Enterprise 450 server running the Solaris operating system which they wanted moved to a Dell server running Red Hat Enterprise Linux. The new feature in Oracle 10g release 2 that allows transporting an entire database across platforms in one step cannot be used to transport between the Sun SPARC processor architecture and the Intel processor architecture, so we used the cross-platform transportable tablespace feature instead. The database, at about 8 Gb, was quite small. We could probably have used the export/import method to move all of the data in a reasonable amount of time, but the goal was to move the database with as little down time as possible.

The Sun and Dell servers in this case study both ran Oracle 10g release 2 with the 10.2.0.2.0 patch set. The procedure should be generally the same for other releases of Oracle 10g, but there may be minor changes as Oracle fixes bugs, improves usability, and enhances functionality. Although the 10.2.0.2.0 patch set was the latest available when this project took place, it is likely that a newer patch set has been released by the time you read this paper. Please bear this in mind and use this paper as a starting point only, and not a substitute for proper planning and preparation.

A note about the export and import utilities: Data pump has been introduced in Oracle 10g as a new and improved method for exporting and importing data in Oracle databases. The export and import utilities we used in Oracle 9i and earlier are still around in Oracle 10g and are now known as “Original” export and import. Both tool sets are supported in Oracle 10g, and you can use either for transporting tablespaces across platforms. One of the advantages of original export and import is stability—these tools have been around a long time and do what they do very well. Data pump, being new, has more bugs and is apt to be less stable.

Whether you should use original export and import or data pump will depend on certain aspects of your database environment and in general how comfortable you are with newer Oracle products. My team favors stability and predictability over new and whiz-bang, so we used original export and import for this project. Here are some data points for you to consider in choosing which tool set you should use:

- Metalink documents 271984.1 and 294992.1 present two examples of documented bugs in data pump where a suggested workaround is to use original export and import instead. There probably are many others. But to be fair, the two documents listed here cite bugs in Oracle 10g release 1 data pump that are claimed to be fixed in Oracle 10g release 2.
- Metalink document 371556.1 points out that data pump cannot transport XMLTypes while original export and import can.

Moving Oracle Databases Across Platforms without Export/Import

- Data pump offers many benefits over original export and import in the areas of performance and job management, but these benefits have little impact when transporting tablespaces because metadata export and import is usually very fast to begin with.
- Original export and import are not obvious winners, however. Original export and import cannot transport BINARY_FLOAT and BINARY_DOUBLE data types, while data pump can.
- When original export and import transport a tablespace that contains materialized views, the materialized views will be converted into regular tables on the target database. Data pump, on the other hand, keeps them as materialized views.

Step 1: Check Platform Support and File Conversion Requirement

First we need to verify that Oracle supports cross-platform tablespace transport between the two platforms we have in mind. The v\$database view shows the name of the platform (as Oracle sees it) and the v\$transportable_platform view shows a list of all supported platforms. If we join these two together in a query, we get the information we need.

On the source database:

```
SQL> SELECT A.platform_id, A.platform_name, B.endian_format
  2 FROM   v$database A, v$transportable_platform B
  3 WHERE  B.platform_id (+) = A.platform_id;
```

| PLATFORM_ID | PLATFORM_NAME | ENDIAN_FORMAT |
|-------------|-------------------------|---------------|
| 2 | Solaris[tm] OE (64-bit) | Big |

```
SQL>
```

On the target database:

```
SQL> SELECT A.platform_id, A.platform_name, B.endian_format
  2 FROM   v$database A, v$transportable_platform B
  3 WHERE  B.platform_id (+) = A.platform_id;
```

| PLATFORM_ID | PLATFORM_NAME | ENDIAN_FORMAT |
|-------------|-------------------|---------------|
| 10 | Linux IA (32-bit) | Little |

```
SQL>
```

The endian_format column in this query's output will show either "Big," "Little," or will be blank. A blank indicates that the platform is not supported for cross-platform tablespace transport. If the source and target platform have the same endian format, then file conversion will not be necessary. If the endian formats differ, however, then file conversion will be required. (Endian format describes the order in which a processor architecture natively places bytes in memory, a CPU register, or a file. Some processor architectures start with the low-order byte and work their way up

Moving Oracle Databases Across Platforms without Export/Import

while others start with the high-order byte and work their way down. If the source and target platform have different endian formats, then file conversion is necessary to reverse byte orderings in data blocks.)

As you can see from the above query output, we were moving from the Solaris operating environment (which refers to the SPARC processor architecture as opposed to Solaris x86) to Linux on the Intel processor architecture. Both platforms are supported, but file conversion will be required since we are going from a big to a little endian format.

Oracle 10g release 2 with the 10.2.0.2.0 patch set supports cross-platform tablespace transport for 17 platforms including varieties of Solaris, Linux, HP-UX, AIX, Mac OS, Tru64, Open VMS, and Windows.

Step 2: Identify Tablespaces to be Transported and Verify Self-containment

Now that we know we can transport tablespaces between the platforms of our source and target database servers, we need to figure out which tablespaces we want to transport. We only want to transport tablespaces containing application data—there is no need to transport the SYSTEM, undo, or temporary tablespaces. And obviously we don't want to transport tablespaces that contain data we don't want to appear on the target database.

In our case, all of the data for the application we are interested in resided in one schema called DBRD. Moreover, we wanted all of the data in this schema to be transported. So we could identify the tablespaces with the following simple query run on the source database:

```
SQL> SELECT tablespace_name, segment_type, COUNT(*),
2          SUM (bytes) / 1024 / 1024 mb
3 FROM dba_segments
4 WHERE owner = 'DBRD'
5 GROUP BY tablespace_name, segment_type
6 ORDER BY 1, 2 DESC;
```

| TABLESPACE_NAME | SEGMENT_TYPE | COUNT (*) | MB |
|-----------------|--------------|-----------|--------|
| IND1 | INDEX | 88 | 1353.4 |
| TAB1 | TABLE | 41 | 4079.6 |
| TAB1 | LOBSEGMENT | 3 | 0.4 |
| TAB1 | LOBINDEX | 3 | 0.2 |
| TAB1 | INDEX | 53 | 106.4 |

```
SQL>
```

You can transport one or more tablespaces at a time. You cannot transport part of a tablespace. From the query output above, it appears that we need to transport the TAB1 and IND1 tablespaces.

The set of tablespaces to be transported in one set must be “self-contained” meaning that objects in the tablespace set cannot reference or depend on objects that reside outside of the tablespace. For

Moving Oracle Databases Across Platforms without Export/Import

example, if a table in the TAB2 tablespace had an index in the IND1 tablespace, then transporting the TAB1 and IND1 tablespaces (without the TAB2 tablespace) would present a problem. When the TAB1 and IND1 tablespaces are transported into the target database, there would be an index on a non-existent table. Oracle will not allow this and will point out the problem while exporting the metadata. Some other examples of self-containment problems are:

- A table in the transported tablespace set with a LOB segment outside the set.
- A LOB segment in the transported tablespace set with the table segment it belongs to outside the set.
- A table or index partition in the transported tablespace set with one or more partitions outside the set.
- A table in the transported tablespace set with an index outside the set that enforces a declared unique or primary key constraint.
- A table in the transported tablespace set with a foreign key constraint that references a table outside the set. (This is only an issue if constraints are transported).

It is important to note that there are certain situations where a tablespace set will not be strictly self-contained and no error will occur during the transport process, but a side-effect will occur that may or may not be a problem for you. For example, if a table in the transported tablespace set has an index outside of the set and that index does not enforce a declared constraint, no error will occur and the tablespace transport will be successful. However, the index outside of the set will not be present on the target database.

Before we begin the transport process, we can verify that the tablespace set we have chosen is sufficiently self-contained for the transport operation to succeed. It is better to check now than to wait and see if the export fails, because the check procedure shown here will list for you any of the problems while the export will simply fail without telling you how the self-containment requirement was violated.

We ran the following on the source database to verify there were no self-containment problems:

```
SQL> BEGIN
  2   SYS.dbms_tts.transport_set_check
  3   ('TAB1,IND1', incl_constraints=>TRUE, full_check=>FALSE);
  4 END;
  5 /

PL/SQL procedure successfully completed.

SQL> SELECT * FROM SYS.transport_set_violations;

no rows selected

SQL>
```

Note that public synonyms are still missing as of Oracle 10g release 2 with the 10.2.0.2.0 patch set (or were perhaps left out intentionally) and thus the SYS schema needs to be specified when referencing dbms_tts or transport_set_violations.

Moving Oracle Databases Across Platforms without Export/Import

Since we will be transporting constraint definitions (the default behavior), we specified the inclusion of constraints in the self-containment check by setting the `incl_constraints` parameter to `TRUE`. Setting `full_check` to `TRUE` will verify that the tablespace set is completely self-contained, while setting it to `FALSE` will only check that the set is sufficiently self-contained for the transport to succeed. If you recall the example we mentioned earlier of the index outside the transported tablespace set that would get lost in the transport, a full check would point out this condition while the check we did here would not.

If there had been an index in tablespace `IND1` that belonged to a table outside of the tablespace set, we would have seen a violation like:

```
SQL> SELECT * FROM SYS.transport_set_violations;

VIOLATIONS
-----
-----
Index MY_SCHEMA.MY_INDEX in tablespace IND1 points to table
MY_SCHEMA.MY_TABLE
in tablespace TAB2

SQL>
```

If there had been a table in the `TAB1` tablespace with a foreign key referencing a table outside of the tablespace set, we would have seen a violation like:

```
SQL> SELECT * FROM SYS.transport_set_violations;

VIOLATIONS
-----
-----
Constraint MY_CHILD_TABLE_FK1 between table MY_SCHEMA.MY_PARENT_TABLE
in
tablespace TAB2 and table MY_SCHEMA.MY_CHILD_TABLE in tablespace TAB1

SQL>
```

Step 3: Check for Problematic Data Types

As a proactive measure you can check the data types of the columns in the tables you are planning to transport in order to address potential issues as early in the process as possible. As we pointed out earlier, data pump is not able to transport XMLTypes, while original export and import are not able to transport `BINARY_FLOAT` or `BINARY_DOUBLE` data. Furthermore, there are several “opaque” data types including `RAW`, `LONG RAW`, `BFILE`, `ANYTYPE`, and user-defined data types. Because of the unstructured nature of these data types, Oracle does not know if data in these columns will be platform-independent or require byte swapping for endian format change. Oracle simply transports these data types as-is and leaves conversion to the application.

Moving Oracle Databases Across Platforms without Export/Import

We ran the following queries on the source database in order to survey the data types used in our tablespace set:

```
SQL> SELECT B.data_type, COUNT(*)
2 FROM dba_tables A, dba_tab_columns B
3 WHERE A.owner = 'DBRD'
4 AND B.owner = A.owner
5 AND B.table_name = A.table_name
6 GROUP BY B.data_type
7 ORDER BY B.data_type;

DATA_TYPE          COUNT(*)
-----
CLOB                3
DATE               153
NUMBER             207
VARCHAR2           237

SQL> SELECT B.owner, B.table_name
2 FROM dba_xml_tables A, all_all_tables B
3 WHERE B.owner = A.owner
4 AND B.table_name = A.table_name
5* AND B.tablespace_name IN ('TAB1', 'IND1');

no rows selected

SQL>
```

The tablespaces we are planning to transport do not appear to have problematic data types.

Step 4: Check for Missing Schemas and Duplicate Tablespace and Object Names

At this point we check schema, tablespace, and object names on the source and target databases in order to ensure that the transport will go smoothly. We may need to create schemas on the target database if they are missing. We may also need to rename tablespaces or schema objects on either the source or target database if there are duplications. We will look at the missing schema issue first and the duplicate tablespace or object names second.

All of the objects in a tablespace set must go into schemas that already exist on the target database. When importing the metadata into the target database you have the opportunity to “remap” schemas, allowing you to transport objects that were in the FRED schema on the source database into the BOB schema on the target database as an example.

At this time you should determine what schemas will be required on the target database and create them if they don't already exist. In our case, we are transporting tablespaces into a near-empty database and we want to keep schema names the same. First we check the source database to double-check what schemas own objects in the tablespaces we are planning to transport:

Moving Oracle Databases Across Platforms without Export/Import

```
SQL> SELECT  owner, COUNT(*)
  2 FROM    dba_segments
  3 WHERE   tablespace_name IN ('TAB1', 'IND1')
  4 GROUP BY owner;

OWNER                                COUNT(*)
-----
DBRD                                  188

SQL>
```

Next we verify that this schema did not yet exist on the target database:

```
SQL> SELECT username
  2 FROM  dba_users
  3 WHERE username = 'DBRD';

no rows selected

SQL>
```

We create the missing schema on the target database now and give it all roles and system privileges required by the application:

```
SQL> CREATE USER dbrd IDENTIFIED BY password;

User created.

SQL> GRANT connect, resource TO dbrd;

Grant succeeded.

SQL> GRANT create library TO dbrd;

Grant succeeded.

SQL> REVOKE unlimited tablespace FROM dbrd;

Revoke succeeded.

SQL>
```

Next we move on to checking for duplicate tablespace or object names. It will not be possible to transport our tablespace set into the target database if a tablespace already exists there with the same name as one of the tablespaces in our set. We can quickly check the target database for a duplicate tablespace name:

```
SQL> SELECT tablespace_name
  2 FROM  dba_tablespaces
  3 WHERE tablespace_name IN ('TAB1', 'IND1');

no rows selected

SQL>
```

Moving Oracle Databases Across Platforms without Export/Import

If there had been a duplication of tablespace names, we could simply rename a tablespace (on the source or target database) with a statement such as:

```
SQL> ALTER TABLESPACE old_tablespace_name RENAME TO
new_tablespace_name;

Tablespace altered.

SQL>
```

Similarly, a transport will fail if an object already exists on the target database in the target schema with the same name as an object in the transported tablespace set. In our case we knew that the schema in which all transported objects reside did not yet exist on the target database. Thus there cannot be any conflicts. If there had been duplicate object names, however, we could avoid problems by renaming objects in either the source or target database.

Step 5: Make Tablespaces Read-only in Source Database

The tablespaces to be transported need to be made read-only on the source database for long enough to copy them and extract the metadata. This could unfortunately mean a service disruption to end users. The good news is that the user impact will likely be much less severe than would have been the case if you were to copy the data to the target database using the export/import method or copying table rows via database links. With today's filers and sophisticated storage systems, it is often possible to take a filer "snapshot" or split a mirror in order to get a copy of the data files very quickly. Extracting metadata is also quick. So, on a system with a good storage system, tablespaces may only need to be read-only for a few minutes.

We put the tablespaces into read-only mode with the following statements:

```
SQL> ALTER TABLESPACE tab1 READ ONLY;

Tablespace altered.

SQL> ALTER TABLESPACE ind1 READ ONLY;

Tablespace altered.

SQL>
```

Step 6: Extract Metadata from Source Database

We are now ready to extract the metadata from the source database. We could use either data pump or original export to do this. As we discussed earlier, we chose to use original export. The export command and output were as follows:

```
$ exp ''/ as sysdba' file=PROD417_tablind1.dmp transport_tablespace=y
\
>          tablespaces=tab1,ind1

Export: Release 10.2.0.2.0 - Production on Mon Dec 18 12:58:00 2006

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.2.0
- 64bit
Production
With the Partitioning, OLAP and Data Mining options
Export done in WE8ISO8859P1 character set and AL16UTF16 NCHAR character
set
Note: table data (rows) will not be exported
About to export transportable tablespace metadata...
For tablespace TAB1 ...
. exporting cluster definitions
. exporting table definitions
. . exporting table          COMMON_BANNER_SETS
. . exporting table          COMMON_BANNER_TYPES
...
. . exporting table          TXN_COMMENTS
. . exporting table          TXN_LINES
For tablespace IND1 ...
. exporting cluster definitions
. exporting table definitions
. exporting referential integrity constraints
. exporting triggers
. end transportable tablespace metadata export
Export terminated successfully without warnings.
$
```

Note that the export must be performed while connected to the database with SYSDBA privilege. The export session took under two minutes to run, and the dump file generated was 640 Kb in size. The tablespaces to be transported contained 94 tables with over 4 Gb of table data, so you can see how much faster it is to export the metadata than the actual data itself.

Step 7: Copy Files to Target Server and Convert if Necessary

We are now ready to copy the export file and all of the data files that make up the transported tablespace set to the server where the target database resides. We can use any conventional file transfer mechanism, such as FTP (in binary mode!) or SCP. If your storage device has the ability to snapshot the files or split a mirror, then you can get a near-instant copy of the data files and mount the copy on the target database server.

If you wish to minimize the amount of time that the tablespaces are in read-only mode on the source database, then you can return them to read-write mode as soon as the metadata has been exported and you have a complete copy of all the required data files. In our case, where we were moving the database permanently from one server to another, we left the tablespaces on the source database in read-only mode to prevent users from accidentally writing data to the old database.

If we determined back in step 1 that file conversion is necessary when transporting tablespaces between our source and target platforms, then we must use RMAN to convert the data files at this time. The conversion may be run on either the source or target database server, although the syntax of the RMAN commands will differ slightly. You may want to consider server and I/O system speed when choosing where to perform the conversion. If either server hosts other production databases that you don't want to impact negatively by an I/O-intensive conversion operation, then this will also influence your decision as to where to perform the conversion.

In our case, we noted earlier that file conversion is indeed required. We did not see a strategic advantage to performing the conversion on one server versus the other, so during testing we tried both just for comparison's sake. In our case both the source and target servers took approximately 14 minutes to convert the data files.

The RMAN command and output to convert data files on the source database were as follows:

```
$ rman

Recovery Manager: Release 10.2.0.2.0 - Production on Tue Dec 19
16:47:30 2006

Copyright (c) 1982, 2005, Oracle. All rights reserved.

RMAN> CONNECT TARGET

connected to target database: PROD417 (DBID=3437408061)

RMAN> CONVERT TABLESPACE tab1, ind1
2> TO PLATFORM = "Linux IA (32-bit)"
3> DB_FILE_NAME_CONVERT ('/u03/oradata/PROD417/', '/u03/stage/');

Starting backup at 19-DEC-06
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=149 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
```

Moving Oracle Databases Across Platforms without Export/Import

```
input datafile fno=00008 name=/u03/oradata/PROD417/tab103.dbf
converted datafile=/u03/stage/tab103.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:05:36
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00006 name=/u03/oradata/PROD417/ind101.dbf
converted datafile=/u03/stage/ind101.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:03
channel ORA_DISK_1: starting datafile conversion
input datafile fn:55
channel ORA_DISK_1: starting datafile conversion
input datafile fno=00005 name=/u03/oradata/PROD417/tab101.dbf
converted datafile=/u03/stage/tab101.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:01:55
o=00007 name=/u03/oradata/PROD417/tab102.dbf
converted datafile=/u03/stage/tab102.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:02:17
Finished backup at 19-DEC-06

RMAN> EXIT

Recovery Manager complete.
$
```

Note that on the source database we specify a list of tablespaces to convert, and we leave it to RMAN to access the data dictionary for the database and determine which data files make up the specified tablespaces. We used the `db_file_name_convert` parameter in order to cause RMAN to write the converted files into a staging directory, preserving the original base name of each data file. At this point we can copy the converted data files from the source database server to the target database server. On the target, we should go ahead and place the files directly into the directory where we will want them to reside permanently when they become part of the target database.

Alternatively, we could have copied the data files from the source database server to a staging directory on the target database server and performed the file conversion on the target database server. The RMAN command and output to convert the data files on the target database server were as follows:

```
$ rman

Recovery Manager: Release 10.2.0.2.0 - Production on Wed Dec 20
10:11:38 2006

Copyright (c) 1982, 2005, Oracle. All rights reserved.

RMAN> CONNECT TARGET

connected to target database: PROD463 (DBID=2124019545)

RMAN> CONVERT DATAFILE '/u01/stage/tab101.dbf',
2>                        '/u01/stage/tab102.dbf',
3>                        '/u01/stage/tab103.dbf',
```

Moving Oracle Databases Across Platforms without Export/Import

```
4>          '/u01/stage/ind101.dbf'
5> FROM PLATFORM "Solaris[tm] OE (64-bit)"
6> DB_FILE_NAME_CONVERT ('/u01/stage/',
7>          '/u01/oradata/PROD463/');

Starting backup at 20-DEC-06
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=145 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
input filename=/u01/stage/tab103.dbf
converted datafile=/u01/oradata/PROD463/tab103.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:05:47
channel ORA_DISK_1: starting datafile conversion
input filename=/u01/stage/ind101.dbf
converted datafile=/u01/oradata/PROD463/ind101.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:04:16
channel ORA_DISK_1: starting datafile conversion
input filename=/u01/stage/tab101.dbf
converted datafile=/u01/oradata/PROD463/tab101.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:01:16
channel ORA_DISK_1: starting datafile conversion
input filename=/u01/stage/tab102.dbf
converted datafile=/u01/oradata/PROD463/tab102.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time:
00:01:26
Finished backup at 20-DEC-06

RMAN> EXIT

Recovery Manager complete.
$
```

Note that on the target database server we specify a list of data files to convert instead of a list of tablespaces, since RMAN would not be able to identify the files by accessing the target database's data dictionary. Again we used the `db_file_name_convert` parameter—this time so that RMAN will read the unconverted files from a staging directory and write the converted files directly into the directory where we want them to reside permanently when they become part of the target database.

Step 8: Import Metadata into Target Database

We are now ready to load the metadata extracted from the source database into the target database. This will make the data files we copied to the target database server part of the target database and will add objects to the target database's data dictionary. This step is sometimes called “plugging in” the tablespaces. Again we can use data pump or original import, and we've chosen to use original import. The import command and output were as follows:

```
$ imp ''/ as sysdba' file=PROD417_tablind1.dmp transport_tablespace=y
\
```

Moving Oracle Databases Across Platforms without Export/Import

```
>      datafiles=/u01/oradata/PROD463/ind101.dbf,
/u01/oradata/PROD463/tab101.dbf, \
>      /u01/oradata/PROD463/tab102.dbf,
/u01/oradata/PROD463/tab103.dbf

Import: Release 10.2.0.2.0 - Production on Wed Dec 20 16:32:51 2006

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.2.0
- Produc
tion
With the Partitioning, OLAP and Data Mining options

Export file created by EXPORT:V10.02.01 via conventional path
About to import transportable tablespace(s) metadata...
import done in WE8ISO8859P1 character set and AL16UTF16 NCHAR character
set
. importing SYS's objects into SYS
. importing SYS's objects into SYS
. importing DBRD's objects into DBRD
.. importing table          "COMMON_BANNER_SETS"
.. importing table          "COMMON_BANNER_TYPES"
...
.. importing table          "TXN_COMMENTS"
.. importing table          "TXN_LINES"
About to enable constraints...
. importing SYS's objects into SYS
Import terminated successfully without warnings.
$
```

The import session took less than two minutes to complete. Note that we could have specified the fromuser and touser parameters (remap_schema if using data pump) if we wanted the objects to reside in different schemas than they did on the source database. Also note that it is acceptable for the data files on the target database to have different path names than they had on the source database.

At this point all of the objects transported from the source database are available for use in the target database on a read-only basis. We changed the tablespaces to read-write mode on the target database so that transaction activity could begin:

```
SQL> ALTER TABLESPACE tab1 READ WRITE;

Tablespace altered.

SQL> ALTER TABLESPACE ind1 READ WRITE;

Tablespace altered.

SQL>
```

Step 9: Copy Additional Objects to Target Database as Desired

Transported tablespaces carry database objects that directly map to segments (such as tables and indexes) and some objects that correlate to segments (such as LOBs and triggers). But many types of objects that either don't correlate at all to segments or have a looser affiliation are not transported with the tablespace set. For example, stored procedures, packages, and synonyms are not transported to the target database by the transportable tablespace feature.

You will need to assess which additional objects are needed in the target database and transport them manually. You can extract the data definition language (DDL) for these objects from the source database and apply it to the target database using the `dbms_metadata` package, a third party tool like TOAD, or running original export against the source database with the `rows=n` parameter so that DDL is extracted but not table data. Whatever method you use, it should be relatively fast because all you need to copy to the target database are the object definitions. The actual data has all been transported already.

Comparing Cross-platform Tablespace Transport to the Export/Import Method

Before the cross-platform tablespace transport feature was introduced in Oracle 10g, the export/import method was the most common way to move data between Oracle databases on different platforms. For the sake of comparing new with old, we repeated the case study exercise using the export/import method. A summary of findings is as follows:

| | Export/Import | Tablespace Transport |
|---|---------------|----------------------|
| Export time | 37 min | 2 min |
| File transfer time | 8 min | 13 min |
| File conversion time | n/a | 14 min |
| Import time | 42 min | 2 min |
| Approximate total time | 87 min | 31 min |
| Export file size | 4100 Mb | 640 Kb |
| Target database extra TEMP tablespace requirement | 1200 Mb | n/a |

For our project, the cross-platform transportable tablespace method moved the data in less than 40% of the time required by the export/import method. If we had shrunk the data files on the source database before transport in order to eliminate all unused space, the transportable tablespace method would have been even faster. A faster network or a more sophisticated storage system that allowed mirror splitting would have reduced file transfer time and widened the gap between the two methods even further.

Moving Oracle Databases Across Platforms without Export/Import

The transportable tablespace method was significantly faster than export/import for our project involving just 4100 Mb of table data. As the volume of data transported grows, the gap between the two methods will grow even wider. Also, consider that when transporting tablespaces across platforms that do not require data conversion, the transportable tablespace method becomes much faster because the slowest step of the process is eliminated.

Limitations and Things to Keep in Mind

Like any Oracle feature, the cross-platform transportable tablespace feature does have its limitations. It will not always be the best approach for every project. There are many situations where it cannot be used, and perhaps some where it would not make sense.

Restrictions

Cross-platform support for transportable tablespaces is only offered for certain platforms. If the platform of your source or target database is not supported, then this method for data transport is not available to you. Note that when we use the term “platform” here, we are referring to the platform as the Oracle database sees it. Two databases could be running on identical hardware, yet Oracle may see them as different platforms because their operating systems differ. As we saw in step 1 of the case study, the platform of an Oracle database can be found by querying `v$tablespace` and the list of supported platforms for cross-platform tablespace transport can be found in `v$transportable_platform`.

You can only transport tablespaces from an Enterprise Edition database. Although the `dbms_tts` package and `transport_set_violations` view are present in Standard Edition databases, attempting to export the metadata from a Standard Edition database will fail:

```
EXP-00017: feature "Export transportable tablespaces" is needed, but
not present in database
ORA-00439: feature not enabled: Export transportable tablespaces
EXP-00000: Export terminated unsuccessfully
```

This behavior is documented in Metalink document 152130.1. According to Metalink document 271886.1 you are permitted to plug a transported tablespace into a Standard Edition database. This means that transportable tablespaces can be used to distribute data from Enterprise Edition databases to Standard Edition databases but not in the reverse direction.

In order to transport tablespaces across platforms, both the source and target database must be running Oracle 10g or later and have the compatible instance parameter set to 10.0 or higher. Although the transportable tablespace feature dates back to Oracle 8i and you are allowed to transport a tablespace into a database running a later version of Oracle software, you won't be able to transport a tablespace across platforms unless it came from an Oracle 10g or later database with the compatible instance parameter set to 10.0 or higher.

Along those lines, if a tablespace from a pre-Oracle 10g database was transported into an Oracle 10g database and now you want to transport this tablespace to a database on a different platform, the

Moving Oracle Databases Across Platforms without Export/Import

tablespace must first be put into read-write mode on the source Oracle 10g database at least once before attempting to transport it across platforms. You can put the tablespace into read-write mode and immediately switch it back to read-only mode if necessary. This momentary read-write setting allows Oracle to rewrite the data file headers with information that is necessary for cross-platform transport.

You cannot plug a tablespace into a database that is not configured to support the tablespace block size. If you use multiple block sizes in your environment, you can check the block size of the tablespaces you are planning to transport by querying the `dba_tablespaces` view on the source database. If this block size matches the `db_block_size` instance parameter setting on the target database, then the transported tablespace matches the default block size of the target database and there will be no block size issue. If the two figures do not match, however, then the target database needs a buffer cache that matches the transported tablespace block size. Check the appropriate `db_Nk_cache_size` instance parameter on the target database. If it is set to zero, then there is no such buffer cache and you can create one with a statement like:

```
SQL> ALTER SYSTEM SET db_32k_cache_size = 160m;

System altered.

SQL>
```

You cannot plug a tablespace into a database that uses a different character set than the database where the tablespace originated. If you try to, you will get the following error while importing the metadata:

```
IMP-00003: ORACLE error 29345 encountered
ORA-29345: cannot plug a tablespace into a database using an
incompatible character set
```

It is interesting to note that in Oracle 8i and 9i the text of the ORA-29345 error reads “cannot plug a tablespace into a database using a different character set.” In Oracle 10g the word “different” has been changed to “incompatible.” This suggests that plans might be in the works to loosen this restriction in a future Oracle release. However, Metalink document 291024.1 entitled, “Compatibility and New Features when Transporting Tablespaces with Export and Import” (which was updated in August, 2006 and specifically discusses new Oracle 10g functionality) indicates that character sets must be the same. So we can speculate about what enhancements the future may hold, but for now we cannot transport tablespaces between databases that have different character sets.

Things to Keep in Mind

The transportable tablespace feature focuses on moving data—not schema definitions—between databases. While the export/import method will copy all types of schema objects from one database to another, the transportable tablespace feature will not. It is important to understand which object types must be transported manually and to devise a method for doing so. Table 19-3 in the Database Utilities manual for Oracle 10g release 2 provides a comprehensive list of which objects are included in transportable tablespaces when original export and import are used.

Moving Oracle Databases Across Platforms without Export/Import

Continuing on the theme of objects getting lost, it is important to understand the nuances of self-containment. In step 2 of the case study, we discussed what it means for a set of tablespaces in a database to be self-contained. A set of tablespaces can only be transported if they contain no objects dependent on objects residing outside of the set. However, the default behavior of original export and data pump are to allow a set of tablespaces to be transported if there are objects residing outside the tablespace set that are dependent on objects within the set. This can lead to latent problems such as a table being transported to another database with some—but not all—of its indexes. As we discussed in the case study, you may wish to call the `dbms_tts.transport_set_check` procedure with the `full_check` parameter set to `TRUE` in order to make sure you are aware of all dependencies that exist between objects inside and outside the set of tablespaces to be transported.

The most time-consuming steps of transporting tablespaces across platforms are the file conversion (if necessary) and file transfer. The amount of time required by each of these steps is roughly proportional to file size. RMAN is smart enough not to back up blocks in a data file that have never been used, but it does not appear smart enough to skip unused blocks when converting files for transport to another platform. Therefore, empty space in transported tablespaces will slow down the process. For this reason, you may want to resize data files on the source database just before making tablespaces read-only in order to minimize the time required to convert and transfer the data files to the target database server.

Transporting tablespaces across platforms worked well for us in our case study. But it is important to remember that cross-platform support for transportable tablespaces is a new feature in the Oracle continuum (first introduced in Oracle 10g release 1). The feature, therefore, is not yet as mature as many other Oracle features. We should expect enhancements and changes to the feature in upcoming releases and patch sets, and should not be surprised by various rough edges or “opportunities for improvement.” As a trivial example, consider what happens when you forget to specify the `datafiles` parameter while importing metadata with original import:

```
IMP-00017: following statement failed with ORACLE error 29348:
"BEGIN
sys.dbms_plugts.beginImpTablespace('TAB1',6,'SYS',3,0,8192,1,115618"
"4,1,2147483645,8,128,8,0,1,0,8,3437408061,1,33,508403,NULL,0,0,NULL,NU
LL); "
"END;"
IMP-00003: ORACLE error29348 encountered
ORA-29348: You must specify the datafiles to be plugged in
ORA-06512: at "SYS.DBMS_PLUGTS", line 1801
ORA-06512: at line 1
```

Wouldn't it have been nicer and simpler if the import utility had checked the specified parameters for completeness before connecting to the database and trying to plug in an unspecified data file?

Conclusion

Oracle introduced the transportable tablespace feature back in Oracle 8i. Beginning in Oracle 10g release 1 this feature has been enhanced to allow transporting tablespaces across platforms in certain cases. Cross-platform support for transportable tablespaces makes it faster, easier, and less resource-intensive to move large volumes of data between Oracle databases that run on different platforms. Given the heterogeneous nature of most data centers today, this feature should prove extremely valuable for some users.

The ability to transport tablespaces across platforms can dramatically reduce the down time necessary to move a production database from one platform to another—not an uncommon activity these days. Cross-platform tablespace transport can also be useful on a continuing basis for sharing data between Oracle databases running on varied platforms.

In our case study, we found that we could transport tablespaces between Oracle 10g release 2 databases with the 10.2.0.2.0 patch set running on Solaris SPARC and Linux Intel processor architectures. The procedure went smoothly and for the most part as the documentation said it should. Were we lucky? Or is this feature as solid as we would like it to be? It is impossible to know for sure. Before attempting to transport tablespaces across platforms in your production environment, test the procedure fully in your test environment. Also, be sure to check the latest documentation and postings on Metalink³ to see what has changed since this paper was written in December, 2006.



³ <https://metalink.oracle.com/CSP/ui/index.html>