



CHAPTER 1

Oracle9i Database Administration and Management Features

- The compatibility parameter
- Oracle-managed datafiles
- Oracle9i shared memory areas
- The default temporary tablespace
- UNDO tablespaces in Oracle9i
- Resumable space management
- Persistent initialization parameters
- Supported platforms for Oracle9i



Welcome to the first chapter of *Oracle9i New Features*. This chapter is the beginning of your journey into Oracle9i's new and enhanced features. Of course, DBAs want features and enhancements to make the management of their Oracle databases easier, quicker, and with as little impact on their users as possible. Oracle9i responds to this need with many new management features, and it is these features that we will address here.

The Compatibility Parameter

As we begin to review many of the new, changed, and enhanced features of Oracle9i, a word about the *compatibility* parameter: this parameter, which is located in the database parameter file (*init.ora*), controls which database features you can, and cannot, use. It might well be a good idea when migrating to Oracle9i to leave *compatibility* set to the version of the database you are migrating from until you are comfortable with Oracle9i. This is because you will not be able to use the new features of Oracle9i with the compatibility parameter set to a value that is not 9i. This makes it easier if you find you need to roll back your database upgrade or migration, since you will not have taken advantage of some Oracle9i feature that will have to be backed out before you can roll the database back to the previous version.

Also, certain database default actions (such as creating a tablespace) are different depending on the value of *compatibility*. Once you have migrated or upgraded to Oracle9i and you are comfortable with the new database, set the *compatible* parameter to 9.0.0.0, and you can start using some of the new Oracle9i features that you will find inside this book.

Migration/Upgrade Notes

Generally, all operating systems support a direct migration from Oracle 7.2 or later to Oracle9i using the **mig** utility or the database migration assistant. Many platforms even support migration from Oracle 7.1. If you are running Oracle8i, then you will need to upgrade the database following your platform-specific instructions. This generally involves starting the 8i database under the 9i RDBMS software and running an upgrade script. Please refer to your platform-specific documentation for exact instructions on migrating or upgrading to Oracle9i from your current Oracle version.

Oracle-Managed Datafiles (OMFs)

The first new Oracle9i feature you are going to learn about in this chapter are *Oracle-Managed Datafiles* (OMFs). OMFs give Oracle the ability to manage Oracle database files for you. OMFs are part of the Oracle9i move to make the Oracle database easier to manage. Previous to Oracle9i, when you dropped a tablespace,

you would also have to remove the physical datafile associated with that tablespace. With Oracle9i, you can leave physical file management to the database itself, using OMFs. In this section, you will learn about the types of datafiles that are managed by this feature, and some of the benefits and restrictions of OMFs. Then you will learn how to configure your database to take advantage of OMFs, and you will find some examples of using OMFs.

OMF Uses, Rules, and Restrictions

In this section, you will learn about the uses, rules, and restrictions involved in using OMFs. First, we'll look at the concept of using OMFs, when they should be used, and when they should not. Then we'll move on to OMF management issues.

Introducing OMF

You can use OMF when creating database datafiles, tempfiles, online redo logfiles, and database control files. To use OMF, you must first configure the database to use OMF (see the "Configuring the Database to Use OMF" section). Once the database is configured for OMF, Oracle will create the datafiles required during the execution of a DDL statement such as **create tablespace**—if you do not specifically define the datafiles associated with that statement. OMF can be associated with tablespaces, temporary tablespace, redo logs, and control files in Oracle9i. Let's look at some specifics of OMF with regards to the creation of tablespaces, redo logs, and control files.

Tablespace OMF You can create any tablespace using OMF, even the SYSTEM tablespace. To configure Oracle for this operation, you need to set the *db_create_file_dest* parameter in the database parameter file (see "Configuring the Database to Use OMF," later in the chapter, for more on configuring OMF). For example, when you create a tablespace by issuing the **create tablespace** or **create temporary tablespace** commands without any datafile names, Oracle will create the needed datafile for that tablespace. Also, if you issue the **create database** command and do not provide a datafile name for the SYSTEM tablespace, then an OMF datafile will be created. Also, if you define a DEFULAT tablespace or an UNDO tablespace in the **create database** command, then an OMF will be created for each of those tablespace types. The default size for any OMF is 100M, and the datafile(s) are set to **autoextend** with an unlimited maximum extent.

If you wish to define a file size other than 100M for a datafile, include the **datafile** keyword, and then include the **size** parameter (without a filename), and the datafile will be created at the requested size. You can also include **autoextend off** to disable the setting of **autoextend** on the OMF when it is created. An example of this is shown here:

```
 CREATE TABLESPACE new_tbs DATAFILE SIZE 500M AUTOEXTEND OFF;
```

4 Oracle9i New Features

This next example is of the creation of a tablespace using two OMFs:

```
CREATE TABLESPACE new_tbs DATAFILE SIZE 500M, SIZE 500M AUTOEXTEND OFF;
```

You can change the datafile size (via the **alter database datafile resize** command) or change the datafile **autoextend** parameters without affecting the ability of the Oracle database to manage the datafile.

As datafiles associated with tablespaces fill, they will extend—as long as the ability to **autoextend** has not been changed by the DBA. If desired, rather than extending the existing OMF, the DBA can opt to create an additional datafile for the tablespace by issuing an **alter tablespace add datafile** command. If the DBA does not include the datafile name, then the new datafile added will be an OMF. You can mix and match OMF and non-OMF datafiles in the same tablespace if you desire. Oracle will not remove any non-OMF datafiles unless you use the new **including contents and datafiles** keyword of the **drop tablespace** command.

When you drop a tablespace that contains OMF, Oracle will remove the OMFs associated with that tablespace from the operating system. For example, issuing a **drop tablespace** command will cause Oracle to remove the datafiles associated with that tablespace—as long as they are Oracle managed. Of course, if you have defined the names and locations of the datafiles, then Oracle will not remove those datafiles. You will be responsible for that administrative activity yourself.

Another interesting bit of functionality is that you can mix and match OMF with manually defined ones. For example, the following command is perfectly legal:

```
CREATE TABLESPACE new_tbs DATAFILE SIZE 500M,  
'd:\oracle\oradata\mydb\mydb_new_tbs_02.dbf' SIZE 500M AUTOEXTEND OFF;
```

In this event, Oracle will create both the OMF and the manually defined datafile. If you drop the tablespace, the default action will be that Oracle will remove only the OMF, and the DBA will need to manually remove all datafiles that are not Oracle managed. This feature can be extended to existing tablespaces that use manually created datafiles. For examples, adding additional OMFs to an existing tablespace can expand space allocated to the tablespace created originally in Oracle8i with manually created datafiles. You will find several examples of Oracle-managed file operations on redo logfiles later in this section.

Redo Log OMF If you decide to use Oracle-managed redo logfiles, you can create as many redo log groups as you need, bounded of course by the **maxlogfiles** clause setting you used in the **create database** command. You can multiplex each of those groups with up to five additional OMF members (again bounded by the **maxlogmembers** setting when the database is created). The different redo log group members are created in different locations, as defined by multiple parameters such as *db_create_online_log_dest_n* (see “Configuring the Database to Use OMF,” later in the chapter, for a list of these parameters).

You can initially create a database with the **create database** statement, using OMF redo logs. Simply omit the name of the database datafiles, as you can see in the example in “Creating a Database Using OMF,” later in the chapter.

Depending on the operating system, if none of the *db_create_online_log_dest_n* parameters are set, then one member for each redo log group will get created in the location pointed to by the *db_create_file_dest* parameter. If neither parameter is set, then Oracle will return an error when you issue the **create database** statement.

If you issue either the **alter database drop logfile group** or **alter database drop logfile member**, Oracle will remove the associated logfile group or member—if they were created as OMFs. By default, Oracle-managed redo logs are 100M in size. You will find several examples of Oracle-managed file operations on redo logfiles later in this section.

Control File OMF If the *CONTROL_FILES* parameter is not listed in the database parameter file when you create the database, and if the database parameter *db_create_online_log_dest_n* is configured, Oracle will create the control files for you as OMF in the directories defined. As with the redo logfiles, you can configure the database so up to five copies of the control files will be created (see the upcoming “Configuring the Database to Use OMF” section). If the *db_create_file_dest* parameter is set, but the *db_create_online_log_dest_n* is not, then a single control file is created in the *db_create_file_dest*. If the *db_create_online_log_dest_n* parameters are set, then the control files will be written there.

Depending on the operating system, if neither the *control_files*, *db_create_online_log_dest_n* nor the *db_create_file_dest* parameters are set, Oracle might choose to do a couple of things. In some cases, Oracle might create an OMF control file in a default directory that is OS specific. In this case, the control files will not be Oracle-managed control files. If you wish the control files to be Oracle managed, you will need to make sure that the OMF parameters (either *db_create_file_dest* or *db_create_online_log_dest_n*) are correctly set. On some platforms, Oracle will simply signal an error if the location of the control file is not defined by the presence of the *control_files* parameter.

When to Use, and Not to Use, OMF

Oracle OMF is useful in different situations. First, it is quite useful in low-use smaller databases to reduce the administrative overhead associated with such a database. This feature reduces the overall administrative overhead required for such databases and also helps to ensure that old, unused datafiles do not reduce the overall availability of disk space. Configuring the database to use this feature does not imply that the alternative ability to define datafile names and locations is not available. In fact, you can use both features of the database if you choose.

The OMF feature can be particularly useful for development and test databases. With this feature, you can allow the developers some latitude to create and remove their own tablespaces (though there is no support at this time for forcing the use of OMF).

6 Oracle9i New Features

Another use of OMF is to simplify management of a standby database. Previously, when you added a tablespace or datafile to the primary database, human intervention was required on the standby database to perform the same operation. Now, with OMF, this is no longer the case. If the standby database is configured to use OMF, then the creation of a tablespace or addition of a datafile to the primary database will result in the automated creation of that tablespace or datafile on the standby server. No other administrative activity is required.



NOTE

If datafiles are removed from the primary database, Oracle will not automatically remove the related datafiles on the standby database.

Also, if you have a large database environment that is using large disk arrays, you might find OMF of use to you as well. In these environments, typically a small number of large file systems are created that are striped across a number of disks. The main idea is to stripe across as many disks as you can. This can cause significant performance gains.

OMF is not an appropriate choice for use with a high-volume or mission-critical database that is not using high-end striped disk arrays. For example, OMF would not be recommended on systems with many smaller file systems, or systems running RAID-5. This is because the nature of managed datafiles is such that such DBA tasks as IO distribution are not really supportive of this feature (and kind of defeat the whole purpose, in a way). Also, the managed datafile feature does not support the use of raw disk devices.

Administering OMF

When Oracle creates managed database datafiles, it follows a naming convention for these datafiles. You cannot create a new datafile using the OMF naming convention. Any attempt to do so will result in an error. The naming conventions of the database datafiles are shown in the following table. (Note that these might be different for various operating system ports. Check your operating system documentation for the file-naming convention used.)

File Type	Naming Convention	Example
Datafile	ora_{tablespace_name}_ {unique character string}.dbf	ORA_NEW_TBS_ZV3NZF00.DBF
Tempfile	ora_{tablespace_name}_ {unique character string}.tmp	ORA_TEMP_TBS_ZV3NZF01.DBF
Redo logfile	ora_{online redo log group number}_{unique character string}.log	ORA_4_ZV307100.LOG
Control file	ora_{unique character string}.ctl	ORA_4_ZV307100.CTL

NOTE

Up to eight characters of the tablespace name are used. This is why the second part of the name, the unique character string, is important, as two tablespaces might have unique names, but the first eight characters of the tablespace might be the same.

As a DBA, you can use the names of OMF in SQL statements, just as you would normal datafiles. For example, you can use the **alter database rename file** or **alter tablespace rename datafile** commands to rename an Oracle database-managed datafile, you can drop a specific Oracle-managed redo logfile with the **alter database drop logfile** command, and so on.

To rename an OMF datafile, you will first need to offline the tablespace that the OMF datafile is associated with (or offline the OMF datafile). Then, physically rename the datafile at the OS level. Once you have renamed the OMF file, you can issue the **rename** command from within the database (using the **alter database** or **alter tablespace** command) to rename the OMF within the database. Finally, online the tablespace or datafile.

If you rename the OMF datafile using a file-naming convention that does not follow the OMF naming convention, that file will no longer be an OMF. Finally, you cannot rename any existing non-OMF Oracle datafile to a filename that has “ORA_” at the beginning. This will cause an error, as ORA_ prefixes are reserved for OMF.

Here is an example of renaming an existing OMF datafile:

```
alter session set db_create_file_dest = '/home1/teach3';
create tablespace sdgtest4 datafile size 2m;
select file_name,tablespace_name
from dba_data_files where tablespace_name = 'SDGTEST4';
FILE_NAME                                     TABLESPACE_NAME
-----
/home1/teach3/ora_sdgtest4_xx5vcmqf.dbf      SDGTEST4

ALTER TABLESPACE sdgtest4 OFFLINE;

HOST ls
ora_sdgtest4_xx5vcmqf.dbf
host mv ora_sdgtest4_xx5vcmqf.dbf ora_sdgtest4_xx5vsdg4.dbf
alter tablespace sdgtest4 rename datafile
'/home1/teach3/ora_sdgtest4_xx5vcmqf.dbf' to
'/home1/teach3/ora_sdgtest4_xx5vsdg4.dbf';

ALTER TABLESPACE sdgtest4 ONLINE;
```

The backup and recovery procedures for Oracle-managed database datafiles are no different than those for DBA managed datafiles. Also, the use of the Oracle

imp and **exp** utilities are not affected by the presence of OMF. The procedure for recovering from the loss of a control file when using backup control files or re-creating the control file using the results of an **alter database backup control file to trace** has not changed either.

Configuring the Database to Use OMF

To use OMF, you first must configure certain database parameters. These parameters define the locations that the different OMF should be created in. The parameters associated with OMF are seen in Table 1-1 (you can find examples in the “Examples of Using OMF” section, next).

Note that each of the parameters described in Table 1-1 can be dynamically altered via the **alter system** or the **alter session** command, such as

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST='d:\oracle\data\my_datafiles' ;
```

Parameter Name	Default	Purpose
<i>db_create_file_dest</i>	None	This defines the file system where OMF and tempfiles are to be located. This location is also used for Oracle-managed control files and redo logs if the <code>DB_CREATE_ONLINE_LOG_DEST_n</code> parameter is not configured.
<i>db_create_online_log_dest_n</i>	None	This parameter defines the file system location where Oracle-managed online redo logs are to be created. The <i>n</i> value is replaced by a number, 1–5. This allows for up to 5 multiplexed copies of each redo log group member, and up to 5 copies of the control files to be created.

TABLE 1-1. Database Parameters

Changing the location to create files does not affect Oracle's ability to manage datafiles already created in other directories.


NOTE

Even if you have not configured `db_create_file_dest` or `db_create_online_log_dest_n`, you can still configure them dynamically and take advantage of OMF without having to shut down and restart the database.

Examples of Using OMF


In this section, you will learn about the various database operations that can use OMF. First, you will configure a parameter file so you can use OMF. You will then learn about creating a database using OMF for the SYSTEM tablespace, UNDO tablespace, default tablespace, redo logs, and control files. We will then move on to examine the impacts of several different types of operations involving OMF. This includes dropping tablespaces and adding and removing online redo logs to the database. In this example, we will be using my Oracle9i database, called mydb.


NOTE

These examples are not intended to be a "how-to" on the overall process of the creation of a database. There are several steps that occur before and after the steps being demonstrated. It is assumed that you are already familiar with the procedure to create a database.

Configuring the Database to Use OMF

The first step in using OMF is to configure the `init.ora` database parameter file to support the use of this feature. Here is an example of the `init.ora` for the mydb (we have left out many of the settings that don't pertain to configuration of OMF).



```
db_name=mydb
undo_management=auto
undo_tablespace=myundotbs
DB_CREATE_FILE_DEST=c:\oracle\admin\mydb\data
DB_CREATE_ONLINE_LOG_DEST_1=c:\oracle\admin\mydb\redo
DB_CREATE_ONLINE_LOG_DEST_2=d:\oracle\admin\mydb\redo
```

In this example, the mydb database has been set up to use automated undo management. An UNDO tablespace called MYUNDOTBS has been defined as the `db_create_file_dest` parameter, which serves to enable the Oracle-managed datafile

feature. In this case, when Oracle creates a datafile, it puts it in the `c:\oracle\admin\mydb\data` directory.

Next, two different directory locations for redo logs and control files to be created in have been defined using the `db_create_online_log_dest_1` and `db_create_online_log_dest_2` parameters. Notice that the directories are using two different drives to protect the redo logs and control files from accidental erasure and for some IO balancing.

Creating a Database Using OMF

Now let's create our database! If we wanted to just let Oracle do all the work for us, we could actually just issue the command **create database**. In this case, if the OMFs are configured in the `init.ora`, Oracle will create everything for you. To exert a bit more control over our database creation process, however, let's use the following **create database** command to create the **mydb** database:

```
CREATE DATABASE mydb
DATAFILE SIZE 200M
LOGFILE GROUP 1 SIZE 20M, GROUP 2 SIZE 20M
DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 50M
UNDO TABLESPACE undo_ts DATAFILE SIZE 50M
MAXLOGFILES=5
MAXLOGMEMBERS=5
MAXDATAFILES=200
NOARCHIVELOG;
```

So, in our example, the database `mydb` will be created with a `SYSTEM` tablespace that is 200M in size. Based on the parameters we set in the previous section, the datafile for the `SYSTEM` tablespace will reside in the directory `c:\oracle\admin\mydb\data`. Next, we have defined a default temporary tablespace called `dflt_ts`. This tablespace is 50M in size and will also be created in `c:\oracle\admin\mydb\data`. We have also created an `UNDO` tablespace called `undo_ts` that is 50M in size. Again, this tablespace's datafile will be in `c:\oracle\admin\mydb\data`. When this database is created, the redo logfiles will be created in two locations. The first member of each group will be in the directory `c:\oracle\admin\mydb\redo`, and the second will be in the `d:\oracle\admin\mydb\redo` directory.

Management of OMF

Now that our database is created, let's look at some examples of administrative functions involving OMF. This includes adding and dropping tablespaces. Other administrative items you will learn about are the changing of the default locations for datafile creations and the process of adding and dropping redo logs from a database that is using OMF.


Adding a Tablespace Adding a tablespace is a simple operation when using OMF. Simply issue the **create tablespace** command with only the name of the tablespace, and Oracle will create the tablespace using the 100M default datafile size, as shown in this example:

```
 CREATE TABLESPACE auto_created_tbs;
```

If we wanted a larger tablespace created, we could include the DATAFILE clause and indicate what the size of the datafile should be. Also, in this next case, the datafile's ability to extend will be disabled by including the NOEXTEND clause in the statement. The resulting statement is seen in this example:

```
 CREATE TABLESPACE bigger_tbs DATAFILE 200M NOEXTEND;
```

The previous statement creates a 200M datafile, and disables the AUTOEXTEND functionality of the datafile. You can also use OMF when creating temporary tablespaces, as shown in this example:

```
 CREATE TEMPORARY TABLESPACE temp_obj_tbs DATAFILE 200M NOEXTEND;
```

Dropping a Tablespace Now that we have created tablespaces, there will be a need to drop them from time to time. In this example, let's drop the bigger_tbs we created earlier in this section. Simply use the **drop tablespace** command, and Oracle will handle the rest, as shown in the following example:

```
 DROP TABLESPACE bigger_tbs;
```

The datafile for the bigger_tbs tablespace will be dropped by Oracle automatically during the execution of the statement.

NOTE

Even if the db_create_file_dest parameter has been changed, Oracle will remove any OMF—as long as it remains in its original directory.

NOTE

*The **including contents** clause of the **drop tablespace** statement has had a new clause added to it, and datafiles. When this clause is included in the **drop tablespace** command, the tablespace will be dropped and all associated datafiles will be dropped as well. This is a new feature in Oracle9i!*

Changing the Location of Datafile Creation The **alter system** and **alter session** commands can be used to alter all the parameters associated with OMF. Thus, we can change the location that database datafiles, as well as redo logs, are created in. Here is an example of changing the location of the parameter *db_create_file_dest*, and then creating a datafile after that operation:

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST='d:\oracle\admin\mydb\data';  
CREATE TABLESPACE new_tbs DATAFILE SIZE 150m NOEXTEND;
```

In this case, any newly created datafile will be created in the d:\oracle\admin\mydb\data directory. If we dropped any tablespace that had datafiles in the old directory, those datafiles would be removed by Oracle, just as Oracle would remove the datafiles for the new_tbs tablespace just created.

Adding a Redo Log Group When we created our database, two redo log groups of 200M each were defined for our database. Let's create a third logfile group. To do this, simply issue the following command:

```
ALTER DATABASE ADD LOGFILE;
```

Alternatively, you can issue the following command:

```
ALTER DATABASE ADD LOGFILE GROUP 3 SIZE 300M;
```

Note that in this example we have indicated that the redo log group will be 300M, as opposed to the 100M default size.

Dropping a Redo Log Group Perhaps you have discovered that your existing redo log members are not large enough and you wish to re-create them so they are larger. In this case, first you need to remove one of the existing redo log groups, and then re-create it. This is a simple operation, performed with the **alter database** command:

```
ALTER DATABASE DROP LOGFILE GROUP 1;
```

Something to keep in mind is that it's not possible to add an additional log group member that is an OMF (that is, **alter database add logfile member to group 2**). You can drop an OMF redo log member, however, with the **alter database drop logfile member**. In this event, Oracle will remove the dropped redo log member.

NOTE

As a DBA, you should already be aware that if you are going to drop a logfile group, it cannot be the current logfile group.

Managing Oracle9i Shared Memory Areas

Oracle9i has made several changes to the management of shared memory areas contained with the SGA. This includes database parameter changes, the ability to dynamically change shared memory allocations, and the ability to support multiple database block sizes within the Oracle9i database. Let's look at each of these features, and how they relate to Oracle9i memory areas, in a bit more detail.

Multiple Database Block Size Support

Oracle now supports multiple database block sizes. This was done, in part, to allow transportable tablespaces to be plugged in from databases with differing database block sizes. Each Oracle9i database has a *standard block size* assigned, which is defined by the DB_BLOCK_SIZE parameter and is assigned to the database when it is created. This block size must be used for the SYSTEM, TEMPORARY, and ROLLBACK tablespaces and is the default block size used for other tablespaces. As with previous versions of Oracle, once you determine the standard block size, it is set in stone unless you re-create the database. When you upgrade from Oracle8i to Oracle9i, the standard block size for your database will be whatever the block size of the Oracle8i database was when it was created.

When creating a tablespace, you can use the BLOCKSIZE parameter of the **create tablespace** command to create that tablespace using a *nonstandard block size*. Oracle9i can support up to four nonstandard block sizes that can range anywhere from 2K to 32K in size, depending on operating-system restrictions.

```
CREATE TABLESPACE my_16k_tbs
BLOCKSIZE 16k
DATAFILE 'd:\oradata\mydb\data\my_8k_tbs.dbf' size 100m;
```

The DBA_TABLESPACES and V\$DATAFILE views have had a block-size column added to them that defines the assigned block sizes for each tablespace, as shown in this example:

```
SQL> select tablespace_name, block_size
2  FROM dba_tablespaces
3  where tablespace_name = 'MY_16K_TBS';
TABLESPACE_NAME          BLOCK_SIZE
-----
MY_16K_TBS                16384
```

NOTE

To use nonstandard block sizes, you must set up some subcaches in shared memory. This will be discussed in the following section.

New Oracle9i Memory Database Initialization Parameters

Oracle9i has introduced several new database initialization parameters that have impacts on memory allocations in the database. In this section, you will learn about the new `db_cache_size` parameter and the memory subcache configuration parameters introduced to support the new multiple database block size features of Oracle9i.

The `db_cache_size` Parameter

Oracle9i has deprecated the `db_block_buffers` parameter, which controls the size of the database buffer cache component of the SGA, in favor of a new parameter, `db_cache_size`. This parameter is defined in bytes (*K* and *M* can be used to indicate kilobytes and megabytes) and allocates memory in blocks based on the standard block size of the Oracle9i database (just as `db_block_buffers` does). Note that the `db_block_buffers` parameter is backward compatible. The new features of Oracle9i (such as being able to dynamically change the size of the database buffer cache, or the use of multiple database block sizes) are not available when using the `db_block_buffers` parameter, and you cannot define both parameters at the same time. If you try, an ORA-00381 error will be raised.



NOTE

When you migrate or upgrade your 7.x, 8.x, or 8i Oracle database to Oracle9i, you will probably want to replace the `db_block_buffers` parameter with the `db_cache_size` parameter. Do this so you can take advantage of the new Oracle9i features such as dynamically changeable SGA memory configurations.

The `db_keep_cache_size` and `db_recycle_cache_size` Parameters


As with the `db_block_buffers` parameter, the `buffer_pool_keep` and `buffer_pool_recycle` parameters have been deprecated in favor of new parameters in Oracle9i. The new parameter `db_keep_cache_size` replaces the `buffer_pool_keep` parameter. The new `db_recycle_cache_size` parameter replaces the `buffer_pool_recycle` parameter. Both old parameters are still available for backward compatibility, but the old parameters are not dynamically alterable as the new parameters are. Also, you cannot use both the old and new parameters at the same time.

**NOTE**

In Oracle9i, the `db_keep_cache_size` and `db_recycle_cache_size` memory areas are separate memory areas and are not allocated out of the default buffer pool, as was the case in Oracle8i and the `buffer_pool_keep` and `buffer_pool_recycle` parameters.

Configuring Memory Subcaches

To take advantage of Oracle9i support for multiple block sizes, you will need to allocate shared memory subcaches. Five new parameters are introduced in Oracle9i to support the subcaching feature: `db_2k_cache`, `db_4k_cache`, `db_8k_cache`, `db_16k_cache`, and `db_32k_cache`. These parameters are defined in bytes, and Oracle allows the use of *K* to denote kilobytes and *M* to indicate megabytes. Note that these allocations require additional memory, and are not taken from the memory allocated by the `db_cache_size` parameter. Additionally, you cannot use the subcache parameter that is the same for the block size of your database. Thus, if your `db_block_size` is 4K, you cannot use the `db_4k_cache` parameter. An example of setting these parameters would look like this:



```
DB_2K_CACHE_SIZE=8364032
DB_8K_CACHE_SIZE=8000K
```

In this example, three different memory caches have been allocated. First, the 2K cache is allocated about 8 megabytes, defining the cache size in bytes. Second, the 8K cache is allocated with about 8 megabytes by defining the cache size in kilobytes using the *K* indicator. Finally, we allocate 10 megabytes to the 16K cache by using the *M* identifier. Note that each memory cache is considered part of the overall size of the SGA. Also note that, in each case, Oracle might choose to round the allocations to the nearest granule size. For example, in our last case, Oracle might well round the 10M to 12M, which would be the nearest granule multiple. A *granule* is a unit of contiguous memory whose size depends on the estimated total size of the SGA. A granule is either 4 megabytes (in the case where the estimated SGA total size will be less than 128 megabytes); or if the database is to be larger than 128 megabytes, then the granule will be 16 megabytes. All growth and shrinkage of SGA structures is done based on granule boundaries.

Data Dictionary Views and Memory Subcaches

When you create a memory subcache, the individual subcaches do not appear in V\$SGA. The total combined memory allocated to all the subcaches and the default

cache will appear in V\$SGASTAT under the line for the setting of *db_block_buffers*. If you want to see the individual pools, you will need to use the *v\$buffer_pool* view, as shown in this example:

```
SQL> select id, name, block_size, current_size, buffers
2 from v$buffer_pool;
```

ID	NAME	BLOCK_SIZE	CURRENT_SIZE	BUFFERS
1	KEEP	8192	0	0
2	RECYCLE	8192	0	0
3	DEFAULT	8192	4	501
4	DEFAULT	2048	12	5676
5	DEFAULT	4096	0	0
6	DEFAULT	8192	0	0
7	DEFAULT	16384	12	759
8	DEFAULT	32768	0	0

In this case, we see that three buffer pools are currently defined. The name column contains the type of buffer pool. In the case of subcaches and the default buffer pool, the name is always default. The BLOCK_SIZE column is what really tells us what is going on. In this case, there is the default buffer pool for our block size, which is 8K. Also there are the 2K and 16K buffers established, as can be seen from the output. The CURRENT_SIZE column gives the current size of the individual buffer cache in megabytes. Note that this size might actually be larger than the size that was defined for it because Oracle will round the subcache size up to the nearest granule (see the next section for more on granules). The buffers column indicates how many BUFFERS are currently allocated to the cache.

Dynamically Changeable Shared Memory

Often, when performance-tuning a system, you will find you need to change the size of the database buffer cache or the shared pool. Previous to Oracle9i, this would require a shutdown of the database system. For 24/7 environments, shutting down a system, even for a short time, can mean lost productivity, lost business, even lost customers.

Oracle9i solves this problem by allowing the DBA to dynamically alter the size of the shared memory areas. This includes the principle areas of the SGA:

- The default database buffer cache
- The memory subcaches
- The shared pool

NOTE

The redo log buffer, large pool, and java pools cannot currently be dynamically resized.

All Oracle shared memory areas are dynamically settable via the **alter system** command. Some examples of setting these areas are shown here:

```
ALTER SYSTEM SET SHARED_POOL=50000000;
ALTER SYSTEM SET DB_CACHE_SIZE =50000000;
ALTER SYSTEM SET DB_16K_CACHE_SIZE=10M;
```

A new parameter, *sga_max_size*, is used in Oracle9i to indicate the maximum overall size of the SGA. Thus, you can dynamically expand the SGA by altering the size of any of the buffers, but you cannot alter them such that the total amount of memory allocated is greater than that set by the *sga_max_size*. The default value of *sga_max_size* is the total size of the configured SGA at instance startup. If you set *sga_max_size* to a value smaller than the amount of memory initially allocated at instance startup, then *sga_max_size* will default to the total amount of memory initially allocated. You cannot dynamically change this parameter, so take care to make sure it is set correctly if there is some chance that you will want to increase overall SGA memory use. Note that, contrary to some documentation, this parameter will cause Oracle to reserve memory of an amount of *sga_max_size* on most operating systems (Solaris 8 was the exception at the time this book was written), so be careful when setting it to avoid causing swapping or paging.

Automated PGA Memory Management

The PGA in Oracle consists of two different memory types: not tunable and tunable. Several database parameters can be used to configure the tunable area. These parameters include *sort_area_size*, *hash_area_size*, *bitmap_merge_area_size*, and *create_bitmap_area_size*. In Oracle8i, these parameters were able to be set dynamically; however, it was difficult to really tune them well. Often, however, more memory was allocated to a given session than was really needed. As a result, memory was wasted.

Now Oracle9i offers the option of automated PGA space management. Two new parameters have been introduced to allow the DBA to have the PGA dynamically configured by the RDBMS software, removing this responsibility from the DBA. The first parameter is *pga_aggregate_target*, which allows you to set a target aggregate amount of memory that becomes the target amount of PGA memory available to be allocated. This memory can be allocated in bytes, kilobytes, megabytes, or gigabytes (using the *K*, *M*, and *G* symbols to denote the allocation type). This parameter can be set only at the system level, but it is dynamic in its nature.

A second parameter, *workarea_size_policy*, which can be set at the system or session level, establishes whether a given session's PGA size should be sized automatically or via the database init.ora parameters. There are two valid values for this parameter. When set to *AUTO*, the database will size the tunable PGA memory, and the total aggregate allocated amount of PGA memory will be bounded by

pga_aggregate_target. When set to `MANUAL`, the size of the PGA memory allocation is based on the various database parameter settings. If the *pga_aggregate_target* is set, then the *workarea_size_policy* will default to `AUTO`.

In the `V$SYSSTAT` and `V$SESSTAT` views, three new statistics have been added that relate to automated PGA memory. These are

- **Work Area Executions: Optimal Size** Represents the number of work areas that had an optimal size, and no writes to disk were required.
- **Work Area Executions: One Pass Size** Represents the number of work areas that had to write to disk, but required only one pass to disk.
- **Work Area Executions: Multipasses Size** Represents the number of work areas that had to write to disk using multiple passes. High numbers of this statistic might indicate a poorly tuned PGA.

In addition, new columns have been added to `V$PROCESS` to help tune the PGA.

- **PGA_USED_MEM** This reports how much PGA memory the process uses.
- **PGA_ALLOCATED_MEM** This is the amount of PGA memory allocated to the process.
- **PGA_MAX_MEM** This is the maximum amount of PGA memory allocated by the process.

Also, three new views are available to help the DBA extract information about the PGA:

- **V\$SQL_WORKAREA** Provides information about SQL work areas.
- **V\$SQL_WORKAREA_ACTIVE** Provides information on current SQL work area allocations.
- **V\$SQL_MEMORY_USAGE** This displays current memory-use statistics.

Oracle9i Default Temporary Tablespace

Oracle9i has introduced a new feature called the *default temporary tablespace*. In the past, whenever you created a user, the user would be given the `SYSTEM` tablespace as its default temporary tablespace. Left unchanged, this could lead to

serious fragmentation and IO issues with the SYSTEM tablespace, because many users would be creating and removing temporary segments.

To deal with this problem, Oracle9i introduces the default temporary tablespace. The default temporary tablespace is either set at database creation with the **default temporary tablespace** clause of the **create database** command, or it can be set or changed after database creation with the **alter database default temporary tablespace** command. When the default temporary tablespace is changed with the **alter database default temporary tablespace** command, all users assigned to the previous default temporary tablespace will be reassigned to the newly defined temporary tablespace. Users assigned to a temporary tablespace that was not the default temporary tablespace will remain unchanged. Note that the tablespace selected to be the new default temporary tablespace must be of the same block size as the standard block size of the database. Also, any default temporary tablespace must be of type TEMPORARY.

If you wish to know what tablespace is current assigned as the default tablespace, you can use the new DATABASE_PROPERTIES view. Look in the PROPERTY_NAME column for the value DEFAULT_TEMP_TABLESPACE, and you will find the tablespace name in the associated PROPERTY_VALUE column. Here is an example of such a query:

```
SQL> column property_value format a16
SQL> select property_name, property_value from database_properties
  2  where property_name = 'DEFAULT_TEMP_TABLESPACE';
PROPERTY_NAME                PROPERTY_VALUE
-----
DEFAULT_TEMP_TABLESPACE      NEW_TEMP
```

Finally, note that Oracle will no longer allow you to assign a permanent locally managed tablespace as a user's temporary tablespace. This was allowed in Oracle8i, but the users session would get an error when it tried to create a temporary segment in the tablespace.

Automated UNDO Management in Oracle9i

One of the more maintenance-intensive architectural components of a pre-Oracle9i database was the management of rollback segments. It was often time-consuming to first decide how many rollback segments to create, how big to make them, and how many extents to make them. Then you had to monitor the use of the rollback segments to ensure that you were getting optimal use of the rollback segment configuration that you had created.

Oracle9i introduces automated UNDO management to alleviate the need to manage rollback segments. To use the Oracle9i automated UNDO management features, you must first create an UNDO tablespace. Then you must configure the Oracle9i instance to use the Oracle9i automated UNDO management feature.



NOTE


In Oracle9i, according to Oracle documentation, Oracle has actually depreciated the use of rollback segments for undo space management. This implies that Oracle intends to do away with rollback segments all together at some point. While this is not likely to occur for some time, it is probably a good idea to start learning about and using UNDO tablespaces.

Creating the UNDO Tablespace

There are two different ways of creating an UNDO tablespace. The first method is by the use of the new UNDO clause of the **create tablespace** command. The second method is through the use of the **create database** command. Let's look at each of these methods in more detail.

Using the CREATE UNDO TABLESPACE Command

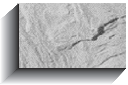
In Oracle9i, you can create an UNDO tablespace with the new UNDO clause of the **create tablespace** command, as shown in the following example:



```
CREATE UNDO TABLESPACE undo_tbs
DATAFILE '/ora100/oracle/mydb/data/mydb_undo_tbs_01.dbf' SIZE 100m
AUTOEXTEND ON;
```

In this case, we have created an UNDO tablespace called UNDO_TBS. As you can see, the **create undo tablespace** command syntax is much like the **create tablespace** command, including the **datafile** and **size** clauses. Note that when creating an UNDO tablespace, you can use only the **datafile** clause and a restricted form of the extent_management clause of the **create tablespace** command. Thus, you cannot define any default storage characteristics for an UNDO tablespace. Also note that Oracle creates an UNDO tablespace as locally managed tablespace, and that there is no option to create it as a dictionary-managed tablespace.

Once an UNDO tablespace is created, it will be brought online, along with the undo segments within it, each time the database is started. This can be seen in the messages that will appear in the alert log each time that you start the database.

**NOTE**

At the very least, you will still need to have the SYSTEM rollback segment.

Creating an UNDO Tablespace with the CREATE DATABASE Command

You can opt to create an UNDO tablespace when you initially create a database. Oracle has modified the **create database** command to support the definition of UNDO tablespaces during the database creation process through the use of the **undo tablespace** clause, as shown in the following example:

```
CREATE DATABASE mydb
CONTROLFILE REUSE
LOGFILE GROUP 1 ( 'd:\oradata\mydb\redo\mydb_redo_01a.log' ,
                 ' e:\oradata\mydb\redo\mydb_redo_01b.log ' ) SIZE 50K,
            GROUP 2 ( 'd:\oradata\mydb\redo\mydb_redo_02a.log' ,
                     ' e:\oradata\mydb\redo\mydb_redo_02b.log ' ) SIZE 50K
MAXINSTANCES 1 MAXLOGFILES 5  MAXLOGHISTORY 100 MAXDATAFILES 100
ARCHIVELOG
DATAFILE 'e:\oradata\mydb\mydb_system_01.dbf'
SIZE 100M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED,
DEFAULT TEMPORARY TABLESPACE temp_ts
TEMPFILE 'e:\oradata\mydb\mydb_temp_ts_01.dbf' SIZE 20m
UNDO TABLESPACE undo_ts DATAFILE 'e:\oradata\mydb\mydb_undo_ts_01.dbf'
SIZE 50M AUTOEXTEND OFF;
```

Note that we have used the UNDO TABLESPACE clause of the CREATE DATABASE command to create an UNDO tablespace called undo_ts. Further, we used the DATAFILE clause to define the name and location of the datafile associated with the UNDO tablespace, and we also included the SIZE and AUTOEXTEND clauses.

When issuing a **create database** command, there are several rules that you should consider that relate to UNDO tablespaces. The rules will differ depending on how the database is configured (see the next section, “Configuring the Instance for Automated UNDO Management”). If the database instance is not configured for automated UNDO management, and you omit the UNDO TABLESPACE clause, then the CREATE DATABASE statement will work as it always has, with no UNDO tablespace being created.

If the instance is configured for automated UNDO management, however, then the default behavior of the **create database** statement changes. If you do not include the **undo tablespace** clause, then Oracle will create an UNDO tablespace for you by default. This tablespace will be called SYS_UNDOTBS. This tablespace will be created using a default size of 100M for the database datafile.

Dropping an UNDO Tablespace

To drop an UNDO tablespace, simply issue a **drop tablespace** command. Oracle will drop the tablespace. If that UNDO tablespace is the active UNDO tablespace, then Oracle will generate an error.

Configuring the Instance for Automated UNDO Management

To take advantage of Oracle9i automated UNDO management features, you must configure the database. Configuration of the database for automated UNDO management is done through changes to the databases parameter file (init.ora). The parameters in Table 1-2 have been added to Oracle9i to support automated UNDO management.

Let's look at a couple of notable aspects of these parameters. Some SQL commands such as **set transaction use rollback segment** will, by default, return an ORA-30019 error to the session issuing the SQL statement. This is because these commands are not compatible with automated UNDO management. Because such an error might cause problems with existing scripts, you can set the *undo_suppress_errors* parameter to avoid getting the ORA-30019 error message.

If you have managed Oracle databases before, no doubt you are familiar with the "snapshot to old" error messages. These messages appear for several reasons, but principally they appear because a read-consistent image of the data that a given session needs is no longer in the rollback segments of the database. The *undo_retention* parameter is used with automated UNDO management to provide a guide to Oracle on how long it should retain UNDO after the transaction that has generated it has committed. By default, *undo_retention* is set to 900 seconds. This means that Oracle will try not to reuse generated UNDO space for 900 seconds after the transaction committing it has been committed.

The keyword here is *try*, because if Oracle runs out of available UNDO space, it will begin to use space that was otherwise protected by the *undo_retention* parameter. The *undo_retention* parameter can be modified dynamically with the **alter system** command. This is handy if you are finding that long-running transactions are getting "snapshot to old" errors from Oracle, though you might also need to add space to the UNDO tablespace. The *undo_retention* parameter also has some significant impact on another new Oracle9i feature, flashback queries, which we will cover in Chapter 3.

Finally, note that the *undo_tablespace* parameter is dynamic. This implies that you can have multiple UNDO tablespaces, however, you can have only one active UNDO tablespace in use at any given time.

Parameter Name	Default Value	Valid Values	Dynamic?	Description
<i>undo_</i> <i>Management</i>	MANUAL	AUTO, MANUAL	No	Determines whether automated UNDO management is enabled in the database. AUTO enables automated UNDO management and MANUAL disables the feature.
<i>undo_</i> <i>tablespace</i>	The first available UNDO tablespace, SYS_UNDOTBS, uses the SYSTEM rollback segment if no UNDO tablespace is available	Valid UNDO tablespace name. Multiple UNDO tablespaces are not supported, though Oracle does not generate an error.	Immediate for system	Defines one or more UNDO tablespaces that should be used by Oracle for automated UNDO management. If this parameter is set, and <i>undo_management</i> is set to AUTO when issuing a create database command, then you must include all tablespaces listed in this parameter in the create database statement or the statement will fail. You can list multiple tablespaces here but only the last one listed will be used since Oracle allows only one.
<i>undo_</i> <i>suppress_errors</i>	TRUE	TRUE, FALSE	Immediate for system, session allowed	Allows you to control the displaying of error messages that result from certain SQL commands when the database is in automated UNDO management mode.

TABLE I-2. *Oracle9i UNDO Management Parameters*

Parameter Name	Default Value	Valid Values	Dynamic?	Description
<i>undo_</i> <i>retention</i>	900 Seconds	0 to the maximum value allowed by 32 bits	Immediate for system	This parameter defines the minimum amount of time that Oracle will retain UNDO after it has been generated and after the generating transaction has been completed. This parameter can be modified dynamically using the alter system command. Note that Oracle will make a best effort to retain UNDO for the requested amount of time, but there is no guarantee.

TABLE I-2. *Oracle9i UNDO Management Parameters (continued)*

The Data Dictionary and Automated UNDO Management

New data dictionary views, V\$UNDOSTAT and DBA_UNDO_EXTENTS, have been created that are associated with automated UNDO management. Also, the V\$ROLLSTAT and V\$ROLLNAME views can be used to monitor overall performance of UNDO tablespaces. Just as with Oracle8i, though, there is really little you can do to tune UNDO tablespaces. Let's take a look at the V\$UNDOSTAT and DBA_UNDO_EXTENTS views in a bit more detail.

V\$UNDOSTAT

The V\$UNDOSTAT data-dictionary view provides system-generated statistics, collected every 10 minutes for the last 24 hours. This view can be used to monitor and tune UNDO space. Use this view to determine whether you have allocated sufficient space to the UNDO tablespaces for the current workload. In particular, the UNDOBLKS column is useful in determining if the tablespace is large enough. This column indicates the total number of undo blocks that were used during the statistics collection period. Thus, if the number of undo blocks consumed during the collection period is significantly larger than the size of the UNDO tablespace, you might well consider increasing the size of the UNDO tablespace for performance reasons.

Also watch the UNXPSTEALCNT column, as high numbers in this column indicate that unexpired blocks (as determined by the *undo_retention* parameter) are being expired prematurely and the space is being taken for use by transactions because available UNDO space was not available for those transactions. This is particularly important for new databases as more and more people begin to use the system, generating more undo, which can lead to “snapshot to old” error messages. Finally, the SSOLDERRCNT and NOSPSPACEERRCNT columns keep track of the number of Oracle errors generated during the snapshot. If these columns are non-zero, consider increasing the size of your UNDO tablespace.

DBA_UNDO_EXTENTS

This view provides information on each extent in the UNDO tablespaces, including the commit time for each transaction. It is also with this view that you can determine which tablespaces are defined as UNDO tablespaces.

Resumable Space Management

Ever run a large data load, just to have the database run out of space on you and roll back the entire thing? Perhaps you ran a large **select** statement that ran for six hours before it ran out of temporary tablespace space during its final large sort. With Oracle9i, rather than fling the box out of the window in anger, you can take advantage of the new Oracle9i resumable space management feature. In this section, first we’ll look at when resumable space management can be used, and then how to enable and disable this feature. Finally, we’ll review some of the various administration issues around resumable space management and provide some examples of resumable space management in action.

Resumable Space Management Features and Limitations

Resumable space management can be used to manage the impacts a space failure of a long-running transaction may have on a session. Some of the space failures include:

- **Running out of tablespace space** This includes errors such as ORA-1650 (Unable to extend rollback segment), ORA-1653 (Unable to extend table), and ORA-1654 (Unable to extent index), in which Oracle is not able to allocate another extent due to lack of available space. Adding space to the given tablespace generally solves this error condition (though coalescing space might be an alternative). If this error involves a temporary tablespace, other user sessions can result in the release of temporary segments in the tablespace, thus freeing space for the suspended session.

- **Maximum number of extents reached** This includes errors such as ORA-1628 (Max # of extents reached for rollback segment), and ORA-1631 ORA-1654 (Max # of extents reached in Table or Index). In this case, the object has a MAX_EXTENTS value assigned to it, which has been reached. The solution to this problem is to increase the MAX_EXTENTS setting for the object in question.
- **Attempt to exceed a tablespace quota** This includes error ORA-1536 (space quota exceeded for tablespace). The solution to correcting this problem is to increase the user quota for the tablespace in question.

When one of these conditions is reached, and if resumable space management has been enabled for the session, then the resumable statement will be suspended. At the time the statement is suspended, an error will be raised in the alert log. In addition, the user session running the query will become suspended, until either the time-out period passes, or the error condition is resolved. Also, Oracle has provided an **after suspend** system trigger event that can be used to automate a response to a SUSPEND condition.

Once a statement is suspended, it will wait for a defined period of time (two hours is the default). After that period of time elapses, the error will be raised, and the statement rolled back. During the period of the statement suspension, if the condition that caused the statement to be suspended is corrected, then the statement will automatically resume execution. Thus, for example, if space in the temporary tablespace were released by another user session, the query that was suspended because it ran out of temporary tablespace space would resume automatically without user intervention.

Suspended operations can be monitored through the use of the DBA_RESUMABLE and USER_RESUMABLE views (see “Data Dictionary Views Associated with Resumable Space Management,” later in the chapter). Also, Oracle provides a package called DBMS_RESUMABLE that allows you to manage the Resumable Space Management features of the database, which we will discuss later in this section.

Candidate Database Operations for Resumable Space Management

There are many different kinds of operations that can take advantage of resumable space management. The following table lists these operations:

Operation Type	Comment
SELECT queries	If the query runs out of temporary sort space, then it can be suspended and resumed.
INSERT, UPDATE, DELETE operations	These operations can be suspended if they raise an out-of-space exception.

Operation Type	Comment
INSERT AS ... SELECT operation	This operation can be suspended if it runs into an out-of-space condition.
Import/export operations	Resumable space management is supported by the IMP and EXP facilities of Oracle.
SQL*Loader	Supports resumable space management operations
CREATE TABLE ... AS SELECT	Supports resumable space management
CREATE INDEX	Supports resumable space management
ALTER INDEX ... REBUILD	Supports resumable space management
ALTER TABLE ... MOVE PARTITION	Supports resumable space management
ALTER TABLE ... SPLIT PARTITION	Supports resumable space management
ALTER INDEX ... REBUILD PARTITION	Supports resumable space management
ALTER INDEX ... SPLIT PARTITION	Supports resumable space management
CREATE MATERIALIZED VIEW	Supports resumable space management
CREATE MATERIALIZED VIEW LOG	Supports resumable space management

Resumable Space Management Administration

In this section, you will learn about several administrative issues in regard to Oracle resumable space management features. First on the hit parade is the DBMS_RESUMABLE package, followed by a look at some restrictions that exist for this feature in regard to dictionary-managed tablespaces. Following that will be a look at some of the data dictionary views associated with resumable space management.

Controlling Resumable Space Management Features

Resumable space management is controlled on a session-by-session level, and is disabled by default. Any user who wants to enable resumable space management

must first be granted the *resumable* system privilege. Having been granted that privilege, the user will enable resumable space management features by issuing the command **alter session enable resumable**. Likewise, to disable resumable space management, issue the command **alter session disable resumable**. If you wish to cause specific users to enable resumable space management, then create a login trigger to alter the users' sessions when they log in to the database.

Any session that is suspended during a resumable space operation is suspended for a specific period of time, after which it will error out and abort completely. By default, when you enable resumable space management, this time-out is 7,200 seconds, or two hours. You can modify this default value by including the *timeout* parameter to the **alter session enable resumable** command, as shown in the following example:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 14400;
```

In this example, our time-out is set to 14,400 seconds, or four hours. Keep in mind that the longer you make the time-out interval, the more likely that suspended sessions will occur and pile up. This can cause resource contentions on your database to appear, as suspended sessions do consume a certain amount of overhead. If you should wish to change the time-out, simply issue another **alter session enable resumable** command with a different time-out value. Alternatively, you can use the **dbms_resumable.set_timeout** procedure (we will discuss this package in the next section).

With each execution of **alter session enable resumable**, you can also specify a name to be associated with the set of resumable sessions. This makes it easier to identify sessions in the data dictionary views for resumable space management such as `DBA_RESUMABLE`. An example of enabling resumable space management and defining a name for the sessions is shown here:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 14400 NAME 'ROBERT';
```

Note that if you do not define a name, a system default name will be used.

The DBMS_RESUMABLE Package

The principle method of managing the Oracle resumable space management features is the use of the `DBMS_RESUMABLE` package. This package contains five procedures that can be used in concert with resumable space management. These subprograms are **abort**, **get_session_timeout**, **set_session_timeout**, **get_timeout**, and **set_timeout**. We will look at each of the subprograms in a bit more detail in the following sections.

DBMS_RESUMABLE.ABORT The purpose of the **abort** procedure is to cancel all suspended statements for a given session. When calling **abort**, for the command to succeed, you must be the owner of the session, have **alter system**

privilege, or have DBA privileges. Here is the syntax for the **dbms_resumable.abort** procedure:

```
PROCEDURE DBMS_RESUMABLE.ABORT
(session_id IN NUMBER);
```

DBMS_RESUMABLE.GET_SESSION_TIMEOUT The purpose of the **get_session_timeout** function is to return the current value of a given session's resumable space management time-out setting in seconds in the form of a NUMBER type. If the session is not present, a -1 error is returned. The syntax for the **get_session_timeout** function is shown here:

```
FUNCTION DBMS_RESUMABLE.GET_SESSION_TIMEOUT
(session_id IN NUMBER)
RETURN NUMBER;
```

DBMS_RESUMABLE.SET_SESSION_TIMEOUT The purpose of the **set_session_timeout** procedure is to set the current value of a given session's resumable space management time-out setting in seconds. This change is immediate in its effect, and no error is returned if the session does not exist. The syntax for the **set_session_timeout** function is shown here:

```
PROCEDURE DBMS_RESUMABLE.SET_SESSION_TIMEOUT
(session_id IN NUMBER,
timeout IN NUMBER);
```

DBMS_RESUMABLE.GET_TIMEOUT The purpose of the **get_timeout** function is to return the current value of the current session's resumable space management time-out setting in seconds. There are no parameters to this procedure. This function returns a NUMBER type.

DBMS_RESUMABLE.SET_TIMEOUT The purpose of the **set_timeout** procedure is to set the value of the current session's resumable space management time-out setting in seconds. This change is immediate in its effect. The syntax for the **set_timeout** function is shown here:

```
PROCEDURE DBMS_RESUMABLE.SET_TIMEOUT
(timeout IN NUMBER);
```

DBMS_RESUMABLE.SPACE_ERROR_INFO The purpose of the **space_error_info** function is to return space-related errors. It is generally called by an after-suspend trigger (see the "After-Suspend Trigger" section). This function returns a great deal of information about the cause of a given statement failure in the form of its OUT parameters. This allows you to customize the system's response to a given

failure condition. Thus, you can choose to suspend statements for certain failure conditions (say, out of temporary tablespace space), and just abort the statement in the case of other conditions (such as running out of rollback segment space). This function returns TRUE if a suspended statement was discovered, and FALSE if it was not. The syntax for the **space_error_info** function is shown here:

```
FUNCTION DBMS_RESUMABLE.SPACE_ERROR_INFO
( error_type          OUT VARCHAR2,
  object_type        OUT VARCHAR2,
  object_owner       OUT VARCHAR2,
  table_space_name   OUT VARCHAR2,
  object_name        OUT VARCHAR2,
  sub_object_name    OUT VARCHAR2)
RETURN BOOLEAN;
```

Resumable Space Management Restrictions

Oracle9i resumable space management comes with certain limitations when used with dictionary-managed tablespaces. If the creation of an object (that is, table or index) fails and the DDL includes an explicit *maxextents* clause that caused the failure, then that failure will cause an error to be generated and the object creation will not be resumable. The solution to this problem is to set *maxextents* to UNLIMITED while creating the object and then **alter** the object to reset *maxextents*, if that is what you wish to do. This applies only to DDL that is actually creating the object, not subsequent DML operations (**select**, **insert**, **update**, or **delete**) on that same object. Also, this does not apply to locally managed tablespaces.

Another limitation with regard to dictionary-managed tablespaces has to do with space management for rollback segments. If a rollback-segment space allocation fails, and the associated rollback segment is in a dictionary-managed tablespace, then the operation will fail. This restriction does not apply if you are using Oracle's new UNDO tablespaces, or if the tablespace that the rollback segments belong to is locally managed.

Note that statements that involve remote operations (such as DML statements using database links) do not support resumable space management. In regard to parallel execution, any parallel process that runs into a space management error will be suspended, and will restart when the problem is corrected. The remaining parallel processes will continue until they run into an error and are suspended or they complete. Note that this might cause multiple calls of the after-suspend trigger (see the next section for more on the after-suspend trigger). Note that if the parallel execution server process receives any nonrecoverable error, then the entire statement will fail and any suspended process will be aborted.

After-Suspend Trigger

Each time a SQL statement is suspended because of a space management failure, Oracle will call an after-suspend event trigger, if one exists. This trigger can be used for a number of reasons, including controlling which type of space allocation

failures you wish to suspend, and which you wish to allow. You can also use the trigger for notification purposes, such as sending e-mail to notify the DBA or a monitoring group that the process had failed. The Oracle documentation provides a good example of an after-suspend trigger. Check it out!

Data Dictionary Views Associated with Resumable Space Management

The main views associated with resumable space management are the DBA_RESUMABLE and USER_RESUMABLE views. These views contain information on each statement that is currently suspended. Here is an example of a query against the DBA_RESUMABLE view:

```
SQL> select user_id, session_id, error_msg from dba_resumable;
USER_ID SESSION_ID ERROR_MSG
-----
26      7          ORA-30036: unable to extend segment by 128 in undo tablespace
          'SECOND_UNDO'
```

In this example, session 7 has a suspended session. It is apparently stalled, waiting for UNDO space in the UNDO tablespace called SECOND_UNDO. At this point, a DBA would have a few options to resume this statement. The DBA could add space to the UNDO tablespace or enable AUTOEXTEND. The DBA could also choose to just wait for UNDO segments to be released by other transactions so they could be used. Of course, once Oracle has resolved the problem, it will automatically restart the suspended session.

Another view that has some use with relation to resumable space management is the V\$SESSION_WAIT view. In this view, there will be an event for each statement that is suspended. The name of the event is "Suspended on space error." Here is a query that displays this event:

```
SQL> select sid, event, seconds_in_wait from v$session_wait WHERE sid = 7;
SID EVENT
-----
7 statement suspended, wait error to be cleared 517
```

Also, the V\$SYSTEM_EVENT and V\$SESSION_EVENT views provide wait information on suspension events. Here are some example queries from those tables of a session waiting on a suspended session:

```
SQL> select event, total_waits, time_waited from v$system_event where
2 event like '%suspend%';
EVENT TOTAL_WAITS TIME_WAITED
-----
statement suspended, wait error to be cleared 186 37204

SQL> select sid, event, total_waits, time_waited from
2 v$session_event where event like '%suspend%';
SID EVENT TOTAL_WAITS TIME_WAITED
-----
7 statement suspended, wait error to be cleared 193 38605
```

Resumable Space Management and Oracle Utilities

In this section, you will learn about the use of reusable space management in concert with Oracle utility programs. First, we'll look at how this feature works with Oracle's IMP and EXP facilities. Then you will learn about resumable space management and SQL*Loader.

Using imp and exp with Resumable Space Management

To facilitate the use of resumable space management in Oracle9i with Oracle's **imp** and **exp** utilities, new parameters have been added to the **imp** command. The new parameters include

- *resumable*
- *resumable_name*
- *resumable_timeout*

These parameters enable resumable space management when using the **imp** utility. As with an Oracle user session, if the **imp** session is suspended, then the **imp** process will freeze until the suspension time-out passes (in which case, the process will fail). Of course, if the space failure is fixed during the suspension, then the **imp** process will simply continue to process the work until it's finished or it's suspended again. The default is to not have resumable space features enabled with the **imp** facility.

Using SQL*Loader with Resumable Space Management

Oracle's SQL*Loader product has had the following command-line parameters added:

- *resumable*
- *resumable_name*
- *resumable_timeout*

These parameters enable resumable space management features when loading data using SQL*Loader. The default is to not have resumable space features enabled.

Persistent Initialization Parameters

In this section, you will learn to set up the new server parameter file (spfile), which allows parameter settings to be persistent. We will then look at how to change parameters and ensure that the change is persistent. Finally, we will look at some management issues related to spfiles.

Oracle has long offered the ability to change a certain number of database parameters dynamically using the **alter system** or **alter session** command. One problem with this was that the dynamic changes that were made were valid only until the instance was shut down. For those changes to persist through future database shutdown-startup cycles, you needed to change the database parameter file manually. If these changes were not made, then the database settings would revert to those in the database parameter file during the next database shutdown-startup cycle, which could lead to a variety of problems.

Creating the Server Parameter File

The spfile is a binary version of the text database file that is created using a new Oracle command, **create spfile**. The spfile must reside on the server and can be used only if the database is started using the *pfile* parameter of the STARTUP command. If the *pfile* parameter is not used, then the database will revert to trying to use the standard database parameter file. Of course, you can still use the *pfile* parameter to point to a nondefault database parameter file. The rule for using the spfile is as follows: When you start the database, Oracle first checks for a system parameter file called spfile<instance_name>.ora, then a system parameter file called spfile.ora in the \$ORACLE_HOME/dbs directory on a UNIX system. If neither exists, then it looks for the traditional init.ora file. The Oracle9i database creation assistant will create an spfile (you can choose to disable this operation if you like) for you when it creates your database. It will also start the database using the spfile it created. The steps for creating and using a server parameter file are as follows:

1. Create the server parameter file using the **create spfile** command as shown here:

```
CREATE spfile='c:\oracle\admin\mydb\pfile\spfilemydb.ora';
FROM PFILE='c:\oracle\admin\mydb\pfile\initmydb.ora';
```

2. Shut down the database and restart it using the following commands:

```
SHUTDOWN IMMEDIATE
STARTUP PFILE= c:\oracle\admin\mydb\pfile\spfilemydb.ora
```

NOTE

The only comments that will survive a conversion to a spfile are those contained in the same line of a parameter. Stand-alone and header comments are not preserved.

A few rules to note about the **create spfile** command: first, if you do not define the name of the server file to be created, then Oracle will create an spfile in the default location of the Oracle parameter files (that is, %ORACLE_HOME\database

in NT or \$ORACLE_HOME/dbs in UNIX). The spfile will take on the default name spfile{oracle_sid}.ora. Further, if you do not define the name of the source pfile, then the default pfile for the database will be used.

When creating an spfile with the **create spfile** command, Oracle will overwrite the old spfile, so be careful. If you try to overwrite an spfile that you started the database with, then Oracle will generate an error.

Also, you cannot point to an spfile from a normal parameter file by using the **ifile** command. Instead, include an **spfile** command in the normal parameter file to point to an spfile.

Setting Persistent Parameters

In Oracle9i, you will still use the **alter system set** command to change a parameter and ensure that change is persistent across database shutdowns. A new clause to the **alter system** called **scope** has been introduced to allow the DBA to indicate how Oracle should interpret the desired persistence of the change. There are three options to the **scope** clause:

- **spfile** The parameter being changed will be changed in the spfile, but will not take effect until the next time the instance is cycled. This is the only way that static parameters can be modified in the spfile. Attempting to use any other **scope** for static parameters will result in an error.
- **memory** The parameter being changed will be changed only for the instance that is currently active. The parameter in the spfile will not be updated, and thus the parameter is not persistent. You cannot use this parameter for static parameters.
- **both** The parameter being changed will be changed in the current instance and will be updated in the spfile for the database. Thus, the change will persist through future cycles of the database instance.

Also, a new **comment** clause has been added to the **alter system** command. This clause allows you to associate comments with the parameter being set. This allows you to document the change being made (which is a really good idea!). Here are a couple of examples of using the **alter system** command to change parameters in an spfile:

```
ALTER SYSTEM
SET query_rewrite_enabled=TRUE COMMENT='Change on 9/30' SCOPE=BOTH
```

This example sets the *query_rewrite_enabled* parameter to a value of TRUE. We have also associated this parameter change with a comment that indicates the date

we made the change. Finally, we indicate the **scope** of the change is BOTH, so that the change will take place immediately and will also be reflected in the spfile of the database. Here is another example:

```
ALTER SYSTEM
SET shared_pool_size=100m, COMMENT='Change on 9/30' SCOPE=BOTH,
aq_tm_processes=5, COMMENT='Change on 9/30' SCOPE=BOTH
```

In this case, we are changing two parameters at the same time. If you wish to remove a parameter that is set, you would simply issue an **alter system** command with the parameter set to 0 (or "" if it is a string parameter), as shown in this example:

```
ALTER SYSTEM SET db_16k_cache_size=0;
```

Also, you can use the *deferred* parameter to cause the change to take effect only for future sessions that connect to the database, as in this example:

```
ALTER SYSTEM
SET query_rewrite_enabled=TRUE COMMENT='Change on 9/30' SCOPE=MEMORY
DEFERRED;
```

If you are running real application clusters in Oracle9i, you can define the *sid* that the change is associated with by using the *sid* parameter. If you wish the change to take effect for all instances, you can use an asterisk (*) in the *sid* parameter to indicate this. The default for *sid* is * if the database was started with an spfile; otherwise, the default is the current instance. Here is an example of the use of the *sid* parameter:

```
ALTER SYSTEM
SET query_rewrite_enabled=TRUE COMMENT='Change on 9/30' SCOPE=MEMORY
DEFERRED SID=mysid2;
```

Finally, if you are using real application clusters, you might want to remove a previously set spfile setting for one of the instances of the cluster. To do this, you can use the **alter system** command with the **reset** clause, as shown here:

```
ALTER SYSTEM
RESET query_rewrite_enabled SCOPE=MEMORY DEFERRED SID=mysid2;
```

Managing the spfile

Let's look at a few of the management issues relating to spfiles. First, we will look at how to create a text parameter file from an existing spfile. Then we will look at some of the data dictionary views that can be used in Oracle9i that relate to spfiles.

Creating a Text Parameter File from an spfile

There might be times that you will want to convert an existing spfile into a standard text database parameter file. You might want to do this for documentation purposes, backup and recovery purposes, or if you just want a parameter file to work with when creating a new database. Oracle provides the ability to convert an spfile into a text parameter file through the use of the new **create pfile** command, which works like the **create spfile** command, but in reverse. Here is an example of the operation of this command:

```
CREATE PFILE='c:\oracle\admin\mydb\pfile\initmydb_pfile.ora';
FROM spfile='c:\oracle\admin\mydb\pfile\spfilemydb.ora';
```

Note, in this case, that we have defined the location of the pfile to be created. If we did not define the pfile name and location, Oracle would default to putting the file in the default pfile location (that is, %ORACLE_HOME\database in NT or \$ORACLE_HOME/dbs in UNIX). The default name of the file will be the default parameter filename for the database. If you do not include a filename for an spfile, then the default database spfile name will be used. Of course, if you choose to use the defaults and the source file does not exist, then the statement will fail.

Data Dictionary Views and spfiles

The data dictionary provides some useful views that allow you to display not only the current value of database parameters, but also how the parameters will be set in the future (again, assuming we have set them using DEFERRED or SCOPE=spfile). There are three principle views that we can use to display database parameters. Let's take a moment to look at these.

V\$PARAMETER This view remains generally unchanged from Oracle8i. It provides the current settings that are in effect for the instance for a given parameter. One thing that is new is that comments that are associated with the parameter are listed in this view.

V\$PARAMETER2 This view is new for Oracle9i. This view lists the parameters that are current for a given database session. Its format is generally the same as V\$PARAMETER except for the inclusion of an ORDINAL column. The ORDINAL column shows you the actual order of the parameters, which is handy for parameters that have included a list of strings.

V\$SPPARAMETER This view is new for Oracle9i. Its purpose is to allow you to view the settings of parameters in the instance spfile.

Oracle Supports Fewer Platforms in 9i

Here is a list of platforms that Oracle has announced it will support for Oracle9i:

- Sun Sparc Solaris
- HP 9000 Series HP-UX (64-bit)
- Compaq Tru64 UNIX
- AIX-based systems (64-bit)
- Sun Sparc Solaris (64-bit)
- Linux Intel
- IBM OS/390
- IBM DYNIX/ptx
- Alpha OpenVMS
- Microsoft Windows NT
- Microsoft Windows 2000