ArcGIS® 9

ArcSDE® Configuration and Tuning Guide for Oracle®

Contents

CHAPTER 1

Getting started

Creating and populating a geodatabase is arguably a simple process, especially if you use ESRI's ArcCatalog™ or ArcToolbox™ to load the data. So why is there a configuration and tuning guide? Well, while database creation and data loading can be relatively simple, the resulting performance may not be acceptable. It requires some effort to build a database that performs optimally. Also, as an Oracle® user, you have some choices for storing the geometry of your spatial data.

This book provides instruction for configuring the physical storage parameters of your data in the database management system (DBMS), as well as information about the available options you have to store the geometry. This book also provides some important guidelines for configuring and tuning the Oracle instance itself.

Tuning and configuring the Oracle instance

Building an efficient geodatabase involves properly tuning and configuring the Oracle instance and proper arrangement and management of the database's tables and indexes. Chapter 2, 'Essential Oracle configuring and tuning', teaches you how to do just that.

Chapter 2 lists the necessary steps to create a geodatabase. You will learn how to properly:

- Create an Oracle database.
- Create the tablespaces that will store your tables and indexes.

- Tune the Oracle instance that will mount and open the database.
- Manage the optimization statistics of the tables and indexes after they have been created and populated.

Arranging your data

Every table and index created in a database has a storage configuration. How you store your tables and indexes affects your database's performance.

DBTUNE storage parameters

How is the storage configuration of the tables and indexes controlled? ArcSDE® reads storage parameters from the SDE.DBTUNE table to define physical data storage parameters of ArcSDE tables and indexes. The storage parameters are grouped under configuration keywords. You assign configuration keywords to your data objects (tables and indexes) when you create them from an ArcSDE client program.

The initial source of storage parameters is the dbtune.sde file found under the ArcSDE etc directory. When the ArcSDE sdesetupora* setup command executes, the configuration parameters are read from the file and written into the DBTUNE table.

Most ArcSDE storage parameters are configuration strings and represent the entire storage configuration for a table or index. Most SDE.DBTUNE storage parameters hold the parameters of an Oracle CREATE TABLE or CREATE INDEX statement.

The sdedbtune command provides the ArcSDE administrator with an easy way to maintain the SDE.DBTUNE table. The sdedbtune command exports and imports all the records of the SDE.DBTUNE table to a file in the ArcSDE etc directory.

The ArcSDE installation creates the SDE.DBTUNE table. If the dbtune.sde file is absent or empty, sdesetupora* creates the SDE.DBTUNE table and populates it with default configuration keywords representing the minimum ArcSDE configuration.

In almost all cases, you will populate the SDE.DBTUNE table with specific storage parameters for your database. Chapter 3, 'Configuring DBTUNE storage parameters', describes in detail the SDE.DBTUNE table and all possible storage parameters and default configuration keywords.

Spatial data storage choices

The SDE.DBTUNE storage parameter GEOMETRY_STORAGE allows you to choose from three possible spatial column storage formats.

The three possible storage formats are:

- ArcSDE compressed binary with LONG RAW. The ArcSDE geometry is stored in a 'LONG RAW' column in a separate feature table. A business table's spatial column is a foreign key reference to the records of a feature table. This is the default spatial storage format for ArcSDE.

- ArcSDE compressed binary with binary large object (BLOB). The schema of this storage format is the same as the previous one except that the geometry is stored in the BLOB data type.
- Oracle Spatial geometry type. The object-relational model extends the database model to include an SDO_GEOMETRY type. Under this storage format, the spatial column is an SDO_GEOMETRY data type, and no foreign key reference to another table storing a geometry column is required.

These spatial storage choices are discussed more fully later in this book.

Appendix C, 'ArcSDE compressed binary', describes the ArcSDE compressed binary for both LONG RAW and BLOB.

Appendix D, 'Oracle Spatial geometry type', describes ArcSDE support for the Oracle Spatial storage format.

Creating spatial data in an Oracle database

ArcCatalog and ArcToolbox are graphical user interfaces (GUIs) specifically designed to simplify the creation and management of a spatial database. These applications provide the easiest method for creating spatial data in an Oracle database. With these tools you can convert existing ESRI® coverages and shapefiles into ArcSDE feature classes. You can also import an existing ArcSDE export file containing the data of a business table, feature class, or raster column.

Multiversioned ArcSDE data can be edited directly with the ArcMap™ GUI.

An alternative approach to creating spatial data in an Oracle database is to use the administration tools provided with ArcSDE.

Chapter 4, 'Managing tables, feature classes, and raster columns', describes the methods used to create and maintain spatial data in an Oracle database.

ArcSDE geodatabase maintenance

Periodically the administrator must perform various maintenance tasks on the ArcSDE geodatabase to maintain performance. Tasks, such as updating table and index statistics, rebuilding out of balance indexes, and compressing the states table, are discussed.

National language support

If you intend to support a database that does not use the Oracle default character set, you will have to take a few extra steps when creating the Oracle database. You will also need to set the national language system environment of the client applications.

Chapter 5, 'National language support', describes how to configure the Oracle database and set up the application environment.

Backup and recovery

Developing and testing a backup strategy is every bit as important as the effort put into creating the database. A good backup strategy protects the database in the event of a media failure.

Chapter 6, ‘Backup and recovery’, lists the ArcSDE files that must be included as part of the regular Oracle backup. In addition, suggested Oracle reference materials are listed for further reading.

CHAPTER 2

Essential Oracle configuring and tuning

The performance of an ArcSDE service depends on how well you configure and tune Oracle. This chapter provides basic guidelines for configuring an Oracle database for use with ArcSDE. It assumes that you have a basic understanding of the Oracle data structures, such as tablespaces, tables, and indexes, and that you are proficient with Structured Query Language (SQL). Refer to Oracle's extensive documentation, in particular *Oracle Database Administrator's Guide, Oracle Concepts Guide,* and *Oracle Database Performance Tuning Guide and Reference,* for your Oracle release.

How much time should you spend tuning?

The importance of having a well-tuned database depends on how it is used. A database created and used by a single user does not require as much tuning as a database that is in constant use by many users. The reason is quite simple—the more people using a database, the greater the contention for its resources.

By definition, tuning is the process of sharing available resources among users by configuring the components of a database to minimize contention and maximize efficiency. The more people you have accessing your databases, the more effort is required to provide access to a finite resource.

A well-tuned Oracle database makes optimum use of available central processing unit (CPU) time and memory while minimizing disk input/output (I/O) contention. Database administrators approach this task knowing that each additional hour spent will often return a lesser gain in performance. Eventually, they reach a point of diminishing returns when it becomes impractical to continue tuning; instead, they continue to monitor the database and address performance issues as they arise.

Reducing disk I/O contention

Although disk I/O contention has been alleviated through the advancement of hardware technology, it remains an important consideration to the database administrator. Disk I/O contention within an Oracle database is minimized by properly arranging the components of the database throughout the file system. Ultimately, the database administrator must reduce the possibility of one process waiting for another to complete its I/O request. This is often referred to as “waiting on I/O”.

Creating a database using the Oracle installer

Before installing Oracle and creating the database, decide where to position the software and the files of the Oracle database. The Oracle installer program will request this information.

If you have already installed Oracle and created your database files, you should still read the sections that follow. Although it involves more effort, you can move the Oracle database files after they have been created.

Defining the database’s components and their size

The physical components of an ArcSDE service and the underlying Oracle database, as they exist on any given file system, include the ArcSDE and Oracle software and all of the physical files (data files, redo log files, and control files) of the Oracle database. Each of the components is described below.

Software

The software includes both Oracle and ArcSDE. The ArcSDE software occupies approximately 60 MB of space. The disk space occupied by the Oracle software varies depending on the version of Oracle and products you choose to install. Please see your Oracle installation notes for further details.

Control files

The control files maintain an inventory of the overall physical architecture of an Oracle database. During the creation of the database, specify at least three control files on different disk drives. If a disk containing a control file fails, the Oracle server must be shut down. Recovery is accomplished by reconfiguring Oracle to use only copies of good—uncorrupted and current—control files. It is essential that control files be placed on physically separate disks so that a disk failure does not cause the loss of all control files.

Note: The ability of an Oracle database to recover after loss of a control file depends upon the existence of identical, current copies of the lost control file. ESRI does *not* recommend operating ArcSDE under Oracle without having multiple control files on *physically separate* disk drives.

Control files are dynamic and initially require about 4 MB of disk space and can grow to more than 10 MB depending on the activity of your database.

The initialization parameter CONTROL_FILE_RECORD_KEEP_TIME controls the size of the control files. By default, this parameter is set to 7—instructing Oracle to overwrite its reusable section every seven days. For more information on initialization parameters, refer to ‘Setting the Oracle initialization parameters’ later in this chapter.

Online redo log files

The online redo log files record the changes made to the database. Oracle requires a database to have at least three online redo log file groups present.

An Oracle database that receives regular edits (inserts, updates, or deletes) has highly active online redo log files. Writes to the current online redo log file occur according to the following schedule:

- The log buffer becomes one-third full.
- Any session issues a commit.
- Every three seconds if the buffer cache contains nonlogged dirty blocks.

It is important to physically separate the online redo log files from other data files that also experience high rates of I/O. Whenever possible, create the log files on their own disk drives or with other relatively static files.

Each time a log file fills up, a log file switch occurs and Oracle begins writing to the next log file. When a log switch occurs, all log files in a log file group are closed, all log files in the next group are opened, and a checkpoint occurs. Checkpoints help ensure the success of database recovery by establishing a starting point that reflects a consistent state of the database. Frequent checkpoints reduce the amount of time taken to recover a database. However, checkpoints are costly operations and should not occur too frequently. This can be done by choosing suitable values for size of the redo log files and the initialization parameters: LOG_CHECKPOINT_INTERVAL, LOG_CHECKPOINT_TIMEOUT, and FAST_START_MTTR_TARGET and setting the size of the redo log file so that it does not fill before the LOG_CHECKPONT_INTERVAL or LOG_CHECKPONT_TIMEOUT occurs. See Chapter 6 of this manual for details.

The size and number of online redo log files in your database depend on the type of database. This section will describe three basic kinds of databases:

- A newly created database.
- An Online Transaction Processing (OLTP) database. A multiversioned ArcSDE database that is constantly edited while it is being queried is an example of an OLTP database.
- A read-only database, meaning a database that, once loaded, receives changes at posted intervals. An ArcIMS® database is an example of a read-only database.

Establishing a new spatial database

With the vast amount of spatial data available as coverage and shapefile format, many spatial databases are mass populated immediately following their creation. For this type of database, create three log file groups each containing at least one large redo log file. If possible, place all log files on a disk drive separate from all other data files. In this situation, it is not unreasonable to create log files in excess of 1 GB.

After you have finished loading the data, connect to Oracle as the database administrator (DBA) and issue a checkpoint with ALTER SYSTEM CHECKPOINT. Then create new, smaller log files. The size and number of the log files depend on what kind of database it will become, either OLTP or read-only (see below).

When loading a database you may turn off archiving. You obtain a performance gain by eliminating the periodic copy to the archive log destination following a log switch. It is just as easy to recover the database from your load scripts and source data as it is to reapply the changes stored in the archive logs. Remember to turn archiving on after the database has been loaded if it is going to be an OLTP database.

OLTP database

For these types of databases, the redo log files should be large enough to ensure that the checkpoint is not a frequent occurrence. Redo log files should also be mirrored to provide maximum protection against loss of transaction data.

If you are archiving the redo log files, create three to 10 redo log file groups having log files about 50 MB in size. If possible, place them on disk drives that experience very low I/O. The archive log file destination should also be placed on a separate disk drive to ensure that disk failure does not cause the simultaneous loss of the active redo log files *and* the archived redo log files.

If you are not going to archive your log files, your total space given to redo log files should be enough to store all log entries generated between full database backups. Depending on the size of the redo log files and the amount of redo log entries created by changes to the database, there is always a chance that all redo logs will fill before they are backed up, reducing chances for complete database recovery.

Note: ESRI does *not* recommend that NOARCHIVELOG be the normal operating mode of an OLTP database due to the risk of losing committed transactions in the event of media failure.

In either case, you should mirror the online redo log files. Place the mirror copy of the online redo log file on a physically separate disk drive from the original copy.

Read-only databases

Some ArcSDE databases become relatively static following their creation. Such databases receive posted intervals of changes over their lifetime. For this type of database, create three 50 MB online redo log files. Since they’re used infrequently, positioning is not as critical as for the other two types of databases just described.

Monitoring the log files

For all three types of databases, connect as the SYSTEM user and issue the following query to determine if your online redo log files are large enough and if the checkpoint frequency is occurring at a desirable interval:

```
SELECT TO_CHAR(FIRST_TIME,'dd-mon-yy hh24:mi:ss') FROM V$LOGHIST;
```

This is an example of the output:

```
TO_CHAR(FIRST_TIME)
------------------
04-nov-99 13:15:14
04-nov-99 13:21:04
04-nov-99 13:27:04
04-nov-99 13:32:36
```

The example output shows the log switches are occurring at intervals greater than five minutes, the interval at which Oracle issues the checkpoints. If the interval was less than five minutes, the DBA should consider increasing the size of the online redo log files.

Modifying the online redo log files

To change the size of the redo log files, you must create new log file groups of the correct size and drop old log file groups that are the wrong size. You may only drop a log file group that is INACTIVE. Oracle requires that you always have at least two log file groups.

Follow this procedure using the SQL statements listed below.

0. Determine which of the existing log file groups is current:

```
SELECT GROUP#, STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        1 INACTIVE
        2 INACTIVE
        3 CURRENT
```

0. Add at least three new log file groups with their new size by executing this statement three times:

```
ALTER DATABASE ADD LOGFILE
('<path to log file member1>','<path to log file member2>',...)
SIZE <size>;

ALTER DATABASE ADD LOGFILE ...  ;

ALTER DATABASE ADD LOGFILE ...  ;

SELECT GROUP#, STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        1 INACTIVE
        2 INACTIVE
        3 CURRENT
        4 INACTIVE
        5 INACTIVE
        6 INACTIVE
```

Note: ESRI recommends that you always mirror the online redo log file groups across physically separate disk drives.

0. Execute the correct number of manual log switches required to make the first log file group current.

```
ALTER SYSTEM SWITCH LOGFILE;

SELECT GROUP#, STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        1 INACTIVE
        2 INACTIVE
        3 ACTIVE
        4 CURRENT
        5 INACTIVE
        6 INACTIVE
```

0. If the database is running in ARCHIVELOG mode, the log file will have an ACTIVE status until the archive process for that file has completed. If a redo log group to be dropped has an ACTIVE status, you may force the archive to occur with the ALTER SYSTEM ARCHIVE LOG GROUP command.

```
ALTER SYSTEM ARCHIVE LOG GROUP 3;

SELECT GROUP#, STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        1 INACTIVE
        2 INACTIVE
        3 INACTIVE
        4 CURRENT
        5 INACTIVE
        6 INACTIVE
```

0. Remove the old log file groups, identifying them by their group numbers.

```
ALTER DATABASE DROP LOGFILE GROUP 1;

ALTER DATABASE DROP LOGFILE GROUP 2;

ALTER DATABASE DROP LOGFILE GROUP 3;

SELECT GROUP#, STATUS FROM V$LOG;

GROUP#    STATUS
--------- ----------------
        4 CURRENT
        5 INACTIVE
        6 INACTIVE
```

Tablespace data files

The tablespace represents Oracle's logical storage container. Each tablespace has one or more physical data files assigned to it.

System tablespace

The system tablespace stores Oracle's data dictionary. Each time Oracle parses a SQL statement, it checks the data dictionary for metadata concerning data objects referenced by the statement. Among other things, Oracle ensures the data objects actually exist and the user has the proper privileges.

If possible, place the SYSTEM tablespace on a disk of low to moderate activity.

Undo tablespaces

The undo tablespaces store undo segments, which maintain the undo image needed to roll back aborted transactions. Undo segments also provide read consistency for queries started prior to another session's transaction.

Determine the storage parameters of the undo tablespace and the undo segments by the type of transactions using them. ArcSDE has three basic categories of transactions.

The AUTOCOMMIT interval—Initial loading of data into an ArcSDE database generally entails converting an ArcSDE coverage, shapefile, ArcSDE export file, or a data vendor's format into an ArcSDE feature class.

To avoid exceeding the undo space, ArcSDE provides a commit interval, a threshold that causes inserts, updates, or deletes to be committed. The commit interval also serves to regulate transaction size. The commit interval defaults to 5,000 features and is set with the ArcSDE server configuration AUTOCOMMIT parameter.

Refer to *Managing ArcSDE Application Servers* for more information on the AUTOCOMMIT parameter and how to set it.

Managing the large transactions of the multiversion geodatabase compress— Perodically, the ArcSDE administrator is required to compress the states of a multiversioned database. To guarantee the consistency of the geodatabase, compress transactions potentially grow to a large size. Therefore, for Oracle databases configured with automatic undo space management, make sure the UNDO_POOL is not set too low for the SDE user and that enough undo tablespace exists to complete the compress.

Alternatively, if you are manually controlling the undo space using rollback statements, you should create a separate rollback tablespace that is large enough to hold the transactions of the compress operation, and assign one rollback segment to it. The size of the tablespace and the inherent rollback segment required to store the before image of the compress transactions depends on the number of states that will be deleted during the compress. As a rule of thumb, you should start with a rollback segment that is at least 300 MB. However, if the editing environment is extremely active or you infrequently compress the database, a much larger rollback segment may be required. Set the name of this rollback segment in the COMPRESS_ROLLBACK_SEGMENT storage parameter of the DEFAULTS DBTUNE configuration keyword. If this parameter is not set, the next available online rollback segment will be used. If the rollback segment is not large enough, the compress operation will fail since the transaction will be forced to roll back.

Managing undo space for online geodatabase transactions—As a rule, if you find that your transactions are rolled back because your undo space fills up, you should reduce the size of your transactions or increase the size of the undo tablespace. Large transactions delay recovery, increase overhead for queries that must access them for read consistency, and increase overhead for other transactions that must allocate additional extents. ArcSDE allows you to limit the size of your transactions by setting the AUTOCOMMIT server configuration parameter.

Temporary tablespace

An Oracle server process writes to a segment of temporary tablespace whenever it performs a sort or hash join that exceeds the memory allocated to the Oracle program global area (PGA) by the PGA_AGGREGATE_TARGET initialization parameter. Sorts occur when indexes are created (Oracle: CREATE INDEX statement), statistics are generated (ANALYZE statement), and queries require on-the-fly sorting (SELECT statements that include table joins, ORDER BY clauses, and GROUP BY clauses).

Note: Oracle uses the PGA_AGGREGATE_TARGET to allocate memory for sorting only if the WORKSPACE_POLICY is set to AUTO. If it is not, Oracle will use the older manual method of managing sort area which included setting the SORT_AREA_SIZE and HASH_AREA_SIZE parameters.

When establishing a new database, the temporary tablespace will need to be large enough to create the indexes. Oracle requires twice as much temporary space to create the indexes as it does to store it.

If you are using the ArcSDE compressed binary format to store your spatial data, the S<n>_IX1 index on the spatial index table is likely to be your largest index. Another candidate is the raster block's table index for a large raster stored in the ArcSDE geodatabase. Refer to Appendix A, 'Estimating the size of your tables and indexes', for information on determining the size of your indexes.

If you are storing your spatial data as an Oracle Spatial data type, refer to Appendix A, 'Tuning Tips and Sample SQL Scripts', of the *Oracle Spatial User's Guide and Reference* for more information on sizing temporary tablespace for the construction of the Oracle Spatial data type indexes.

After the data has been loaded and the indexes created, temporary tablespace is used for data sorts. Temporary tablespace is used when sorts exceed the maximum PGA shared space managed through the Oracle PGA_AGGREGATE_TARGET initialization parameter.

The temporary tablespace can be mixed with other data files of higher I/O since there is a low risk of disk I/O contention.

The Oracle Database Configuration Assistant creates the temporary tablespace for you. Make sure the total size of the tablespace is large enough to create your largest index.

If for some reason you must re-create the temporary tablespace, use a SQL CREATE TEMPORARY TABLESPACE statement similar to the following:

```
CREATE TEMPORARY TABLESPACE temp
   TEMPFILE 'C:\oradata\temp01.dbf' SIZE 300M
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

ArcSDE system tablespaces

The ArcSDE system tablespaces store the ArcSDE and geodatabase system tables and indexes created by the ArcSDE sdesetupora* command. The number and placement of the tablespaces depend on the intended use of the ArcSDE database.

The placement of these tables and their indexes is controlled by the storage parameters of the DBTUNE DATA_DICTIONARY configuration keyword. The DATA_DICTIONARY keyword is used exclusively for the creation of the ArcSDE and geodatabase system tables.

Multiversioned databases that support ArcGIS® OLTP applications have a highly active state tree. The state tree maintains the states of all editing operations that have occurred on tables registered as multiversioned. Four ArcSDE system tables—STATES, STATE_LINEAGES, MVTABLES_MODIFIED, and VERSIONS—maintain the transaction information of the versioned database's state tree. In this type of environment, these four tables and their indexes have their own DATA_DICTIONARY configuration keyword storage parameters.

In an active multiversioned database, the STATES_LINEAGE table can easily grow beyond one million records, occupying more than 26 MB of tablespace. The STATES table is much smaller, storing approximately 5,000 records, occupying approximately 2 MB of tablespace. The MVTABLES_MODIFIED table typically has approximately 50,000 records occupying approximately 1 MB of tablespace. The VERSIONS table is usually quite small with less than 100 rows occupying approximately 64 KB.

For highly active editing ArcGIS applications, the STATES, STATES_LINEAGE, and MVTABLES_MODIFIED tables and their indexes need to be created in separate tablespaces and positioned across the file system to minimize disk I/O contention.

If you are not using a multiversioned database, the aforementioned tables are dormant, in which case the tables can be stored with the other ArcSDE system tables and indexes.

The remainder of the ArcSDE and geodatabase system tables store information relating to schema changes. They are relatively small and have a low frequency of I/O. They should be grouped together and positioned with other tablespaces of high activity.

To summarize, if you are creating an active multiversioned database, create a 70 MB tablespace to store ArcSDE tables. On a separate disk drive create a 30 MB tablespace for the indexes.

If you are not going to use a multiversioned database, reduce the extent sizes of the STATE_LINEAGES, STATES, and MVTABLES_MODIFIED tables and their indexes to 40 KB.

For more information about the DATA_DICTIONARY configuration keyword, see Chapter 3, ‘Configuring DBTUNE storage parameters’.

Business table and index tablespaces

The Oracle installation creates the USER tablespace as the default user tablespace. There is no requirement by Oracle that you use the USER tablespace, and you may drop it if you want. If you are building a sizable, permanent spatial database, create tablespaces with names reflecting the data they will store.

The B_STORAGE DBTUNE storage parameter holds business table storage parameters.

The 15 MB INDX tablespace created by the Oracle installation process can be used to store indexes. However, for large spatial databases you will need to create index tablespaces whose name and size reflect the indexes they store. You may drop the Oracle-generated INDX tablespace if you do not intend to use it.

The B_INDEX_USER DBTUNE storage parameter holds the storage parameters of the business table indexes that you create.

A note about RAID storage devices

RAID, short for Redundant Array of Inexpensive Disks, is a method whereby information is spread across several disk drives to reduce latency, increase data availability, and/or improve data safety. In RAID storage, two or more physical disk drives are combined into one physical configurable device. RAID storage operates in various modes identified as RAID 0, RAID 1, RAID 4, RAID 5, and RAID 10.

With RAID 0, also called “stripe mode”, data may be spread over multiple disks. The first 4 KB might be written to disk 0, the second 4 KB block to disk 1, the third to disk 2, and so on. Data is distributed back to disk 0 after disk N is written. RAID 0 may be faster during sequential reads and sequential writes since successive blocks can be read or written nearly in parallel across multiple disks. However, RAID 0 provides no extra data safety since there is no duplication of any data.

RAID 1 provides an exact duplicate of the file information maintained on each disk. RAID 1 can be used on two disks with zero or more spare disks. RAID 1 protects data through redundancy; however, it is always slower to write since the data must be written to at least two locations.

RAID 4 is similar to RAID 1 except that one disk stores parity information and not duplicate data. There need be only one parity disk to protect any number of data disks. However, since the disk holding parity data will always be written to, it will become a bottleneck on writes. Like RAID 1, RAID 4 is also slower on writes because two physical blocks—the data block and the parity block—must be written for each logical block sent. The disk holding the parity data becomes a bottleneck since all data writes will affect this disk.

RAID 5 may be the most useful mode, since it eliminates the bottleneck of RAID 4 but still provides data protection. In RAID 5, parity information is distributed evenly among the participating disk drives.

RAID 1+0 is also useful. A group of drives is duplicated, akin to RAID 1, and the resulting group is striped, akin to RAID 0. The real benefit of this arrangement lies in the duplication, which provides data protection.

While RAID 1, RAID 1+0, and RAID 5 provide an additional level of protection, ESRI does not recommend relying solely on a disk configuration for safeguarding data. ESRI recommends implementing and testing a database backup strategy that copies the appropriate Oracle files to a safe offline location. As discussed in Chapter 6, this should be accomplished immediately after installing a new ArcSDE database.

Arranging the database components

Minimizing disk I/O contention is achieved by balancing disk I/O across the file system—positioning frequently accessed "hot" files with infrequently accessed "cold" files. Estimate the size of all the database components and determine their relative rates of access. Position the components given the amount of disk space available and the size and number of disk drives. Diagramming the disk drives and labeling them with the components help keep track of the location of each component. Have the diagram handy when you create the Oracle database.

Establish a maximum data file size

Choose the maximum size of a data file. It is a good policy to set a maximum size limit for your data files because doing so facilitates interchanging them and helps ensure that files can be copied to your backup medium.

After the database has been in use for some time and a "normal" pattern of usage has been established, heavily accessed data files sharing the same disk drive need to be separated. They can only be interchanged with medium- or low-accessed data files of a similar size (unless you have a lot of free space on your disk drives).

Check your operating system documentation for further details on maximum data file size.

Determine the size of the tables and indexes

To determine the sizes of tables and indexes stored in an ArcSDE database, refer to the formulas listed in Appendix A, 'Estimating the size of your tables and indexes'.

Store tables and indexes by access

Decide which tables to store together in the same tablespace based on expected access. Ideally, storage should be distributed as broadly as possible, to spread out the file access load.

Locally managed tablespaces

Starting in Oracle9*i*, all tablespaces are locally managed by default. A locally managed tablespace offers faster access for data storage since lookups in the data dictionary are avoided. Tablespace management is accomplished by including a bit map at the head of each data file indicating which segments are allocated and which are free. The bit map is based on a specified minimum size of an extent.

You should create tablespaces that contain a single table or index according to the size of the table or index they contain. A tablespace may be so large that it requires several data files. If the tablespace is locally managed you should add about 1 MB of extra space in each data file to accommodate the segment allocation bit map.

For example, if a tablespace must store a table or index that will grow to 7 GB and the maximum data file size allowed is 2 GB, the tablespace should include three, 2 GB data files and one data file that is somewhat larger than 1 GB.

Because the extent bit map takes about 1 MB of space in each data file, a data file of 2,048 MB will contain 2,047 MB of usable space. Extent allocation in this tablespace should not be done with segment sizes of 1,024 MB, but slightly smaller, say 1,023 MB, so that two extents plus the bit map will fit into the file.

Example:

```
create tablespace roads datafile '/gis1/oradata/roads1.dbf' size 2048M
extent management local uniform size 1023M;

alter tablespace roads add datafile '/gis2/oradata/roads2.dbf' size 2048M;

alter tablespace roads add datafile '/gis3/oradata/roads3.dbf' size 2048M;

alter tablespace roads add datafile '/gis4/oradata/roads4.dbf' size 1024M;
```

Find the sum of the sizes of the objects you will store together, to determine the size of their tablespaces. If you expect the table or index to grow in the future, be sure to allow for this as well.

Positioning the files

Once you have estimated the size of the data files, determine where to position them on the file system. This section provides a list of guidelines that you may not be able to follow in its entirety, given the number and size of your disk drives. The guidelines have been listed in order of importance—from the greatest to the least.

Store the online redo log files on their own disk drive. In a database that is frequently edited, the online redo log files are the most active in terms of I/O. If you cannot position them on their own disk drive, store them with other files that experience relatively low rates of I/O.

Keep the UNDO tablespace files separate from the redo log files. The undo segments are frequently accessed when a database is edited. Try to separate these data files from other highly active data. Doing so improves the rate at which Oracle is able to process transactions.

Position the system tablespace data file with other data files that experience high I/O activity. The access to these data dictionary tables is moderately low because their data is cached in the shared pool and the buffer cache.

Position your business table and index data files according to their expected I/O. If you expect a particular data file to experience a high degree of I/O, try to position it alone on its own disk drive or with other data files of low to moderate activity.

The spatial index table of the ArcSDE compressed binary storage format is written to whenever new features are added to a feature, but the table is never read. The spatial index table's S<n>_IX1 index is a covering index; therefore, Oracle reads the values from the index and never accesses the table. Since the table is never read from, the I/O is low, so the positioning of this table is unimportant.

Repositioning data files

After the database has been in use for a while, you can examine the reads and writes to the data files with the following query:

```
select vd.name, vs.phyrds, vs.phywrts
from v$datafile vd, v$filestat vs
where vs.file# = vd.file#;
```

If you find that some disk drives are receiving a higher percentage of the I/O than others, you can balance the I/O by repositioning the data files. The data files are repositioned by shutting down the Oracle instance, performing a full backup, and moving the files using operating system commands to copy them from one disk to another. Using SQL*Plus, the instance is started, and the database is mounted but not opened:

```
startup mount
```

Before the database can be opened, you must update the control files with the new locations of the data files using the *alter database* statement:

```
alter database rename datafile 'old name' to 'new name';
```

Once all of the new locations have been entered, you may open the database.

```
alter database open;
```

Creating the database

The database should be created after the location of the database files has been determined. At the very least, you should assign the locations for the undo, temporary, and system tablespace data files; the control files; and online redo log files.

Once you have mapped out how you want the files to be arranged on disk, create the database.

First, you must install the Oracle and ArcSDE software.

Refer to the Oracle installation guide for instructions on installing the Oracle software and the ArcSDE installation guide for instructions on installing the ArcSDE software.

The Oracle installation provides you with the opportunity to create the Oracle database following the installation of the software. If you elect to do so, choose the custom database option so that you can position the Oracle files on the disks according to your layout.

The database creation scripts are not generated automatically. Instead, you are given the choice of generating the scripts and creating the database later or creating the database immediately without creating the scripts. If you want the scripts available as a record of how the database was created, then you will have to generate the scripts and run them following the completion of the installation process.

The control files, online redo log files, and Oracle database tablespaces (system, undo, and temporary) are created during the Oracle installation process. The USER and INDX tablespaces created by the Oracle installation process may be dropped if you do not want to use them.

Setting the data block size

During the installation process, you will have the opportunity to set the data block size for the database. Data blocks are the Oracle atomic unit of data transfer. The database block size is the standard block size, the block size of the system tablespace, and the default block size of any other tablespace unless a nonstandard block size is specified. Oracle supports up to four nonstandard block sizes. After a database has been created, you cannot change the database block size.

The recommended database data block size for ArcSDE applications is 16 KB.

Creating the business and index tablespaces

Once you have completed the installation of Oracle, create the tablespaces that store your tables and indexes. You may want to write a SQL script and execute it as the SYSTEM user within SQL*Plus. Alternatively, you could use the graphical user interface of the Oracle Enterprise Manager's Storage Manager to perform this task.

Creating the Oracle SDE user space

During the installation of ArcSDE, you will create the Oracle SDE user and the SDE user's default tablespace, to store the geodatabase system tables.

Note: The SDE user and its tablespace should only be used to manage and store ArcSDE system metadata. You should create separate users to store your ArcSDE data objects such as feature classes and raster datasets. You should not store these objects as the SDE user or in the SDE user's tablespace, since you could possibly crash the ArcSDE application server by filling up the SDE user's tablespace. Following the practice of storing only SDE metadata in the SDE user schema simplifies the management of ArcSDE.

Updating the storage parameters of the DATA_DICTIONARY keyword

Update the storage parameters of the DATA_DICTIONARY configuration keyword in the dbtune.sde file. (On UNIX® systems this file is located in the SDEHOME/etc directory. On Windows systems, the install shield will prompt you for the location of this file. Refer to the *ArcSDE for Oracle Installation Guide* for more information.) Set the extent sizes and the names of the tablespaces in which the geodatabase system tables will be stored. The geodatabase system tables are created by the ArcSDE sdesetupora* administration command.

Setting the SDE log file pool

ArcSDE provides log files for the temporary storage of *key* lists of table rows. Applications use ArcSDE log files to maintain such things as selected lists of records.

Starting at ArcSDE 9 a pool of log file tables may be created in the SDE user's schema and checked out by other users. With this new feature, it is now possible to create users that do not own log file tables since they can borrow those owned by the SDE user.

If you decide to create a Spatial Database Engine™ (SDE®) log file pool, you will need to consider the space required for these tables. The server configuration parameter SESSIONLOGPOOLSIZE regulates the number of log file tables created in the pool.

Setting the SESSIONLOGPOOLSIZE server configuration parameter greater than 0 will cause the log file data tables to be created in the SDE user's schema. You must create a tablespace large enough to hold all of the log file data for all users writing to the checked out tables. The amount of data and, therefore, the expected size of the log file tables depend on the number of log file tables and how the applications are expected to use them. If you expect your users to maintain large selection sets, create a tablespace large enough to store the tables and index of the pooled log file tables.

The DBTUNE LOGFILE_DEFAULTS keyword holds the storage parameters that regulate the storage of log file tables and indexes. (Refer to Chapter 3 for more information on DBTUNE parameters). Within the LOGFILE_DEFAULTS keyword, the SESSION_STORAGE parameter specifies the CREATE TABLE Oracle storage parameters for the log file data tables of the pool. The SESSION_INDEX parameter specifies the parameters of the CREATE INDEX statement for the indexes created on the log file data tables. The tablespaces specified in these storage parameters must be large enough to hold the data of the user's log file entries.

Installing the optional sdesys_util package

The optional sdesys_util package, found under the %SDEHOME%/tools/oracle directory, can be installed in the Oracle SYS users schema. This package contains stored procedures that allow you to create and manage the SDE user.

To install the sdesys_util package, log in as the Oracle SYS user and execute the package specification followed by its body:

```
connect sys/<password>

@sdesys_util.sps

Package created.

@sdesys_util.spb

Package body created.
```

You can allow a user other than SYS to execute the stored procedures of the sdesys_util package by executing the sdesys_util.grant_admin_privs that grants the necessary privileges. The usage of the grant_admin_privs stored procedure is as follows:

```
grant_admin_privs (
    administrator IN varchar2(30) DEFAULT 'system'
)
```

If you do not specify any arguments, the privileges are granted to the Oracle system user:

```
connect sys/<password>
exec sdesys_util.grant_admin_privs('JOE');
```

If a user, other than SYS, executes a stored procedure from the sdesys_util package, the user must qualify it with the SYS user's schema. For example, the system user would execute the grant_pipes_locks stored procedure as follows:

```
connect system/<password>
exec sys.sdesys_util.grant_pipes_locks;
```

Granting execute privileges on DBMS_PIPE and DBMS_LOCK to PUBLIC

ArcSDE uses the stored procedures in the DBMS_PIPE and DBMS_LOCK Oracle built-in packages. ArcSDE calls stored procedures of the DBMS_PIPE package when it stores and transmits ArcSDE ROWIDs. ArcSDE calls the stored procedures of the DBMS_LOCK package to add a row to the PROCESS_INFORMATION table whenever an ArcSDE session connects. Your Oracle DBA must connect to the Oracle instance as the SYS user and grant execute privileges on these packages to PUBLIC:

```
connect sys/<password>
grant execute on dbms_pipe to public;
grant execute on dbms_lock to public;
```

Alternatively, if you have installed the sdesys_util package you can connect as the Oracle SYS user and execute its sdesys_util.grant_pipes_locks stored procedure:

```
connect sys/<password>
exec sdesys_util.grant_pipes_locks;
```

Creating the Oracle SDE user

Before you run the sdesetupora* command to create the geodatabase system tables on a UNIX system, the SDE Oracle user must be created. The Microsoft Windows installation of ArcSDE has automated this procedure. However, you can, if you want, create the SDE Oracle user prior to installing ArcSDE.

Like any Oracle user, the SDE user requires a default tablespace. Review the previous discussion of the ArcSDE system tablespaces to determine whether or not you need to create separate tablespaces to store your ArcSDE system data.

Use the following SQL commands to create the SDE user. Substitute your own entries for the SDE user's password, default tablespace, and temporary tablespace:

```
connect system/<password>

create user sde
  identified by <password>
  default tablespace <sde user's default tablespace>
  temporary tablespace <temporary tablespace>;
```

Alternatively, you can use the SYS user's SDESYS.UTIL_CREATE_SDE stored procedure to create the SDE Oracle user. The usage for CREATE_SDE_USER is:

```
SDESYS_UTIL.CREATE_SDE_USER(
    sdedatafile IN VARCHAR2(256) DEFAULT NULL,
    sdetabspace IN VARCHAR2(30) DEFAULT 'SDE',
    temptabspace IN VARCHAR2(30) DEFAULT 'TEMP',
    sdepasswd IN VARCHAR2(30) DEFAULT 'SDE');
```

Executing this stored procedure without any arguments will create the SDE Oracle user with a default tablespace set to SDE, a temporary tablespace set to TEMP, and the password set to SDE:

```
    connect sys/<password>
    exec sdesys_util.create_sde_user;
```

CREATE_SDE_USER creates the SDE user's default tablespace if you specify the location of the tablespace's data file.

If the default tablespace already exists, CREATE_SDE_USER will check it to make sure it is big enough. If it is less than 100 MB, CREATE_SDE_USER issues a warning message telling you to either increase the size of the default tablespace or decrease the initial extents of the state, state_lineages, and mvtables DATA_DICTIONARY configuration keyword storage parameters. If you have configured these storage parameters to store these tables and their indexes in other tablespaces, then you may disregard this warning. The sdesetupora* command will fail if it cannot find enough space to store the geodatabase system tables and indexes.

Granting install privileges to the sde Oracle user

If you are creating a fresh install of ArcSDE, grant the following list of privileges to the SDE user. These privileges must be granted to the SDE user:

```
    SELECT ANY TABLE
    CREATE SESSION
    CREATE TABLE
    CREATE PROCEDURE
    CREATE SEQUENCE
    CREATE TRIGGER
    UNLIMITED TABLESPACE
```

Alternatively, you can call the GRANT_SDE_INSTALL_PRIVS stored procedure (found in the SDESYS_UTIL package) to grant these privileges to the SDE Oracle user:

```
    connect sys/password
    exec sdesys_util.grant_sde_install_privs;
```

Note: ArcSDE requires the Oracle SDE user be granted SELECT ANY TABLE privileges. Granting SELECT ANY TABLE privileges to the SDE user allows it to autoregister Oracle Spatial tables. Autoregistration occurs during a layer list when an ArcSDE application detects an Oracle Spatial table that is listed in the Oracle Spatial metadata table but is not listed in the SDE.LAYERS table. The SDE user must be able to confirm the presence of the Oracle Spatial table before autoregistering it. Since there is no guarantee that the owner of the table has granted select permissions on the table to the SDE user, the SDE user must be granted SELECT ANY TABLES privileges.

ArcSDE does not require the SDE user to have the UNLIMITED TABLESPACE privilege. From a space management perspective, the DBA may want to impose quotas.

Granting upgrade privileges to the SDE Oracle user

To upgrade a previous version of an ArcSDE database, grant the following list of privileges to complete the upgrade process. Following the successful completion of the upgrade process, you can reduce the privileges to those required for the install operation.

```
ALTER ANY INDEX
ALTER ANY TABLE
ANALYZE ANY
CREATE ANY INDEX
CREATE ANY PROCEDURE
CREATE ANY SEQUENCE
CREATE ANY TRIGGER
CREATE ANY VIEW
CREATE SESSION
DROP ANY INDEX
DROP ANY TABLE
DROP ANY VIEW
DROP ANY PROCEDURE
DROP ANY SEQUENCE
EXECUTE ANY PROCEDURE
SELECT ANY SEQUENCE
SELECT ANY TABLE
UNLIMITED TABLESPACE
```

You can grant this list of privileges by executing the SDESYS_UTIL.GRANT_SDE_UPGRADE_PRIVS stored procedure:

```
connect sys/<password>
exec sdesys_util.grant_sde_upgrade_privs;
```

The upgrade privileges can be revoked and the install privileges granted by executing the SDESYS_UTIL.REVOKE_SDE_UPGRADE_PRIVS stored procedure:

```
connect sys/<password>
exec sdesys_util.revoke_sde_upgrade_privs;
```

Note: To upgrade the ArcSDE database, the SDE user must be granted privileges to create the sequences and triggers in the schemas of users who own tables referenced in the SDE.TABLE_REGISTRY, SDE.LAYERS, or SDE.RASTER_COLUMNS tables. Following the completion of a successful upgrade, the privileges can be revoked and lesser privileges, assigned for the install operation, can be granted.

Creating Oracle users

The privileges that you grant to an ArcSDE Oracle user depend on the user's function within your organization and the type of ArcSDE log files to be used.

Typically the more responsibility a user has, the more access is required. Therefore, users who create data objects must have the privileges required to do so, while users who edit contents of data objects owned by other users require fewer privileges.

If your users create log files and data tables, you will need to grant the following privileges to each user:

```
CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

On the other hand if you intend to have your users borrow their log files from the SDE user's log file pool, you only need to grant your users CREATE SESSION privileges.

The following table lists the privileges required based on a user's responsibilities within the organization.

Title	Description	Privileges

Viewer	The viewer is allowed to connect to an ArcSDE database. Other users grant select privileges on their tables and feature classes to the viewer or to the public role. The DBA can create a role that can be granted select privileges on data objects owned by other users. The role can be granted to the viewer.	CREATE SESSION
		
		SELECT on other user's data objects

Editor	The editor is allowed to connect to an ArcSDE database. Other users grant to the editor select and insert, update, or delete on data objects they own. The DBA may create a role that can be granted select, insert, update, and delete privileges on data objects owned by other users. The role can be granted to the editor.	CREATE SESSION
		
		SELECT, INSERT, UPDATE, or DELETE on other user's data objects

Owner	The owner is allowed to connect to an ArcSDE database and create data objects. The owner may grant privileges on their objects to other users or roles. The DBA may create a role that can be granted select, insert, update, and delete privileges on data objects owned by other users. The role can be granted to the owner.	CREATE SESSION
		CREATE TABLE
		CREATE SEQUENCE
		CREATE TRIGGER
		UNLIMITED TABLESPACE
		
		SELECT, INSERT, UPDATE, or DELETE on other user's objects

Manually creating ArcSDE Oracle users

When you create an Oracle user, assign your users a default tablespace and a temporary tablespace. Permanent objects, such as tables, indexes, and so on, will be stored in the default tablespace unless a tablespace name is specified when the data object is created. The temporary tablespace holds sort runs that exceed the PGA_AGGREGATE_TARGET—the approximate amount of memory allocated to the PGA by the Oracle server.

Warning: If you do not specifically assign a default tablespace to a user, Oracle uses the SYSTEM tablespace by default. Since the SYSTEM tablespace holds the Oracle system tables, a vital part of the database, it is important not to use it for anything else. Since the datasets managed by ArcSDE may be large, using the SYSTEM tablespace to store this data could completely fill the SYSTEM tablespace and cause the Oracle database to crash.

Oracle9*i* allows you to create a default temporary tablespace as a precaution against overusing the SYSTEM tablespace for temporary purposes. Users not specifically assigned a temporary tablespace will store temporary segments in the default temporary tablespace.

This is the Oracle user creation syntax:

```
create user <username>
  identified by <password>
  default tablespace <default tablespace>
  temporary tablespace <temporary tablespace>
  quota unlimited on <default tablespace>;
```

If ArcSDE 8 log files are used, these are the basic privileges that must be granted to all ArcSDE Oracle users until they connect for the first time:

```
CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

Connect as the user to create the standard log file tables, SDE_LOGFILES and SDE_LOGFILE_DATA. To create a user of type VIEWER or EDITOR, revoke the following list of privileges after the SDE log file tables have been created:

```
CREATE TABLE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

For the VIEWER, grant SELECT privileges on the specific data objects you want the user to have permission to display or query. If you have a large community of viewers, create a role that can be granted SELECT access on the data objects.

To grant or revoke SELECT access to complex geodatabase data objects, such as feature datasets or standalone feature classes, you should use ArcCatalog. Use the grant or revoke operation of the sdelayer administration command to grant or revoke SELECT access if you have not installed ArcGIS Desktop. The access can be granted directly to a user or to a role.

For the editor, grant SELECT privileges and editing (insert, update, or delete) privileges on the specific data objects you want the users to have permission to query and edit. If you must create many users who will be editors, you can save time by creating a role and granting the SELECT and edit privileges to the role. Then grant the role to each editor.

In this example, the DBA creates the editor user Sam with the password TREETOP. The default tablespace in which Sam will create his tables and indexes that are not assigned a tablespace is GIS1.

Note: The ArcSDE Administrator creates DBTUNE configuration keywords that hold the tablespace and other storage parameters Oracle assigns to tables and indexes when it creates them. For more information about DBTUNE configuration keywords and how to create and use them, see Chapter 3, ‘Configuring DBTUNE storage parameters’.

The temporary tablespace used when Sam creates indexes and performs sorts is TEMP1:

```
create user sam
  identified by treetop
  default tablespace gis1
  temporary tablespace temp1
  quota unlimited on gis1;
```

To create Sam’s standard log file tables (SDE_LOGFILES and SDE_LOGFILE_DATA) and their associated sequence generator and trigger, the DBA grants the user Sam the following privileges:

```
CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE TRIGGER
UNLIMITED TABLESPACE
```

The DBA connects to the ArcSDE database as user Sam to create the log file tables in user Sam's schema. The sdelayer administration command "describe" operation provides a simple way to do this:

```
$ sdelayer -o describe -u sam -p treetop
```

Use the REVOKE <privilege> FROM <user> command to revoke all but the CREATE SESSION privileges, since Sam is an editor and is not allowed to create a schema:

```
REVOKE CREATE TABLE FROM SAM;
REVOKE CREATE PROCEDURE FROM SAM;
REVOKE CREATE SEQUENCE FROM SAM;
REVOKE CREATE TRIGGER FROM SAM;
REVOKE UNLIMITED TABLESPACE FROM SAM;
```

Create SDE_VIEWER and SDE_EDITOR roles. The DBA has determined that all users who own schema should grant SELECT privileges on their data objects to the SDE_VIEWER role, and that they should grant INSERT, UPDATE, and DELETE to the SDE_EDITOR role:

```
CREATE ROLE SDE_VIEWER;
CREATE ROLE SDE_EDITOR;
```

The DBA grants both the SDE_VIEWER and SDE_EDITOR roles to Sam:

```
GRANT SDE_VIEWER TO SAM;
GRANT SDE_EDITOR TO SAM;
```

Betty, a creator, uses the grant operation of the sdelayer command to grant select access on her roads feature class to the SDE_VIEWER role. All viewers granted the SDE_VIEWER role will have SELECT access to Betty's roads feature class:

```
$ sdelayer -o grant -l roads,feature -A select -U sde_viewer -u betty -p
cdn
```

Betty also grants INSERT, UPDATE, and DELETE access to the SDE_EDITORS role. Now editors granted the SDE_EDITORS role, such as Sam, will be able to make changes to Betty's roads feature class:

```
$ sdelayer -o grant -l roads,feature -A insert -U sde_editors -u betty -p
cdn
$ sdelayer -o grant -l roads,feature -A update -U sde_editors -u betty -p
cdn
$ sdelayer -o grant -l roads,feature -A delete -U sde_editors -u betty -p
cdn
```

Installing the USER_UTIL package to create and maintain ArcSDE users

As an alternative to executing the SQL statements to create and maintain ArcSDE users, you can install the user_util package of stored procedures located in the SDEHOME/tools/oracle directory. To install the package, connect as an Oracle DBA (usually the Oracle system user) and run the package specification script followed by the body script:

```
connect system/<password>
@user_util.sps
```

```
Package created

@user_util.spb

Package created
```

Using the stored procedures of the USER_UTIL package

The CREATE_USER_AND_LOGFILES stored procedure creates an Oracle user with standard log files and grants privileges according to its user type. The usage for this stored procedure is as follows:

```
USER_UTIL.CREATE_USER_AND_LOGFILES (
    username IN VARCHAR2(30),
    password IN VARCHAR2(30),
    default_tabsp IN VARCHAR2(30) DEFAULT 'USERS',
    temp_tabsp IN VARCHAR2(30) DEFAULT 'TEMP',
    usertype IN VARCHAR2(8) DEFAULT 'OPERATOR'
)
```

The username and password must be entered. The user's default tablespace defaults to USERS tablespace, while the temporary tablespace defaults to the TEMP tablespace. The usertype argument defines the type of user that will be created, and it defaults to OPERATOR.

A user created as an OPERATOR and granted CREATE SESSION privilege is not allowed to create a schema and does not have access to any data objects owned by other users.

A user created as an OWNER is granted privileges to create tables and granted CREATE SCHEMA privileges. OWNER users automatically grant SELECT, INSERT, UPDATE, or DELETE privileges on their own tables to other OWNER users and to OPERATOR users.

Updating the SDE.DBTUNE table

After you have created the tablespaces for your tables and indexes, you should update the SDE.DBTUNE table. The SDE.DBTUNE table is updated by editing a DBTUNE file, which must be located under the $SDEHOME/etc directory, and importing the file into the SDE.DBTUNE table. The default DBTUNE file is called dbtune.sde. This file is imported into the SDE.DBTUNE table during the SDE setup that occurs when you run sdesetupora*.

Create the keywords in the DBTUNE file that will contain the Oracle configuration parameters of the feature classes and tables that you intend to create with ArcCatalog or the ArcSDE administration tools.

For a detailed discussion on the maintenance of the DBTUNE table, refer to Chapter 3, 'Configuring DBTUNE storage parameters'. This chapter describes DBTUNE parameters that can be applied to each of the spatial storage methods supported by ArcSDE for Oracle. The supported spatial storage formats are described in Appendixes C and D.

Creating tables, feature classes, and raster columns

ArcGIS Desktop offers several ways to create and maintain the tables, indexes, and feature classes of an ArcSDE database. Chapter 4, 'Managing tables, feature classes, and raster columns', describes in detail the possible methods for creating tables and feature classes using the ArcGIS Desktop.

Setting the Oracle initialization parameters

Whenever you start an Oracle instance, Oracle reads its initialization parameters from either the init.ora file or from the server parameter file, spfile.ora. Both of these files define the characteristics of the instance, but they are managed differently.

Changing parameters using the ALTER SYSTEM command will automatically be reflected in the server parameter file if the instance was started by that method. If the instance was started using an init.ora file, you will have to manually edit the file with a text editor if you want changes to system parameters to affect more than the current instance of the database.

This section describes some of the parameters that control allocation of shared memory. For a detailed discussion of the Oracle initialization parameters, refer to *Oracle Server Tuning* for your Oracle release.

The init.ora file is located under the $ORACLE_BASE/admin/<ORACLE_SID>/pfile directory or folder. init.ora is a common name given to the initialization file of an Oracle database instance, but for any given instance, the file is actually called init<oracle SID>.ora. For example, if the Oracle System ID (SID) is GIS, the init.ora file for this instance would be called initGIS.ora.

Managing Oracle's memory

Care must be taken when setting the initialization parameters that affect memory. Setting these parameters beyond the limits imposed by the physical memory resource of the host machine significantly degrades performance. This section provides a few general rules regarding configuration of System Global Area (SGA) as well as memory structures affecting the size of an Oracle user's private area, the PGA. The SGA is a block of shared memory that Oracle allocates and shares with all sessions. For more information about the SGA, refer to the *Oracle Concepts Guide* for your Oracle release.

SGA must not swap

You should not create an SGA that is larger than two-thirds the size of your server's physical RAM. Your virtual memory must be able to accommodate both the SGA and the requirements of all active processes on the server.

Avoid excessive paging

Using your operating system tools (vmstat on UNIX systems and the Task Manager on Windows), check for excessive paging. A high degree of paging can be the result of an SGA that is too large.

Configure enough virtual memory

As a rule, Oracle recommends that your swap space be at least three to four times the size of your physical RAM. The required size of the swap file on UNIX or the page file on Windows depends on the number of active ArcSDE sessions.

For every ArcSDE application server session, a gsrvr process and a corresponding Oracle process are started. To determine the memory usage of these processes, use the ps -elf command on UNIX systems and the Processes tab of the Windows Task Manager. You must deduct the size of the Oracle SGA from the Oracle user processes. The total size of the ArcSDE gsrvr processes, the ArcSDE giomgr processes, Oracle user processes, Oracle background processes, operating system processes, and any other process running on the server must be able to fit into virtual memory.

For ArcSDE client applications that connected directly to an Oracle instance, the gsrvr process does not exist. Also, if the ArcSDE service is not used because all client applications connect directly to the Oracle instance, the giomgr process will not be started either. For this reason direct connections have a smaller memory imprint on the server since the gsrvr process is absent.

Redo log buffer

The redo log buffer is a component of the Oracle SGA that holds uncommitted changes to the database. The log buffer is flushed to the current online redo log file every three seconds or whenever a user issues a commit and the buffer becomes one-third full. The size of the redo log buffer is controlled by the LOG_BUFFER parameter. Because of the rapid rate at which the log buffer is flushed, it does not need to be that large.

Oracle recommends that you set this parameter to 512 KB or 128 KB multiplied by the number of CPUs. If you have less than four CPUs, set this parameter to 512 KB.

```
log_buffer = 524288
```

Otherwise, set LOG_BUFFER to 128 KB times the number of CPUs.

Setting the LOG_BUFFER to a large value to process huge loading transactions may, in fact, result in a performance reduction. Latch contention between transactions may occur if the log buffer is set too large.

To determine if the redo log buffer is large enough while the system is active, examine the Oracle dynamic table V$SYSSTAT. Compare the values of the *redo log space requests* and the *redo entries.* Oracle recommends that you increase the size of the redo log buffers if the ratio of these values is greater than 1:5,000.

```
select name, value
from v$sysstat
where name in ( 'redo entries' , 'redo log space requests' );
```

Shared pool

The shared pool is another component of the Oracle SGA that holds both the data dictionary cache and the library cache. The data dictionary cache holds information about data objects, free space, and privileges. The library cache holds the most recently parsed SQL statements.

Generally, if the shared pool is large enough to satisfy the resource requirements of the library cache, it is already large enough to hold the data dictionary cache. The size of the shared pool is controlled by the SHARED_POOL_SIZE parameter. At Oracle9*i*, you must allocate shared pool space in multiples of a granule, which is 4M, 8M, or 16M depending on the size of the SGA and the operating system. ESRI recommends that you set the SHARED_POOL_SIZE parameter to a multiple of 16M to accommodate any system ESRI supports and that you set this parameter to at least 128 MB:

```
shared_pool_size = 128,000,000
```

Highly active geodatabases supporting volatile utility or parcel editing systems may require the SHARED_POOL_SIZE to be set as high as 200 MB.

Of the three SGA buffers, the shared pool is the most important. If the SGA is already as large as it can be, given the size of your physical memory, reduce the size of the buffer cache to accommodate a larger shared pool.

Buffer cache

The buffer cache is another component of the Oracle SGA that stores the most recently used data blocks. Data blocks are the Oracle atomic unit of data transfer. Oracle reads and writes data blocks to and from the database whenever the user edits or queries it. The size of the buffer cache is controlled by the DB_CACHE_SIZE parameter.

For optimum performance, increase the size of the buffer cache without causing the operating system to page excessively and/or swap the SGA. Oracle recommends that the SGA not be larger than two-thirds of the physical RAM.

To estimate the size of the buffer cache, first determine how much physical RAM your server has. Multiply this number by 0.66 to determine the target size of the SGA. Deduct the SHARED_POOL_SIZE and LOG_BUFFER to return the amount of memory available to the buffer cache. Reduce this number by 10 percent to account for Oracle’s internal memory usage. Finally, divide by the database block size to determine the DB_BLOCK_BUFFERS setting. (The recommended minimum data block size for ArcSDE is at least 16 KB.)

```
memory available to SGA = physical RAM * 2/3

memory available to buffer cache
= (memory available to SGA - (shared_pool_size + log_buffer)) * 0.9

db_block_buffers

= memory available to buffer cache / db_block_size
```

Allocate space for the PGA

Allocate space for the program global area of the Oracle server processes. This space is typically used as a temporary buffer for sorting and merging data during a table join. Set the WORKAREA_SIZE_POLICY to AUTO, then initially set the PGA_AGGREGATE_TARGET to the total physical RAM multiplied by 0.16. Once the application has been in use for some time, tune the PGA_AGGREGATE_TARGET according to the procedure outlined in the *Oracle Performance Tuning Guide and Reference*.

```
workarea_size_policy = auto
pga_aggregate_target = <total physical RAM * 0.16)
```

Prepage the SGA

Set the PRE_PAGE_SGA to true for servers that are running a single Oracle instance. Prepaging the SGA increases the amount of time required to start the Oracle server as well as the amount of time required for users to connect. However, it does reduce the number of page faults—which occur whenever Oracle must allocate another page to the SGA—while the server is active:

```
pre_page_sga = true
```

Enabling the optional Oracle startup trigger

ArcSDE includes an optional startup trigger that is run whenever the Oracle instance is started. The startup trigger cleans up any orphaned session information that remains in the ArcSDE system tables following an instance failure. The startup trigger is optional because ArcSDE invariably cleans up the orphaned metadata during its normal operation. The startup trigger merely offers a guarantee that orphaned metadata will not be present following the Oracle instance startup.

To create the startup trigger, run the SQL arcsde_database_startup.sql script as the Oracle SYS user. The script is located at $SDEHOME/tools/oracle on UNIX systems and %SDEHOME%\tools\oracle on Windows.

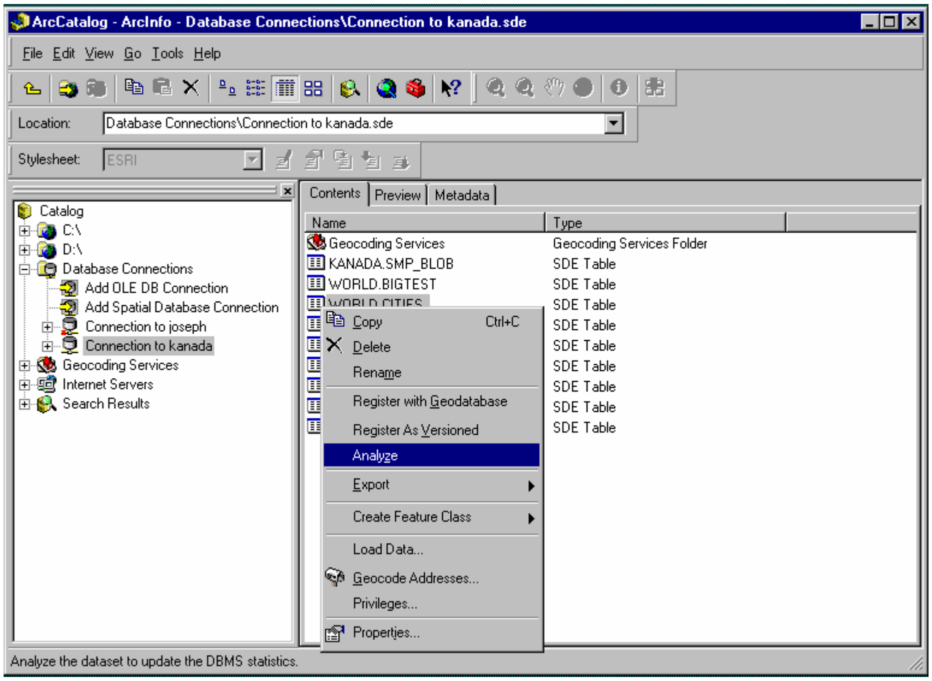
Updating Oracle statistics

The ArcSDE server has been built to work optimally with the cost-based optimizer, though certain statements have been tuned using optimizer hints. As a result, you should set the init.ora initialization variable OPTIMIZER_MODE to CHOOSE, allowing the optimizer to choose the cost-based approach if you have computed statistics.

The ArcSDE software has been developed as an OLTP system. As such, Oracle initialization parameters geared toward optimizing data warehousing queries will adversely affect the performance of ArcSDE queries and should not be used. The START_TRANSFORMATION_ENABLED parameter should be set to FALSE. You should create the Oracle database using an OLTP template and not a data warehousing template. You should not enable parallel execution on the ArcSDE schema.

Updating ArcSDE compressed binary statistics

For optimal performance of feature classes created with the ArcSDE compressed binary storage format, keep the statistics up-to-date.



Location: Database Connections\Connection to kanada.sde

Name	Type
Geocoding Services	Geocoding Services Folder
KANADA.SMP_BLOB	SDE Table
WORLD.BIGTEST	SDE Table
WORLD.CITIES	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table
	SDE Table

Analyze the dataset to update the DBMS statistics.

In ArcCatalog, to update the statistics of all of the tables and indexes within a feature dataset, right-click the feature dataset and click Analyze. To update the tables and indexes within a feature class, right-click the feature and click Analyze.

From the command line, use the update_dbms_stats operation of the sdetable administration command to update the statistics for all the tables and indexes of a feature class. It is better to use the update_dbms_stats operation rather than individually analyzing the tables with the Oracle SQL ANALYZE statement because it updates the statistics for all the tables of a feature class that require statistics. To have the update_dbms_stats operation update the statistics for all the required tables, do not specify the -K (schema object) option.

```
sdetable -o update_dbms_stats -t roads -m compute -u av -p mo
```

When the feature class is registered as multiversioned, the adds and deletes tables are created to hold the business table’s added and deleted records. The version registration process automatically updates the statistics for all the required tables at the time it is registered.

Periodically update the statistics of dynamic tables and indexes to ensure that the Oracle cost-based optimizer continues to choose an optimum execution plan. To save time, you can analyze all the data objects within a feature dataset in ArcCatalog.

If you do not have enough temporary tablespace to compute statistics on larger tables, use the -m option to estimate statistics. The tables will be estimated at a sample rate of 33 percent.

```
sdetable -o update_dbms_stats -t roads -m estimate -u av -p mo
```

You should consider increasing the size of your temporary tablespace to compute statistics rather than estimate them, as it provides more accurate statistics for the Oracle cost-based optimizer.

The statistics of a table's indexes are automatically computed when the table is analyzed, so there is no need to analyze the indexes separately. However, if you need to do so you can use the UPDATE_DBMS_STATS -n option with the index name.

The example below illustrates how the statistics for the f2_ix1 index of the roads feature table can be updated:

```
sdetable -o update_dbms_stats -t roads -K f -n F2_IX1 -u av -p mo
```

For more information on analyzing geodatabase objects from ArcCatalog, refer to *Building a Geodatabase*.

For more information on the sdetable administration command and the UPDATE_DBMS_STATS operation, refer to the *ArcSDE Developer Help*.

Updating Oracle Spatial geometry type statistics

Update the statistics of the business table containing an Oracle Spatial column with Analyze from ArcCatalog or with the sdetable UPDATE_DBMS_STATS operation.

```
sdetable -o update_dbms_stats -t roads -m compute -u av -p mo
```

You may also update the statistics using the Oracle ANALYZE statement. Use the compute statistics option if there is enough temporary tablespace available to permit the operation. Otherwise, use the estimate statistics option with as high a sample rate as possible.

```
analyze table roads compute statistics;
```

When ArcSDE creates data using Oracle Spatial geometry, it can optionally create the layer's Oracle Spatial index. Oracle Spatial suggests that you analyze this spatial index. If ArcSDE is asked to create the Oracle Spatial index for you, it automatically analyzes this index. If you decide to create the Oracle Spatial index on your own, you will have to use the Oracle ANALYZE statement to analyze your index after creating it.

For more information on the Oracle ANALYZE statement, refer to the *Oracle SQL Reference Manual* for your release of Oracle.

Note: For Oracle 10g, Oracle recommends DBMS_STATS for collecting optimizer statistics. Please consult your Oracle documentation for more information.

CHAPTER 3

Configuring DBTUNE storage parameters

The DBTUNE storage parameters are stored in the DBTUNE table. The DBTUNE table, along with all other metadata tables, is created during the setup phase that follows the installation of the ArcSDE software. The ArcSDE software install creates a DBTUNE file under the etc directory from which the DBTUNE table is populated. If no DBTUNE file is present during setup, ArcSDE will populate the DBTUNE table with default values.

DBTUNE storage parameters allow you to control how ArcSDE clients create objects within an Oracle database. They allow you to determine such things as how to allocate space to a table or index, which tablespace a table or index is created in, as well as other Oracle-specific storage attributes. They also allow you to specify one of the available storage formats for the geometry of a spatial column.

This chapter discusses the mechanism by which ArcSDE manages storage parameters that you provide and how ArcSDE applies them to specific statements submitted to Oracle when creating ArcSDE tables, indexes, and other objects in an Oracle database.

Many DBTUNE parameters include corresponding Oracle storage parameters. For detailed information on how Oracle uses these storage parameters to control space allocation, refer to the *Oracle Server Administrator's Guide* for your Oracle release.

The DBTUNE table

The DBTUNE storage parameters are maintained in the database in the DBTUNE metadata table. The DBTUNE table, along with all other metadata tables, is created during the setup phase that follows the installation of the ArcSDE software.

The DBTUNE table is populated with specific default values, but these values may be changed. Several example versions of the DBTUNE file are provided with the installation media for ArcSDE.

DBTUNE parameters

Parameters define the storage configuration of simple objects, such as tables and indexes, as well as complex objects such as feature classes, network classes, and raster columns. Many different parameters may be grouped together under a single *configuration keyword*.

ArcSDE client applications and some ArcSDE administration tools reference one or more configuration keywords when creating an object.

When a configuration keyword is specified by an ArcSDE application or administration tool, the parameters within the associated *parameter group* are searched, and the necessary configuration strings are incorporated into the CREATE TABLE or CREATE INDEX statement submitted to the Oracle database server.

The structure of the DBTUNE file

Storage parameters in a DBTUNE file occur as a combination of *parameter name* and *configuration string* delimited by white space. A configuration string value may span multiple lines and must be enclosed in double quotes. For example, a valid specification for the parameter named A_INDEX_ROWID might look like this:

```
A_INDEX_ROWID   "PCTFREE 0 TABLESPACE GIS1 INITRANS 4 STORAGE (INITIAL
4M)"
```

Storage parameters are grouped by *keyword*. Each parameter group is introduced by its keyword, which is prefixed by two pound signs, “##”. A line beginning with the word “END” terminates each parameter group. Double pound signs, “##”, signal the presence of a keyword but are not part of the keyword itself.

For example, a group of parameters under the configuration keyword “WILSON_DATA” may look like this:

```
##WILSON_DATA
```

```
ATTRIBUTE_BINARY        "LONGRAW"
A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 TABLESPACE WILSON_INDEXES STORAGE
                 (INITIAL 4M)"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 TABLESPACE WILSON_INDEXES STORAGE
                 (INITIAL 4M)"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 TABLESPACE WILSON_INDEXES STORAGE
                 (INITIAL 4M)"
B_STORAGE       "PCTFREE 0 INITRANS 1 TABLESPACE WILSON_DATA STORAGE
                 (INITIAL 8M)"
END
```

In special circumstances, ArcSDE references a *compound keyword* when creating database objects. The compound keyword allows ArcSDE to create related database objects having different object creation parameters to accommodate different performance needs. A compound keyword consists of a configuration keyword plus a suffix delimited by a double colon, “::”. For example:

```
##ELECTRIC::DESC
```

Comments within the DBTUNE file are indicated by a single pound sign, “#”. Default versions of the DBTUNE file provided in the general software release contain lines that are commented out. Such lines are used as placeholders for certain storage parameters, such as tablespace name, and may be restored by removing the comment character and editing the line.

Any number of parameter groups may be specified in a DBTUNE file. However, certain groups and certain parameter names within groups are expected to exist and will be created in the DBTUNE table if they do not exist in the DBTUNE file.

The structure of the DBTUNE table

Parameters from a DBTUNE file are loaded into the DBTUNE table using the sdedbtune utility. The DBTUNE table has the following definition:

Name	Null?	Datatype
keyword	not null	varchar2(32)
parameter_name	not null	varchar2(32)
config_string	null	varchar2(2048)

The keyword field stores the configuration keyword for the group in which each parameter is found. For a single keyword, there may be many different parameter_name values, each one associated with a config_string value.

After creating the DBTUNE table, the setup phase of the ArcSDE installation populates the table with the contents of the dbtune.sde file, which it expects to find in the %SDEHOME%/etc directory under Windows or the $SDEHOME/etc under UNIX. If the DBTUNE table already exists, the ArcSDE setup phase will not alter its contents.

Managing the DBTUNE table

Through the use of the sdedbtune utility, you can initialize or alter the contents of the DBTUNE table. This utility guarantees that the DBTUNE table maintains a certain default set of keywords, parameters, and parameter values.

In addition to the default keywords and parameters, you may add your choice of keywords and configuration values to the DBTUNE file.

Note: ESRI does *not* recommend using SQL to directly alter the contents of the DBTUNE table. Doing so would bypass certain protections written into the sdedbtune utility, possibly leading to reduced performance.

Initializing the DBTUNE table

The dbtune.sde file that is provided with the install media contains default values, which are used to initialize the DBTUNE table.

On UNIX systems, you can modify the dbtune.sde file prior to running the sdesetupora* command. On Windows systems, the setup phase is part of the install, so you will have to edit the file and use the sdedbtune import operation to customize the DBTUNE table.

If the dbtune.sde file is missing when the sdesetupora* command is executed, or if specific parameters are missing because the dbtune.sde file has been altered, the ArcSDE software will enter *software default* values into the DBTUNE table.

Customizing the DBTUNE file

Prior to creating ArcSDE objects, you should customize the dbtune.sde file by specifying the tablespace names for storage parameters. In the default dbtune.sde file, the tablespace entries in the dbtune.sde file have been commented out with the “#” character.

To customize the dbtune.sde file, remove the comment character preceding each tablespace specification and enter the names of the tablespaces where you want to store your ArcSDE tables and indexes. Be careful not to remove the double quotation marks that surround the configuration strings.

Follow the procedure provided in the following example for updating the Oracle tablespace parameter in the dbtune.sde file.

The DEFAULTS configuration keyword in the dbtune.sde file contains the B_STORAGE storage parameter with the Oracle tablespace parameter commented out:

```
##DEFAULTS

GEOMETRY_STORAGE    "SDEBINARY"
ATTRIBUTE_BINARY    "LONGRAW"
B_STORAGE           "PCTFREE 0 INITRANS 4"
#                    TABLESPACE   <default business table tablespace name>
```

Edit the dbtune.sde file, remove the "#" comment character, and enter the name of the tablespace you want to store business tables in by default:

```
##DEFAULTS

GEOMETRY_STORAGE    "SDEBINARY"
ATTRIBUTE_BINARY    "LONGRAW"
B_STORAGE           "PCTFREE 0 INITRANS 4
                     TABLESPACE   ROADS"
```

When the setup program loads, your customized DBTUNE configuration keywords and storage parameters are written into the DBTUNE table.

Editing the DBTUNE table

If you need to change the contents of the DBTUNE table after it is loaded, you should use the sdedbtune utility and follow these steps.

1. Export the DBTUNE table to a text file using the sdedbtune –o export command.

2. Edit the resulting file with a UNIX file-based editor, such as vi or a Windows file-based editor such as Notepad.

3. Import the edited file to the DBTUNE table using the sdedbtune –o import command.

In the following example, the DBTUNE table is exported to the file dbtune.out. Then, the file is edited with the UNIX "vi" file-based editor.

```
$ sdedbtune -o export -f dbtune.out -u sde -p sdepasswd


ArcSDE   9               Wed Oct  4 22:32:44 PDT 2003
Attribute       Administration Utility
-----------------------------------------------------

        Successfully exported to file SDEHOME\etc\dbtune.out

$ vi dbtune.out

$ sdedbtune -o import -f dbtune.out -u sde -p sdepasswd -N
```

```
ArcSDE   9               Wed Oct  4 22:32:44 PDT 2003
Attribute       Administration Utility
-----------------------------------------------------

        Successfully imported from file SDEHOME\etc\dbtune.out
```

The sdedbtune administration tool always exports and imports from the $SDEHOME/etc directory. You cannot specify that files should be located in another directory. By not allowing the relocation of the file, the sdedbtune command ensures the DBTUNE parameters remain under the ownership of the ArcSDE administrator.

Adding keywords to the DBTUNE table

You may add parameter groups to the DBTUNE table for any special purpose. For instance, you may want to create certain feature classes in a newly created tablespace that is segregated from the rest of the data.

To add keywords, follow the instructions above for editing the DBTUNE table. When you edit the export file, it is often a good idea to create a new parameter group as a cut and paste copy of an existing parameter group in order to avoid introducing syntax errors. You may then edit the configuration keyword and any of the strings to desired new values before saving the DBTUNE file and importing it back into the DBTUNE table.

Using the DBTUNE table

At its most basic level, the DBTUNE table provides configuration strings that ArcSDE appends to a CREATE TABLE or CREATE INDEX statement in SQL. Therefore, the configuration strings specify storage parameters that must be considered valid by the Oracle server.

Choosing the configuration string

The choice of configuration strings by an ArcSDE application depends on the operation being performed and the type of object on which it is being performed, as well as the configuration keyword. For example, if the type of operation is CREATE TABLE and the type of table being created is a business table, the parameter_name of B_STORAGE will be used to determine the configuration string.

The ArcSDE application then searches the DBTUNE table for a configuration keyword that matches the one entered and uses the configuration string of the appropriate storage parameter.

If the application cannot find the requested storage parameter within the specified parameter group, it searches the DEFAULT parameter group. If the requested parameter group cannot be located within the DEFAULT parameter group, ArcSDE uses the Oracle defaults to create the table or index.

Table parameters

Table parameters define the storage configuration of an Oracle table. ArcSDE appends the configuration string associated with the parameter to the CREATE TABLE statement prior to submitting the statement to Oracle.

Valid entries for an ArcSDE table include any parameter allowable to the right of the column list in the CREATE TABLE statement, including the TABLESPACE and STORAGE clauses.

For example, if you define the B_STORAGE parameter in this manner:

```
B_STORAGE "tablespace roads_tabsp storage (initial 10M) initrans 4 pctfree
10"
```

ArcSDE would execute the following Oracle CREATE TABLE statement:

```
CREATE TABLE roads (road_id integer, name varchar2(32), surface_code
integer)
tablespace roads_tabsp storage (initial 10M) initrans 4 pctfree 0;
```

Index parameters

Index parameters define the storage configuration of an Oracle index. ArcSDE appends the index parameter to an Oracle CREATE INDEX statement prior to submitting the statement to Oracle.

Valid entries in an ArcSDE index parameter include any parameter allowable by the Oracle server to the right of the column list of the CREATE INDEX statement, especially the TABLESPACE and STORAGE clauses.

For example, if you specify the B_INDEX_USER parameter in this manner:

```
B_INDEX_USER "tablespace roads_idx_tabsp storage (initial 1M) initrans 4
pctfree 10 nologging"
```

ArcSDE would assemble a CREATE INDEX statement like this:

```
CREATE INDEX roads_idx on roads (road_id)
```

```
tablespace roads_idx_tabsp storage (initial 1M) initrans 4 pctfree 10
nologging;
```

Note: ESRI recommends that you create your indexes with NOLOGGING. Doing so will avoid logging the changes made to the indexes in the Oracle redo log files. Although the index cannot be recovered from the archive log in the event that you should lose the data file the index is stored in, you can easily re-create the index using the ALTER INDEX <index_name> REBUILD command. Therefore, ESRI believes that the ease at which an index may be regenerated outweighs the need to log the changes in the event of a disk failure.

Defining the storage parameters

Configuration keywords may include any combination of three basic types of storage parameters: metaparameters, table parameters, and index parameters.

Metaparameters define the way certain types of data will be stored, the environment of a configuration keyword, or a comment that describes the configuration keyword. Table and index parameters establish the storage characteristics of, respectively, tables and indexes.

The business table storage parameter

A business table is any Oracle table created by an ArcSDE client, the sdetable administration command, or the ArcSDE C application programming interface (API) SE_table_create function.

Use the DBTUNE table's B_STORAGE parameter to define the storage configuration of a business table.

The business table index storage parameters

Three index storage parameters exist to support the creation of business table indexes.

The B_INDEX_USER storage parameter holds the storage configuration for user-defined indexes created with the C API function SE_table_create_index and the create_index operation of the sdetable command.

The B_INDEX_ROWID storage parameter holds the storage configuration of the index that ArcSDE creates on a register table's object ID column, commonly referred to as the ROWID.

Note: ArcSDE registers all tables that it creates. Tables not created by ArcSDE can also be registered with the sdetable –o alter_reg command or with ArcCatalog. The SDE.TABLE_REGISTRY system table maintains a list of the currently registered tables.

The B_INDEX_SHAPE storage parameter holds the storage configuration of the spatial column index that ArcSDE creates when a spatial column is added to a business table. This index is created by the ArcSDE C API function SE_layer_create. This function is called by ArcGIS® when it creates a feature class and by the add operation of the sdelayer command.

The B_INDEX_RASTER storage parameter holds the storage configuration of the raster column index that ArcSDE creates when a raster column is added to a business table. This index is created by the ArcSDE C API function SE_rastercolumn_create. This function is called by ArcGIS when it creates a feature class and by the add, copy, and import operations of the sderaster command.

Multiversioned table storage parameters

Registering a business table as multiversioned allows multiple users to maintain and edit their copy of the object. At appropriate intervals, users merge the changes made to their copy with the changes made by other users and reconcile any conflicts that arise when the same rows are modified.

ArcSDE creates two tables—the adds table and the deletes table—for each table that is registered as multiversioned.

The A_STORAGE parameter maintains the storage configuration of the adds table. Four other storage parameters hold the storage configuration of the indexes of the adds table. The adds table is named A<n>, where <n> is the registration ID listed in the SDE.TABLE_REGISTRY system table. For instance, if the business table ROADS is listed with a registration ID of 10, ArcSDE creates the adds table as A10.

The A_INDEX_ROWID storage parameter holds the storage configuration of the index that ArcSDE creates on the multiversion object ID column, commonly referred to as the ROWID. The adds table ROWID index is named A<n>_ROWID_IX1, where <n> is the business table's registration ID, which the adds table is associated with.

The A_INDEX_STATEID storage parameter holds the storage configuration of the index that ArcSDE creates on the adds table's SDE_STATE_ID column. The

SDE_STATE_ID column index is called A<n>_STATE_IX2, where <n> is the business table's registration ID, which the adds table is associated with.

The A_INDEX_SHAPE storage parameter holds the storage configuration of the index that ArcSDE creates on the adds table's spatial column. If the business table contains a spatial column, the column and the index on it are duplicated in the adds table. The adds table's spatial column index is called A<n>_IX1_A, where <n> is the layer ID of the feature class as it is listed in the SDE.LAYERS table.

The A_INDEX_USER storage parameter holds the storage configuration of user-defined indexes that ArcSDE creates on the adds table. The user-defined indexes on the business tables are duplicated on the adds table.

The D_STORAGE parameter holds the storage configuration of the deletes table. Two other storage parameters hold the storage configuration of the indexes that ArcSDE creates on the deletes table. The deletes table is named D<n>, where <n> is the registration ID listed in the SDE.TABLE_REGISTRY system table. For instance, if the business table ROADS is listed with a registration ID of 10, ArcSDE creates the deletes table as D10.

The D_INDEX_STATE_ROWID storage parameter holds the storage configuration of the D<n>_IDX1 index that ArcSDE creates on the deletes table's SDE_STATE_ID and SDE_DELETES_ROW_ID columns.

The D_INDEX_DELETED_AT storage parameter holds the storage configuration of the D<n>_IDX2 index that ArcSDE creates on the deletes table's SDE_DELETED_AT column.

Note: If a configuration keyword is not specified when the registration of a business table is converted from single version to multiversion, the adds and deletes tables and their indexes are created with the storage parameters of the configuration keyword the business table was created with.

Feature class storage parameters

A feature class created with an ArcSDE compressed binary storage (LONG RAW or BLOB data type) format adds two tables to the Oracle database—the feature table and the spatial index table. Three indexes are created on the feature table, and two indexes are created on the spatial index table. The storage parameters for these tables and indexes follow the same pattern as the B_STORAGE and B_INDEX_* storage parameters of the business table.

The F_STORAGE parameter holds the Oracle CREATE TABLE storage configuration string of the feature table. The feature table is created as F_<n>, where <n> is the layer ID of the table's feature class as found in the SDE.LAYERS table.

The F_INDEX_FID storage parameter holds the Oracle CREATE INDEX storage configuration string of the feature table's spatial column index. The spatial column is created as F<n>_UK1, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The F_INDEX_AREA storage parameter holds the Oracle CREATE INDEX storage configuration of the feature table's area column index. The spatial column is created as F<n>_AREA_IX2, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The F_INDEX_LEN storage parameter holds the Oracle CREATE INDEX storage configuration of the feature table's length column index. The spatial column is created as F<n>_LEN_IX3, where <n> is the layer ID of the index's feature class as found in the SDE.LAYERS table.

The S_STORAGE parameter holds the Oracle CREATE TABLE storage configuration of the spatial index table. The spatial index table is created as S_<n>, where <n> is the layer ID of the spatial index table's feature class as found in the SDE.LAYERS table.

The S_INDEX_ALL storage parameter holds the Oracle CREATE INDEX storage configuration of the spatial table first index. The spatial index table is created as S_<n>_IX1, where <n> is the layer ID of the index's feature class found in the SDE.LAYERS table.

The S_INDEX_SP_FID storage parameter holds the Oracle CREATE INDEX storage configuration of the spatial table second index. The spatial index table is created as S_<n>_IX2, where <n> is the layer ID of the index's feature class found in the SDE.LAYERS table.

Raster table storage parameters

A raster column added to a business table is actually a foreign key reference to raster data stored in a schema consisting of four tables and five supporting indexes.

The RAS_STORAGE parameter holds the Oracle CREATE TABLE storage configuration of the RAS table.

The RAS_INDEX_ID parameter holds the Oracle CREATE TABLE storage configuration of the RAS table index.

The BND_STORAGE parameter holds the Oracle CREATE TABLE storage configuration of the BND table index.

The BND_INDEX_ COMPOSITE parameter holds the Oracle CREATE INDEX storage configuration of the BND table's composite column index.

The BND_INDEX_ID storage holds the Oracle CREATE INDEX storage configuration of the BND table's row ID (RID) column index.

The AUX_STORAGE parameter holds the Oracle CREATE TABLE storage configuration of the AUX table.

The AUX_INDEX_COMPOSITE parameter holds the Oracle CREATE INDEX storage configuration of the AUX table's index.

The BLK_STORAGE parameter holds the Oracle CREATE TABLE storage configuration of the BLK table.

The BLK_INDEX_COMPOSITE parameter holds the Oracle CREATE TABLE storage configuration of the BLK table's index.

Arranging storage parameters by keyword

Storage parameters of the DBTUNE table are grouped by keyword. The following keywords are present by default in the DBTUNE table:

- DEFAULTS
- DATA_DICTIONARY
- IMS_METADATARELATIONSHIPS
- IMS_METADATA
- IMS_METADATATAGS
- IMS_METADATATHUMBNAILS
- IMS_METADATAUSERS
- IMS_METADATAVALUES
- IMS_METADATAWORDINDEX
- IMS_METADATAWORDS
- LOGFILE_DEFAULTS

- NETWORK_DEFAULTS
- NETWORK_DEFAULTS::DESC
- NETWORK_DEFAULTS::NETWORK
- SURVEY_MULTI_BINARY
- TOPOLOGY_DEFAULTS
- TOPOLOGY_DEFAULTS::DIRTYAREAS

DEFAULTS keyword

Each DBTUNE table has a fully populated DEFAULTS configuration keyword.

The DEFAULTS configuration keyword can be selected whenever you create a table, index, feature class, or raster column. If you do not choose a keyword for one of these objects, the DEFAULTS configuration keyword is used. If you do not include a storage parameter in a configuration keyword that you have defined, ArcSDE uses the storage parameter from the DEFAULTS configuration keyword.

The DEFAULTS configuration keyword relieves you of the need to define all the storage parameters for each of your configuration keywords. The storage parameters of the DEFAULTS configuration keyword should be populated with values that represent the average storage configuration of your data.

During installation, if the ArcSDE software detects a missing DEFAULTS configuration keyword storage parameter in the dbtune.sde file, it automatically adds the storage parameter. If you import a DBTUNE file with the sdedbtune command, the command automatically adds default storage parameters that are missing. ArcSDE will detect the presence of the following list of storage parameters and insert the storage parameter and the default configuration string.

```
##DEFAULTS

ATTRIBUTE_BINARY        "LONGRAW"
AUX_INDEX_COMPOSITE "PCTFREE 10 INITRANS 4 NOLOGGING"
AUX_STORAGE     "PCTFREE 10 INITRANS 4"
A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
A_STORAGE       "PCTFREE 0  INITRANS 4"
BLK_INDEX_COMPOSITE "PCTFREE 10 INITRANS 4 NOLOGGING"
BLK_STORAGE     "PCTFREE 0 INITRANS 4"
BND_INDEX_COMPOSITE "PCTFREE 10 INITRANS 4 NOLOGGING"
BND_INDEX_ID    "PCTFREE 10 INITRANS 4 NOLOGGING"
BND_STORAGE     "PCTFREE 0 INITRANS 4"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
```

```
B_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_XML     "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE       "PCTFREE 0 INITRANS 4"
D_INDEX_DELETED_AT      "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID     "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE       "PCTFREE 0 INITRANS 4"
F_INDEX_AREA    "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_FID     "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_LEN     "PCTFREE 10 INITRANS 4 NOLOGGING"
F_STORAGE       "PCTFREE 0 INITRANS 4"
GEOMETRY_STORAGE  "SDEBINARY"
RAS_INDEX_ID    "PCTFREE 10 INITRANS 4 NOLOGGING"
RAS_STORAGE     "PCTFREE 0 INITRANS 4"
S_INDEX_ALL     "PCTFREE 10 INITRANS 4 NOLOGGING"
S_INDEX_SP_FID  "PCTFREE 10 INITRANS 4 NOLOGGING"
S_STORAGE       "PCTFREE 0 INITRANS 4"
UI_TEXT         ""
XML_DOC_INDEX   "PCTFREE 10 INITRANS 4 NOLOGGING"
XML_DOC_STORAGE "PCTFREE 0 INITRANS 4"
XML_DOC_TEXT_TYPE "LONGRAW"
XML_IDX_INDEX_DOUBLE "PCTFREE 10 INITRANS 4 NOLOGGING"
XML_IDX_INDEX_ID "PCTFREE 10 INITRANS 4 NOLOGGING"
XML_IDX_INDEX_PK "PCTFREE 10 INITRANS 4 NOLOGGING"
XML_IDX_INDEX_TAG "PCTFREE 10 INITRANS 4 NOLOGGING"
XML_IDX_STORAGE "PCTFREE 0 INITRANS 4"
XML_IDX_TEXT_UPDATE_METHOD "NONE"

END
```

Setting the system table DATA_DICTIONARY configuration keyword

During the execution of the sdesetupora* administration tool, the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY configuration keyword. You may customize the configuration keyword in the dbtune.sde file prior to running the sdesetupora* tool. In this way, you can change default storage parameters of the DATA_DICTIONARY configuration keyword.

Edits to all the geodatabase system tables and most of the ArcSDE system tables occur when schema change occurs. As such, edits to these system tables and indexes usually happen during the initial creation of an ArcGIS database with infrequent modifications occurring whenever a new schema object is added.

Four of the ArcSDE system tables—VERSION, STATES, STATE_LINEAGES, and MVTABLES_MODIFIED—participate in the ArcSDE versioning model and record events resulting from changes made to multiversioned tables. If your site makes extensive use of a multiversioned database, these tables and their associated indexes are quite active. Separating these objects into their own tablespace allows you to position their data files with data files that experience low I/O activity and thus minimize disk I/O contention.

If the dbtune.sde file does not contain the DATA_DICTIONARY configuration keyword or if any of the required parameters are missing from the configuration keyword, the following records will be inserted into the DATA_DICTIONARY when the table is created. (Note that the DBTUNE file entries are provided here for readability.)

```
##DATA_DICTIONARY

B_INDEX_ROWID              "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 40K)"
B_INDEX_USER               "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 40K)"
B_STORAGE                  "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 40K)"
STATE_LINEAGES_TABLE       "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 40M)"
STATES_TABLE               "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 8M)"

STATES_INDEX               "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 1M)"
MVTABLES_MODIFIED_TABLE    "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 2M)"
MVTABLES_MODIFIED_INDEX    "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 2M)"
VERSIONS_TABLE             "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 256K)"
VERSIONS_INDEX             "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 128K)"
XML_INDEX_TAGS_INDEX       "INITRANS 5 STORAGE (INITIAL 128K)"
XML_INDEX_TAGS_TABLE       "INITRANS 4 STORAGE (INITIAL 128K)"
XML_TAGS_PK_INDEX          "INITRANS 5 STORAGE (INITIAL 128K)"
XML_TAGS_TABLE             "INITRANS 4 STORAGE (INITIAL 1M)"
XML_TAGS_UK_INDEX          "INITRANS 5 STORAGE (INITIAL 128K)"

END
```

The TOPOLOGY keyword

The TOPOLOGY keyword controls the storage of topology tables, which are named POINTERRORS, LINEERRORS, POLYERRORS, and DIRTYAREAS. An SDE instance must have a valid topology keyword in the DBTUNE table, or topology will not be built.

The DIRTYAREAS table maintains information on areas within a layer that have been changed. Because it tracks versions, data will be inserted or updated but not deleted during normal use. The DIRTYAREAS table will reduce in size only when database versions are compressed.

Because the DIRTYAREAS table is much more active than the remaining topology tables, the TOPOLOGY keyword may be compound. You may specify the DIRTYAREAS suffix to list the configuration string to be used to create the topology tables.

For Oracle, the default values for TOPOLOGY and TOPOLOGY::DIRTYAREAS are:

```
##TOPOLOGY_DEFAULTS
```

```
A_INDEX_ROWID       "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_STATEID     "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_USER        "PCTFREE 10 INITRANS 4 NOLOGGING"
A_STORAGE           "PCTFREE 0"
B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_USER        "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE           "PCTFREE 0 INITRANS 4"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE           "PCTFREE 10 INITRANS 4"
F_INDEX_AREA        "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_FID         "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_LEN         "PCTFREE 10 INITRANS 4 NOLOGGING"
F_STORAGE           "PCTFREE 10 INITRANS 4"
S_INDEX_ALL         "PCTFREE 10 INITRANS 4 NOLOGGING"
S_INDEX_SP_FID      "PCTFREE 10 INITRANS 4 NOLOGGING"
S_STORAGE           "PCTFREE 10 INITRANS 4"
UI_TOPOLOGY_TEXT    "The topology default configuration"
END

##TOPOLOGY_DEFAULTS::DIRTYAREAS
A_INDEX_ROWID       "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_STATEID     "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_USER        "PCTFREE 10 INITRANS 4 NOLOGGING"
A_STORAGE           "PCTFREE 10 INITRANS 4"
B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE       "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_USER        "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE           "PCTFREE 10 INITRANS 4"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE           "PCTFREE 10 INITRANS 4"
F_INDEX_AREA        "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_FID         "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_LEN         "PCTFREE 10 INITRANS 4 NOLOGGING"
F_STORAGE           "PCTFREE 10 INITRANS 4"
S_INDEX_ALL         "PCTFREE 10 INITRANS 4 NOLOGGING"
S_INDEX_SP_FID      "PCTFREE 10 INITRANS 4 NOLOGGING"
S_STORAGE           "PCTFREE 10 INITRANS 4"
END
```

The IMS METADATA keyword

The IMS METADATA keywords control the storage of the IMS metadata tables. They are a standard part of the DBTUNE table. If the IMS_METADATA storage parameters are not present in the DBTUNE file when it is imported into the DBTUNE table, ArcSDE will supply software defaults.

The software defaults have the same settings as the parameters listed in the dbtune.sde table, which is shipped with ArcSDE. The Oracle parameter settings, such as *pctfree* and *initrans,* should be sufficient. However, you will need to edit the tablespace names.

For more information about installing IMS metadata and the associated tables and indexes, refer to ArcIMS Metadata Server documentation.

The IMS_METADATA storage parameters control the storage of the ims_metadata feature class. Four indexes are created on the ims_metadata business table. ArcSDE creates the default IMS_METADATA keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

The IMS metadata keywords are as follows:

```
##IMS_METADATA

B_STORAGE            "PCTFREE 10 INITRANS 4
                      STORAGE (INITIAL 100M)"

B_INDEX_ROWID        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 6M) NOLOGGING"

B_INDEX_SHAPE        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 6M) NOLOGGING"

B_INDEX_USER         "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 6M) NOLOGGING"

F_STORAGE            "PCTFREE 0 INITRANS 4 STORAGE (INITIAL 40M)"

F_INDEX_FID          "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 3M) NOLOGGING"

F_INDEX_AREA         "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 3M) NOLOGGING"

F_INDEX_LEN          "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 3M) NOLOGGING"

S_STORAGE            "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 16M)"

S_INDEX_ALL          "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 16M)
NOLOGGING"

S_INDEX_SP_FID       "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 3M) NOLOGGING"

COMMENT              "The IMS metatdata feature class"

UI_TEXT              ""

END
```

The IMS_METADATARELATIONSHIPS keyword controls the storage of the ims_metadatarelationships business table. Three indexes are created on the ims_metadatarelationships business table. ArcSDE creates the default IMS_METADATARELATIONSHIPS keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATARELATIONSHIPS

B_STORAGE            "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 256K)"

B_INDEX_ROWID        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 64K)
NOLOGGING"
```

```
B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 64K)
NOLOGGING"

END
```

The IMS_METADATATAGS keyword controls the storage of the ims_metadatatags business table. Two indexes are created on the ims_metadatatags business table. ArcSDE creates the following default IMS_METADATATAGS keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATATAGS

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 64K)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 64K)
NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 64K)
NOLOGGING"

END
```

The IMS_METADATATHUMBNAILS keyword controls the storage of the ims_metadatathumbnails business table. One index is created on the ims_metadatathumbnails business table. ArcSDE creates the following default IMS_METADATATHUMBNAILS keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATATHUMBNAILS

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 4M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 256K)
NOLOGGING"

END
```

The IMS_METADATAUSERS keyword controls storage of the ims_metadatausers business table. One index is created on the ims_metadatausers business table. ArcSDE creates the following default IMS_METADATAUSERS keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATAUSERS

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 7M)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

END
```

The IMS_METADATAVALUES keyword controls the storage of the ims_metadatavalues business table. Two indexes are created on the ims_metadatavalues business table. ArcSDE creates the following default IMS_METADATAVALUES keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATAVALUES

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 7M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

END
```

The IMS_METADATAWORDINDEX keyword controls the storage of the ims_metadatawordindex business table. Three indexes are created on the ims_metadatawordindex business table. ArcSDE creates the following default IMS_METADATAWORDINDEX keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATAWORDINDEX

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 10M)"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

END
```

The IMS_METADATAWORDS keyword controls the storage of the ims_metadatawords business table. One index is created on the ims_metadatawords business table. ArcSDE creates the following default IMS_METADATAWORDS keyword in the DBTUNE table if the keyword is missing from the DBTUNE file when it is imported.

```
##IMS_METADATAWORDS

B_STORAGE           "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 7M)"

B_INDEX_ROWID       "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

B_INDEX_USER        "PCTFREE 10 INITRANS 4 STORAGE (INITIAL 512K)
NOLOGGING"

END
```

Note: These keywords are now obsolete. They will not be used by ArcSDE or ArcIMS Metadata Services. The gazetteer import scripts provided with ArcIMS 9.1 do not use the above keywords to load the gazetteer data into ArcSDE. For information on configuring XML storage, see Appendix F, ‘Storing XML Data’, of this document.

Setting the geometry storage format—the GEOMETRY_STORAGE parameter

ArcSDE for Oracle provides four spatial data storage formats. The GEOMETRY_STORAGE parameter indicates which geometry storage method is to be used. The GEOMETRY_STORAGE parameter has the following values:

- ArcSDE compressed binary stored in LONG RAW data type. This is the default spatial storage method of ArcSDE for Oracle.

  Set the GEOMETRY_STORAGE parameter to SDEBINARY if you want to store your spatial data in this format. If the GEOMETRY_STORAGE parameter is not set, the SDEBINARY format is assumed.

- ArcSDE compressed binary stored as a BLOB data type. This data type can be replicated through Oracle Advanced Replication.

  Set the GEOMETRY_STORAGE parameter to SDELOB if you want to store your spatial data in this format. If you want to make this format the default, set the GEOMETRY_STORAGE parameter to SDELOB in the DEFAULTS configuration keyword.

- Oracle Spatial geometry type. This object relational type extends the database model to include an SDO_GEOMETRY type in the Oracle DBMS.

  Set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY if you want to store your spatial data in this format. If you want to make this format the default, set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY in the DEFAULTS configuration keyword.

- OGC well-known binary (WKB) geometry type. This type provides a portable representation of a geometry as a contiguous stream of bytes.

  Set the GEOMETRY_STORAGE parameter to OGCWKB if you want to store your spatial data in this format. If you want to make this format the default, set the GEOMETRY_STORAGE parameter to OGCWKB in the DEFAULTS configuration keyword.

The OGCWKB representation supports only simple 2D geometries. See Appendix E for a description of the OGCWKB.

The default value for GEOMETRY_STORAGE is SDEBINARY.

If all the feature classes in your database use the same geometry storage method, set the GEOMETRY_STORAGE parameter once in the DEFAULTS configuration keyword. To change the default GEOMETRY_STORAGE from SDEBINARY to SDO_GEOMETRY, the following change is made:

```
## DEFAULTS
GEOMETRY_STORAGE    "SDO_GEOMETRY"
<other parameters>
END
```

For convenience, four predefined configuration keywords are provided in the installed dbtune.sde file to allow the use of each of the supported geometry storage methods. These configurations are defined as follows:

```
## SDEBINARY
GEOMETRY_STORAGE    "SDEBINARY"
END
##SDELOB
GEOMETRY_STORAGE    "SDELOB"
END
## SDO_GEOMETRY
GEOMETRY_STORAGE    "SDO_GEOMETRY"
END
##WKB_GEOMETRY
GEOMETRY_STORAGE     "OGCWKB"
```

Note: During the installation of ArcSDE on Windows, the ArcSDE installation allows you to choose the default spatial storage type. Depending on your selection, ArcSDE will load the DBTUNE table with the contents of one of three files, dbtune.lr, dbtune.blob, or dbtune.gt.

Setting the raster storage format—the RASTER_STORAGE parameter

ArcSDE provides three raster storage formats for Oracle. The RASTER_STORAGE parameter indicates which geometry storage method is to be used. The RASTER_STORAGE parameter has the following values:

- ArcSDE raster stored in LONG RAW data type. This is the default raster storage method of ArcSDE for Oracle.

  Set the RASTER_STORAGE parameter to LONG RAW if you want to store your raster data in this format. If the RASTER_STORAGE parameter is not set, the LONG RAW format is assumed.

- ArcSDE raster stored as a BLOB data type. This data type can be replicated through Oracle Advanced Replication.

Set the RASTER_STORAGE parameter to BLOB if you want to store your raster data in this format. If you want to make this format the default, set the RASTER_STORAGE parameter to BLOB in the DEFAULTS configuration keyword.

- Oracle GeoRaster type. This object relational type extends the database model to include an SDO_GEORASTER type in the Oracle DBMS.

  Set the RASTER_STORAGE parameter to SDO_GEORASTER if you want to store your spatial data in this format. If you want to make this format the default, set the RASTER_STORAGE parameter to SDO_GEOMETRY in the DEFAULTS configuration keyword.

The default value for RASTER_STORAGE is LONG RAW.

If all the raster columns in your database use the same raster storage format, set the RASTER_STORAGE parameter once in the DEFAULTS configuration keyword. To change the default RASTER_STORAGE from LONG RAW to SDO_GEORASTER, the following change is made:

```
## DEFAULTS
RASTER_STORAGE      "SDO_GEORASTER"
<other parameters>
END
```

The RASTER_STORAGE parameter supersedes the RASTER_BINARY_TYPE which continues to work but is no longer supported.

Setting the COMPRESS_ROLLBACK_SEGMENT storage parameter

Periodically compressing the versioned database’s state tree is a required maintenance procedure. The state tree can be compressed with either the compress operation of the sdeversion administration command or the SE_state_compress_tree() C API function.

The transactions of the compress operation tend to be large; if you are using the Oracle manual undo method, ESRI recommends that you create a separate large rollback segment to contain their changes. The COMPRESS_ROLLBACK_SEGMENT storage parameter stores the name of a rollback segment that you have created for this purpose. Add the COMPRESS_ROLLBACK_SEGMENT storage parameter to the DEFAULTS configuration keyword.

For more information on creating the rollback segment used for compressing the state tree, see Chapter 2, ‘Essential Oracle configuring and tuning’.

Changing the ATTRIBUTE_BINARY storage format

ArcSDE defines attribute columns used to store binary data as LONG RAW or as BLOB. The default is LONG RAW.

If the storage parameter is not set in the DEFAULTS configuration keyword when a DBTUNE file is imported by the sdedbtune administration tool, ArcSDE inserts the ATTRIBUTE_BINARY storage parameter under the DEFAULTS configuration keyword with a configuration string set to LONG RAW.

Changing the appearance of DBTUNE configuration keywords in the ArcGIS user interface

ArcSDE provides UI_TEXT and UI_NETWORK_TEXT storage parameters that allow you to change the appearance of the configuration keywords in the ArcGIS user interfaces.

ArcSDE administrators can add one of these storage parameters to each configuration keyword to communicate to the ArcGIS schema builders the intended use of the configuration keyword. The configuration string of these storage parameters will appear in the ArcGIS interface DBTUNE configuration keyword scrolling lists.

The UI_TEXT storage parameter should be added to configuration keywords that will be used to build tables, feature classes, and indexes.

The UI_NETWORK_TEXT storage parameter should be added to parent network configuration keywords.

Adding a comment to a configuration keyword

The COMMENT storage parameter allows you to add informative text that describes such things as a configuration keyword’s intended use, the last time it was changed, or who created it.

LOGFILE configuration keywords

Log files are used by ArcSDE to maintain temporary and persistent sets of selected records.

The LOGFILE_DEFAULTS keyword holds the parameter group for all users who do not have their own keyword created. Alternatively, you may create individual log file keywords for specific users by appending the user’s name to the LOGFILE_ prefix to form the keyword name. For example, if the user's name is STANLEY, ArcSDE will search the DBTUNE table for the LOGFILE_STANLEY configuration keyword. If this configuration keyword is not found, ArcSDE will use the storage parameters of the LOGFILE_DEFAULTS configuration keyword.

ArcSDE always creates the DBTUNE table with a LOGFILE_DEFAULTS configuration keyword. If you do not specify this configuration keyword in a DBTUNE file imported by the sdedbtune command, ArcSDE will populate the DBTUNE table with software default LOGFILE_DEFAULTS storage parameters. Further, if the DBTUNE file lacks some of the LOGFILE_DEFAULTS configuration keyword storage parameters, ArcSDE supplies the rest. Therefore, the LOGFILE_DEFAULTS configuration keyword is always fully populated.

If a user-specific configuration keyword exists but some of the storage parameters are not present, the storage parameters of the LOGFILE_DEFAULTS configuration keyword are used.

The storage parameters that are used depend on which type of log files the server has been configured to use. If the ArcSDE server is configured to use shared log files, ArcSDE creates the log file tables SDE_LOGFILES and SDE_LOGFILE_DATA and indexes the first time the user connects.

For the creation of shared log file tables, the LD_STORAGE and LF_STORAGE parameters control the storage of the SDE_LOGFILE_DATA and SDE_LOGFILES tables. By default, these tables are created with Oracle logging turned on (the Oracle NOLOGGING parameter is absent from the configuration string of these parameters). If you are not using a customized application that stores persistent log files, you should add NOLOGGING to the LD_STORAGE and LF_STORAGE parameters. ESRI applications accessing ArcSDE data use temporary log files.

The LF_INDEXES parameter defines the storage of the indexes of the SDE_LOGFILES table, while the LD_INDEX_DATA_ID and LD_INDEX_ROWID parameters define the storage of the SDE_LOGFILE_DATA table.

Creating a log file configuration keyword for each user allows you to position the SDE user's log files on separate devices by specifying the tablespace the log file tables and indexes are created in. Most installations of ArcSDE will function well using the LOGFILE_DEFAULTS storage parameters supplied with the installed dbtune.sde file. However, for applications using SDE log files, such as ArcGIS Desktop, it may help performance by spreading the log files across the file system. Typically, log files are updated whenever a selection set exceeds 100 records.

If you have configured the server to use session-based or standalone log files in addition to shared log files, ArcSDE will use a different set of storage parameters when it creates the session-based and standalone log file tables.

The SESSION_STORAGE parameter defines the storage of the session-based and standalone log file tables, which include both session and standalone types.

The SESSION_INDEX parameter defines the storage of the session-based and standalone log file table indexes.

If the imported DBTUNE file does not contain a LOGFILE_DEFAULTS configuration keyword or if any of the log file storage parameters are missing, ArcSDE will insert the following records:

```
##LOGFILE_DEFAULTS

LD_INDEX_DATA_ID  "PCTFREE 10 INITRANS 2 NOLOGGING"
LD_INDEX_ROWID    "PCTFREE 10 INITRANS 2 NOLOGGING"
LD_STORAGE        "PCTFREE 10 INITRANS 1"
LF_INDEXES        "PCTFREE 10 INITRANS 2 NOLOGGING"
LF_STORAGE        "PCTFREE 10 INITRANS 1"
SESSION_STORAGE   "PCTFREE 10 INITRANS 1"
SESSION_INDEX     "PCTFREE 10 INITRANS 2 NOLOGGING"
UI_TEXT ""

END
```

Network class composite configuration keywords

The composite keyword is a unique type of configuration keyword designed to accommodate the tables of the ArcGIS network class. The network table's size variation requires a configuration keyword that provides configuration storage parameters for both large and small tables. Typically, the network descriptions table is large compared with the others.

To accommodate the vast difference in the size of the network tables, the network composite configuration keyword is subdivided into elements. A network composite configuration keyword has three elements: the parent element defines the general characteristic of the configuration keyword and the junctions feature class, the description element defines the configuration of the DESCRIPTIONS table and its indexes, and the network element defines the configuration of the remaining network tables and their indexes.

The parent element does not have a suffix, and its configuration keyword looks like any other configuration keyword. The description element is demarcated by the addition of the ::DESC suffix to the parent element's configuration keyword, and the network element is demarcated by the addition of the ::NETWORK suffix to the parent element's configuration keyword.

For example, if the parent element configuration keyword is called ELECTRIC, the network composite configuration keyword would appear in a DBTUNE file as follows:

```
##ELECTRIC

COMMENT  This configuration keyword is dedicated to the electrical
geometric network class

UI_NETWORK_TEXT "The electrical geometrical network class configuration
keyword"

B_STORAGE     "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE (INITIAL
500M)"

B_INDEX_ROWID "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

B_INDEX_SHAPE "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

B_INDEX_USER  "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

F_STORAGE     "TABLESPACE FEATURE INITRANS 4 PCTFREE 0 STORAGE (INITIAL
500M)"

F_INDEX_FID   "TABLESPACE FEATURE_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

F_INDEX_LEN   "TABLESPACE FEATURE_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

F_INDEX_AREA  "TABLESPACE FEATURE_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

S_STORAGE      "TABLESPACE SPATIAL INITRANS 4 PCTFREE 10 STORAGE (INITIAL
200M)"

S_INDEX_ALL    "TABLESPACE SPATIAL_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 200M) NOLOGGING"

S_INDEX_SP_FID  "TABLESPACE SPATIAL_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 50M) NOLOGGING"

A_STORAGE       "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M)"

A_INDEX_ROWID    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

A_INDEX_SHAPE    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"
```

```
A_INDEX_USER     "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

A_INDEX_STATEID  "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

D_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M)"

D_INDEX_DELETED_AT   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10
STORAGE (INITIAL 20M) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

END

##ELECTRIC::DESC

B_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 500M)"

B_INDEX_ROWID    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

B_INDEX_USER     "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 100M) NOLOGGING"

A_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 500M)"

A_INDEX_ROWID    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

A_INDEX_SHAPE    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

A_INDEX_USER     "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

A_INDEX_STATEID  "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M) NOLOGGING"

D_STORAGE         "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 20M)"

D_INDEX_DELETED_AT   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10
STORAGE (INITIAL 20M) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10
STORAGE (INITIAL 20M) NOLOGGING"

END

##ELECTRIC::NETWORK

B_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M)"
```

```
B_INDEX_ROWID    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

B_INDEX_USER     "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

A_STORAGE        "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M)"

A_INDEX_ROWID    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

A_INDEX_SHAPE    "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

A_INDEX_USER     "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

A_INDEX_STATEID   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M) NOLOGGING"

D_STORAGE         "TABLESPACE BUSINESS INITRANS 4 PCTFREE 10 STORAGE
(INITIAL 1M)"

D_INDEX_DELETED_AT   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10
STORAGE (INITIAL 1M) NOLOGGING"

D_INDEX_STATE_ROWID   "TABLESPACE BUSINESS_INDEX INITRANS 4 PCTFREE 10
STORAGE (INITIAL 1M) NOLOGGING"

END
```

Following the import of the DBTUNE file, these records would be inserted into the DBTUNE table:

```
SQL> select keyword, parameter_name from DBTUNE;

KEYWORD             PARAMETER_NAME
----------------    ----------------
ELECTRIC            COMMENT
ELECTRIC            UI_NETWORK_TEXT
ELECTRIC            B_STORAGE
ELECTRIC            B_INDEX_ROWID
ELECTRIC            B_INDEX_SHAPE
ELECTRIC            B_INDEX_USER
ELECTRIC            F_STORAGE
ELECTRIC            F_INDEX_FID
ELECTRIC            F_INDEX_LEN
ELECTRIC            F_INDEX_AREA
ELECTRIC            S_STORAGE
ELECTRIC            S_INDEX_ALL
ELECTRIC            S_INDEX_SP_FID
ELECTRIC            A_STORAGE
ELECTRIC            A_INDEX_ROWID
ELECTRIC            A_INDEX_SHAPE
ELECTRIC            A_INDEX_USER
ELECTRIC            A_INDEX_STATEID
ELECTRIC            D_STORAGE
ELECTRIC            D_INDEX_DELETED_AT
ELECTRIC            D_INDEX_STATE_ROWID
ELECTRIC::DESC      B_STORAGE
ELECTRIC::DESC      B_INDEX_ROWID
ELECTRIC::DESC      B_INDEX_USER
ELECTRIC::DESC      A_STORAGE
ELECTRIC::DESC      A_INDEX_ROWID
ELECTRIC::DESC      A_INDEX_STATEID
ELECTRIC::DESC      A_INDEX_USER
ELECTRIC::DESC      D_STORAGE
ELECTRIC::DESC      D_INDEX_DELETE_AT
ELECTRIC::DESC      D_INDEX_STATE_ROWID
ELECTRIC::NETWORK   B_STORAGE
ELECTRIC::NETWORK   B_INDEX_ROWID
ELECTRIC::NETWORK   B_INDEX_USER
ELECTRIC::NETWORK   A_STORAGE
ELECTRIC::NETWORK   A_INDEX_ROWID
ELECTRIC::NETWORK   A_INDEX_STATEID
ELECTRIC::NETWORK   A_INDEX_USER
ELECTRIC::NETWORK   D_STORAGE
ELECTRIC::NETWORK   D_INDEX_DELETE_AT
ELECTRIC::NETWORK   D_INDEX_STATE_ROWID
```

The network junctions feature class is created with the ELECTRIC configuration keyword storage parameters, the network descriptions table is created with the storage parameters of the ELECTRIC::DESC configuration keyword, and the remaining smaller network tables are created with the ELECTRIC::NETWORK configuration keyword.

The NETWORK_DEFAULTS configuration keyword

The NETWORK_DEFAULTS configuration keyword contains the default storage parameters for the ArcGIS network class. If you do not choose a network class composite configuration keyword from the ArcCatalog interface, the ArcGIS network is created with the storage parameters within the NETWORK_DEFAULTS configuration keyword.

Whenever a network class composite configuration keyword is selected, its storage parameters are used to create the feature class, table, and indexes of the network class. If a network composite configuration keyword is missing any storage parameters, ArcGIS substitutes the storage parameters of the DEFAULTS configuration keyword, rather than the NETWORK_DEFAULTS configuration keyword. The storage parameters of the NETWORK_DEFAULTS configuration keyword are used when a network composite configuration keyword has not been specified.

If a NETWORK_DEFAULTS configuration keyword is not present in a DBTUNE file imported into the DBTUNE table, the following NETWORK_DEFAULTS configuration keyword is created:

```
##NETWORK_DEFAULTS

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
A_STORAGE       "PCTFREE 10 INITRANS 4"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE       "PCTFREE 10 INITRANS 4"
COMMENT "The base system initialization parameters for NETWORK_DEFAULTS"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE       "PCTFREE 10 INITRANS 4"
F_INDEX_AREA    "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_FID     "PCTFREE 10 INITRANS 4 NOLOGGING"
F_INDEX_LEN     "PCTFREE 10 INITRANS 4 NOLOGGING"
F_STORAGE       "PCTFREE 0 INITRANS 4"
S_INDEX_ALL     "PCTFREE 10 INITRANS 4 NOLOGGING"
S_INDEX_SP_FID  "PCTFREE 10 INITRANS 4 NOLOGGING"
S_STORAGE       "PCTFREE 0 INITRANS 4"
UI_NETWORK_TEXT "The network default configuration"
END

##NETWORK_DEFAULTS::DESC

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4"
A_INDEX_USER    "PCTFREE 10 INITRANS 4"
A_STORAGE       "PCTFREE 0 INITRANS 4"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE       "PCTFREE 0 INITRANS 4"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE       "PCTFREE 0 INITRANS 4"
END

##NETWORK_DEFAULTS::NETWORK

A_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_STATEID "PCTFREE 10 INITRANS 4 NOLOGGING"
A_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
A_STORAGE       "PCTFREE 0 INITRANS 4"
B_INDEX_ROWID   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE   "PCTFREE 10 INITRANS 4 NOLOGGING"
B_INDEX_USER    "PCTFREE 10 INITRANS 4 NOLOGGING"
B_STORAGE       "PCTFREE 0 INITRANS 4"
D_INDEX_DELETED_AT  "PCTFREE 10 INITRANS 4 NOLOGGING"
D_INDEX_STATE_ROWID "PCTFREE 10 INITRANS 4 NOLOGGING"
D_STORAGE       "PCTFREE 0 INITRANS 4"
END
```

Oracle Spatial DBTUNE storage parameters

Oracle Spatial feature classes are ArcSDE feature classes with Oracle's SDO_GEOMETRY data type to store feature geometry in a column in the business table. Many of the storage parameters used for ArcSDE compressed binary feature classes apply to Oracle Spatial feature classes. Also, some storage parameters exist solely to establish how Oracle Spatial feature classes are stored, indexed, and accessed.

For a description of Oracle Spatial, consult the *Oracle Spatial User's Guide and Reference*. For more information about how Oracle Spatial and ArcSDE work together, see Appendix D of this document.

Creating new business tables

ArcSDE uses the B_STORAGE, B_INDEX_ROWID, and B_INDEX_USER storage parameters to store the business table and the nonspatial indexes on the business table.

Creating Oracle Spatial metadata for new feature classes

The database view USER_SDO_GEOM_METADATA is part of Oracle Spatial, not ArcSDE. It contains metadata about SDO_GEOMETRY columns in existing tables owned by the user. Each user has its own USER_SDO_GEOM_METADATA view. To be indexed and queried, the owner of the table must record metadata for each SDO_GEOMETRY column in USER_SDO_GEOM_METADATA. The ArcSDE clients that create a feature class will choose the metadata for the feature class. Often, these clients accept a configuration keyword corresponding to a parameter group in the DBTUNE table.

The storage parameters that control the metadata for new Oracle Spatial feature classes are:

SDO_DIMNAME_<n>
SDO_LB_<n>
SDO_UB_<n>
SDO_TOLERANCE_<n>
SDO_SRID

Oracle Spatial permits feature geometries of two, three, or four dimensions in the combinations X/Y, X/Y/Z, X/Y/M (measure), or X/Y/Z/M. Through these storage parameters, ArcSDE allows you to specify metadata for each dimension. The <n> in some parameter names should be replaced by one of the digits 1, 2, 3, or 4, corresponding to the dimension number. If you do not supply these storage parameters, the ArcSDE client application that creates the feature class will determine the name, upper and lower bound (extent), and tolerance of each dimension.

Creating a spatial index

The DBTUNE parameter SDO_INDEX_SHAPE determines how Oracle Spatial creates the spatial index. ArcSDE appends the contents of this parameter (the configuration string) to the CREATE INDEX statement before submitting the statement to Oracle. The configuration string is inserted into the SQL statement after the PARAMETERS keyword. For example:

```
CREATE INDEX MY_SP_INDEX ON MY_SP_TABLE(SHAPE)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ( <configuration string is inserted here> );
```

The configuration string is a quoted string containing a list of *parameter* = *value* elements. There are many parameters that you can specify in the configuration string. To understand the Oracle Spatial index parameters and how they interact, read the applicable sections of the *Oracle Spatial User's Guide and Reference.*

Notice the differences between the physical storage parameters in the spatial index configuration string and in a business table configuration string (as specified in B_STORAGE). One difference is due to the way Oracle expects these parameters to appear in SQL statements. The Oracle statements are formatted differently, so the configuration strings are formatted differently. Also, not every physical storage parameter used for creating tables is available for creating spatial indexes.

```
B_STORAGE                   "TABLESPACE ORSPBIZ PCTFREE 10 INITRANS 4
                            STORAGE(INITIAL 512000)"

SDO_INDEX_SHAPE             "tablespace=ORSPIDX initial=512000"
```

Verifying polygon rotation

Both ArcSDE and Oracle Spatial expect that exterior polygon boundaries are stored in counterclockwise rotation and that inner boundaries are stored clockwise. If the rotation polygon boundary is not stored with this rotation, then the polygon will fail the ArcSDE topographic validation test and will not be sent to the ArcSDE client. The SDO_VERIFY storage parameter tells ArcSDE whether it should check (and reorder, if necessary) the rotation of polygon boundaries as they are fetched from Oracle Spatial. This check happens automatically for autoregistered Oracle Spatial feature classes. You should consider using this parameter for third party Oracle Spatial tables that you manually register—for example, using the “sdelayer” command. Unlike other parameters used for storing and indexing feature classes, this parameter is used when a feature class is read from the database.

Sample DBTUNE parameter groups for Oracle Spatial feature classes

This section presents DBTUNE parameter groups that apply to several common scenarios. These samples emphasize the storage parameters for Oracle Spatial feature classes. When designing your own parameter groups, you may want to add parameters to support other needs, such as geometric networks or log files. You could also satisfy these requirements via parameters in the DEFAULTS parameter group.

In some examples, you will see the parameter sdo_indx_dims=2, which specifies how many dimensions should be indexed with an R-tree spatial index. With Oracle 9.2, the default value is 2, meaning the first two dimensions, X and Y. For previous versions of Oracle, the default value was the number of dimensions recorded in USER_SDO_GEOM_METADATA. There were various problems creating R-tree spatial indexes on dimensions other than X and Y. For example, Oracle does not support R-tree indexes on the measure dimension. If you are creating R-tree spatial indexes and are using a version of Oracle prior to 9.2, it is recommended you always include this parameter.

If you are not using Oracle Spatial by default, you can create a simple parameter group to create Oracle Spatial feature classes with mostly default settings. The tables and indexes will be created in the user's default tablespace using default physical storage parameters, unless specified otherwise in the DEFAULTS parameter group. The spatial index will be a two dimensional R-tree.

```
##SDO_GEOMETRY
GEOMETRY_STORAGE            "SDO_GEOMETRY"
SDO_INDEX_SHAPE             "sdo_indx_dims=2"
UI_TEXT                     "Oracle Spatial: default settings"
END
```

With Oracle Spatial, if data is often loaded using a specific spatial reference identifier (SRID), such as the geodetic SRID 8307 (Latitude/Longitude WGS 84), you can create an expanded version of the previous parameter group. You don't have to specify the upper and lower bounds and tolerance, but you can if you want all your feature classes to have the same metadata for the X and Y dimensions. This is useful if you want to use the feature classes in the same feature dataset. Also, this case specifies that any polygon boundaries with reversed rotation will be reordered before sending them to ArcSDE clients.

Note: With Oracle9*i* geodetic data, the extents are specified in decimal degrees and the tolerances are specified in meters.

```
##SDO_GEOMETRY_8307
GEOMETRY_STORAGE            "SDO_GEOMETRY"
SDO_INDEX_SHAPE             "sdo_indx_dims=2"
SDO_SRID                    8307
SDO_DIMNAME_1               "Lon"
SDO_LB_1                    -180.000000
SDO_UB_1                    180.000000
SDO_TOLERANCE_1             0.001
SDO_DIMNAME_2               "Lat"
SDO_LB_2                    -90.000000
SDO_UB_2                    90.000000
SDO_TOLERANCE_2             0.001
SDO_VERIFY                  TRUE
UI_TEXT                     "Oracle Spatial: WGS84"
END
```

The following example can be used to load a feature class with an R-tree spatial index into a tablespace called ORSPBIZ. The R-tree spatial index will be created in the tablespace ORSPIDX. The ArcSDE client that is loading the data will decide the values for the metadata.

```
##SDO_GEOMETRY_ORSPBIZ
GEOMETRY_STORAGE            "SDO_GEOMETRY"
B_STORAGE                   "TABLESPACE ORSPBIZ"
SDO_INDEX_SHAPE             "tablespace=ORSPIDX sdo_indx_dims=2"
UI_TEXT                     "Tablespace ORSPBIZ / ORSPIDX"
END
```

The following example can be used to load a feature class with a quadtree spatial index having a fixed-sized tiling level (tessellation level) of 6. The spatial index will be created in the tablespace ORSPIDX. Commit interval is important for quadtree indexes. It indicates the number of business table records that will be processed before committing the index data. Without it, all the records of the business table are processed before committing the index data. This will cause problems when indexing tables with many records.

Note: The parameter sdo_commit_interval is so important that ArcSDE automatically includes it in SQL indexing statements for Oracle Spatial tables, even if you do not specify it as part of the SDO_INDEX_SHAPE parameter. It is set to 1,000.

```
##SDO_GEOMETRY_QT_6
GEOMETRY_STORAGE            "SDO_GEOMETRY"
SDO_INDEX_SHAPE             "tablespace=ORSPIDX sdo_level=6
                            sdo_commit_interval=1000"
END
```

Oracle default parameters

By default, Oracle stores tables and indexes in the user's default tablespace using the tablespace's default storage parameters. Determine a user's default tablespace by querying the DEFAULT_TABLESPACE field of the USER_USERS Oracle system table when connected as that user. As the Oracle DBA, query the DEFAULT_TABLESPACE field of the DBA_USERS table using a Where clause to specify the user.

```
SQL> connect <user>/<password>
SQL> select default_tablespace from user_users;
```

or

```
SQL> connect system/<password>
SQL> select default_tablespace from dba_users where username = <user>;
```

Obtain a list of default storage parameters for a tablespace by querying USER_TABLESPACES:

```
SQL> connect <user>/<password>
SQL> select * from user_tablespaces where tablespace_name = <tablespace>;
```

Converting previous versions of SDE storage parameters into the DBTUNE table

For older versions of the Spatial Database Engine, the DBTUNE storage parameters were maintained in the dbtune.sde file. The storage parameters of these previous versions were mapped directly to each Oracle storage parameter. For example, the obsolete F_TBLSP storage parameter holds the name of the feature table's tablespace.

Beginning at ArcSDE 8.1, storage parameters hold entire configuration strings of the table or index they represent. For example, the F_STORAGE parameter holds the configuration string of the feature table. Any legal Oracle storage parameter listed to the right of the columns clause of the Oracle CREATE TABLE statement can be listed in the F_STORAGE parameter. Therefore, the F_STORAGE parameter incorporates all the obsolete feature table storage parameters.

If you are upgrading to ArcSDE 8.1 or higher from a version of ArcSDE prior to ArcSDE 8.1, the conversion of the DBTUNE files' storage parameters occurs automatically when the sdesetupora* utility reads the storage parameters from the previous version's dbtune.sde file. The import operation of the sdedbtune command will convert a DBTUNE file into current ArcSDE storage parameters before it writes them to the DBTUNE table. To see the results, you can either use SQL*Plus to list the storage parameters of the DBTUNE table, or write the storage parameters of the DBTUNE table to another file using the export operation of the sdedbtune command.

The following table lists the conversion of ArcSDE 8.0.2 software-style storage parameters to ArcSDE 9 software-style storage parameters.

The ArcSDE 8.0.2 business table and index parameter prefix is "A_". The ArcSDE 9 business table and index parameter prefix is "B_". The ArcSDE 8.0.2 business table storage parameters are converted to the single ArcSDE 9 B_STORAGE parameter. The B_STORAGE parameter holds the entire business table's configuration string.

ArcSDE 8.0.2 storage parameters		ArcSDE 9 storage parameters	
A_TBLSP	ROADS	B_STORAGE	"TABLESPACE ROADS
A_INIT	10M		STORAGE (INITIAL 10M)
A_NEXT	5M		INITRANS 5
A_MINX	1		PCTFREE 10"
A_MAXX	200		
A_PCTI	0		
A_ITRANS	5		
A_MAXTRS	255		
A_PCTFREE	10		
A_PCTUSD	90		

The ArcSDE 8.0.2 business table index storage parameters are converted to ArcSDE 9 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 storage parameters are converted into the ArcSDE 9 spatial column index storage parameter B_INDEX_SHAPE. The other ArcSDE 9 business table index storage parameters, B_INDEX_ROWID and B_INDEX_USER, are also constructed this way.

ArcSDE 8.0.2 storage parameters		ArcSDE 9 storage parameters	
INDEX_TABLESPACE	ROADS_IX	B_INDEX_SHAPE	"TABLESPACE ROADS_IX
			STORAGE (INITIAL 10M)
A_IX1_INIT	10M		INITRANS 5
A_IX1_NEXT	5M		PCTFREE 10"
A_MINX	1		
A_MAXX	200		
A_PCTI	0		
A_ITRANS	5		
A_MAXTRS	255		
A_PCTFREE	10		

The ArcSDE 8.0.2 feature table storage parameters are converted to the ArcSDE 9 F_STORAGE parameter. The F_STORAGE parameter holds the entire feature table's configuration string.

ArcSDE 8.0.2 storage parameters

F_TBLSP	ROADS_F
F_INIT	10M
F_NEXT	5M
F_MINX	1
F_MAXX	200
F_PCTI	0
F_ITRANS	5
F_MAXTRS	255
F_PCTFREE	10
F_PCTUSD	90

ArcSDE 9 storage parameters

```
F_STORAGE   "TABLESPACE ROADS_F
             STORAGE (INITIAL 10M)
             INITRANS 5
             PCTFREE 10"
```

The ArcSDE 8.0.2 feature table index storage parameters are converted to ArcSDE 9 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 storage parameters are converted into the ArcSDE 9 FID column index storage parameter F_INDEX_FID. The other ArcSDE 9 feature table index storage parameters, F_INDEX_AREA and F_INDEX_LEN, are also constructed this way.

ArcSDE 8.0.2 storage parameters

INDEX_TABLESPACE	ROADS_F_IX
F_IX1_INIT	10M
F_IX1_NEXT	5M
F_MINX	1
F_MAXX	200
F_PCTI	0
F_ITRANS	5
F_MAXTRS	255

ArcSDE 9 storage parameters

```
F_INDEX_FID     "TABLESPACE ROADS_F_IX
                 STORAGE (INITIAL 10M)
                 INITRANS 5
                 PCTFREE 10"
```

The ArcSDE 8.0.2 spatial index table storage parameters are converted to the ArcSDE 9 S_STORAGE parameter. The S_STORAGE parameter holds the entire spatial index table's configuration string.

ArcSDE 8.0.2 storage parameters		ArcSDE 9 storage parameters	
S_TBLSP	ROADS_S	S_STORAGE	"TABLESPACE ROADS_S STORAGE (INITIAL 10M) INITRANS 5 PCTFREE 10"
S_INIT	10M		
S_NEXT	5M		
S_MINX	1		
S_MAXX	200		
S_PCTI	0		
S_ITRANS	5		
S_MAXTRS	255		
S_PCTFREE	10		
S_PCTUSD	90		

The ArcSDE 8.0.2 spatial index table storage parameters are converted to ArcSDE 9 storage parameter configuration strings. The example below illustrates how the ArcSDE 8.0.2 index storage parameters are converted into the ArcSDE 9 storage parameter S_INDEX_ALL. The S_INDEX_SP_FID storage parameter is converted the same way except ArcSDE 8.0.2 storage parameters S_IX2_INIT and S_IX2_NEXT are used.

ArcSDE 8.0.2 storage parameters		ArcSDE 9 storage parameters	
INDEX_TABLESPACE	ROADS_S_IX	S_INDEX_ALL	"TABLESPACE ROADS_S_IX STORAGE (INITIAL 10M) INITRANS 5 PCTFREE 10"
S_IX1_INIT	10M		
S_IX1_NEXT	5M		
S_MINX	1		
S_MAXX	200		
S_PCTI	0		
S_ITRANS	5		
S_MAXTRS	255		
S_PCTFREE	10		

The complete list of ArcSDE 9 storage parameters

Parameter Name	Value	Parameter Description	Default Value
STATES_LINEAGES_TABLE	<string>	State_lineages table	B_STORAGE
STATES_TABLE	<string>	States table	B_STORAGE
STATES_INDEX	<string>	States indexes	B_INDEX_USER
MVTABLES_MODIFIED_TABLE	<string>	Mvtables_modified table	B_STORAGE
MVTABLES_MODIFIED_INDEX	<string>	Mvtables_modified index	B_INDEX_USER
VERSIONS_TABLE	<string>	Versions table	B_STORAGE
VERSIONS_INDEX	<string>	Version index	B_INDEX_USER
COMPRESS_ROLLBACK_SEGMENT	<string>	Version compression rollback segment (only applies to databases that are using manual undo space management)	No default value
B_STORAGE	<string>	Business table	"PCTFREE 0 INITRANS 4"
B_INDEX_ROWID	<string>	Business table object ID column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
B_INDEX_SHAPE	<string>	Business table spatial column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
B_INDEX_USER	<string>	Business table user indexes	"PCTFREE 0 INITRANS 4 NOLOGGING"

Parameter Name	Value	Parameter Description	Default Value
B_INDEX_RASTER	<string>	Business table raster column index	"PCTFREE 0 INITRANS4 NOLOGGING"
B_INDEX_XML	<string>	Business table XML column index table	“PCTFREE 0 INITRANS 4 NOLOGGING”
F_STORAGE	<string>	Feature table	“PCTFREE 0 INITRANS 4”
F_INDEX_FID	<string>	Feature table FID column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
F_INDEX_AREA	<string>	Feature table area column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
F_INDEX_LEN	<string>	Feature table length column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
S_STORAGE	<string>	Spatial index table	“PCTFREE 0 INITRANS 4”
S_INDEX_ALL	<string>	Spatial index table first index	“PCTFREE 0 INITRANS 4 NOLOGGING”
S_INDEX_SP_FID	<string>	Spatial index table second index	“PCTFREE 0 INITRANS 4 NOLOGGING”
A_STORAGE	<string>	Adds table	“PCTFREE 0 INITRANS 4”
A_INDEX_ROWID	<string>	Adds table object ID column index	“PCTFREE 0 INITRANS 4 NOLOGGING”

Parameter Name	**Value**	**Parameter Description**	**Default Value**
A_INDEX_SHAPE	<string>	Adds table spatial column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
A_INDEX_STATEID	<string>	Adds table sde_state_id column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
A_INDEX_USER	<string>	Adds table index	"PCTFREE 0 INITRANS 4 NOLOGGING"
A_INDEX_XML	<string>	Adds table XML column index table	"PCTFREE 0 INITRANS 4 NOLOGGING"
D_STORAGE	<string>	Deletes table	"PCTFREE 0 INITRANS 4"
D_INDEX_ STATE_ROWID	<string>	Deletes table sde_states_id and sde_deletes_row_id column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
D_INDEX_DELETED_AT	<string>	Deletes table sde_deleted_at column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
LF_STORAGE	<string>	SDE shared log files table	"PCTFREE 0 INITRANS 4"
LF_INDEXES	<string>	SDE shared log file table column indexes	"PCTFREE 0 INITRANS 4 NOLOGGING"
LD_STORAGE	<string>	SDE shared log file data table	"PCTFREE 0 INITRANS 4"
LD_INDEX_DATA_ID	<string>	SDE shared log file data table	"PCTFREE 0 INITRANS 4 NOLOGGING"

Parameter Name	Value	Parameter Description	Default Value
LD_INDEX_ROWID	<string>	SDE shared log file data table SDE ROWID column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
SESSION_STORAGE	<string>	SDE session-based and standalone log file tables	“PCTFREE 0 INITRANS 4”
SESSION_INDEX	<string>	SDE session-based and standalone log file indexes	“PCTFREE 0 INITRANS 4 NOLOGGING”
RAS_STORAGE	<string>	Raster RAS table	“PCTFREE 0 INITRANS 4”
RAS_INDEX_ID	<string>	Raster RAS table RID index	“PCTFREE 0 INITRANS 4 NOLOGGING”
BND_STORAGE	<string>	Raster BND table	“PCTFREE 0 INITRANS 4”
BND_INDEX_COMPOSITE	<string>	Raster BND table composite column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
BND_INDEX_ID	<string>	Raster BND table RID column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
AUX_STORAGE	<string>	Raster AUX table	“PCTFREE 0 INITRANS 4”
AUX_INDEX_COMPOSITE	<string>	Raster AUX table composite column index	“PCTFREE 0 INITRANS 4 NOLOGGING”
BLK_STORAGE	<string>	Raster BLK table	“PCTFREE 0 INITRANS 4”

Parameter Name	Value	Parameter Description	Default Value
BLK_INDEX_COMPOSITE	<string>	Raster BLK table composite column index	"PCTFREE 0 INITRANS 4 NOLOGGING"
ATTRIBUTE_BINARY	<string>	Set this storage parameter to LONG RAW or BLOB	LONG RAW (or BLOB for NETWORK_ DEFAULTS)
GEOMETRY_STORAGE	<string>	Set this storage parameter to SDEBINARY, SDELOB, OGCWKB, or SDO_GEOMETRY	SDEBINARY
RASTER_STORAGE	<string>	Set the storage parameter to LONG RAW, BLOB or SDO_GEORASTER	LONG RAW
SDO_DIMNAME_1 SDO_DIMNAME_2 SDO_DIMNAME_3 SDO_DIMNAME_4	<string>	The name of each dimension for Oracle Spatial geometry types	"X" "Y" "Z" "M"
SDO_INDEX_SHAPE	<string>	The Oracle Spatial geometry types spatial index storage parameters	Oracle defaults
SDO_LB_1 SDO_LB_2 SDO_LB_3 SDO_LB_4	<floating-point number>	Lower dimension boundary for Oracle Spatial geometry type; units specified in coordinate system of the data	Based on extent of data to be loaded. For Oracle9*i* data with geodetic SRID, SDO_LB_1 *must be* -180, and SDO_LB_2 *must be* -90.

Parameter Name	Value	Parameter Description	Default Value
SDO_SRID	<integer>	Oracle Spatial coordinate reference identifier assigned to the SDO_GEOMETRY column	NULL
SDO_TOLERANCE_1 SDO_TOLERANCE_2 SDO_TOLERANCE_3 SDO_TOLERANCE_4	<floating-point number>	The distance two ordinates can be apart in the given dimension and still be considered the same; used by Oracle Spatial functions; must be greater than zero; for Oracle9*i* geodetic data, units are meters; otherwise, units are specified in coordinate system of the data	Oracle9*i* all data: 0.001 Oracle8*i* projected data: 0.001 Oracle8*i* unprojected data: 0.0000005
SDO_UB_1 SDO_UB_2 SDO_UB_3 SDO_UB_4	<floating-point number>	Upper dimension boundary for Oracle Spatial geometry type; used by Oracle Spatial functions; must be greater than zero; for Oracle9*i* geodetic data, units are meters; otherwise, units are specified in coordinate system of the data	Based on extent of data to be loaded For Oracle9*i* data with geodetic SRID, SDO_UB_1 *must be* 180, and SDO_UB_2 *must be* 90
SDO_VERIFY	<bool>	If TRUE, ArcSDE will verify polygon boundary rotations as they are accessed from Oracle Spatial	FALSE
UI_TEXT	<string>	String describing configuration	“”
XML_DOC_INDEX	<string>	Storage clause for *xmldoc<n>_pk* index on the *sde_xml_doc<n>* table	“PCTFREE 0 INITRANS 4 NOLOGGING”
XML_DOC_STORAGE	<string>	Storage clause for *sde_xml_doc<n>* table	“PCTFREE 0 INITRANS 4”

Parameter Name	**Value**	**Parameter Description**	**Default Value**
XML_DOC_TEXT_TYPE	<string>	Data type for document text column	“LONG RAW”
XML_IDX_INDEX_DOUBLE	<string>	Storage clause for the *xmlix<n>_db* index on the *double_tag* column of the *sde_xml_idx<n>* table	“PCTFREE 0 INITRANS 4 NOLOGGING”
XML_IDX_INDEX_ID	<string>	Storage clause for the *xmlix<n>_id* index on the *sde_xml_id* column of *sde_xml_idx<n>* table	“PCTFREE 0 INITRANS 4 NOLOGGING”
XML_IDX_INDEX_PK	<string>	Storage clause for *xmlix<n>_pk* index on the *xml_key_column* identity column of the *sde_xml_idx<n>* table	“PCTFREE 0 INITRANS 4 NOLOGGING”
XML_IDX_INDEX_TAG	<string>	Storage clause for the *xmlix<n>_tg* index on the *tag_id* column of the *sde_xml_idx<n>* table	“PCTFREE 0 INITRANS 4 NOLOGGING”
XML_IDX_INDEX_TEXT	<string>	Index creation parameters (see *Oracle Text Reference)*	“”
XML_IDX_STORAGE	<string>	Storage clause for *sde_xml_idx<n* table	“PCTFREE 0 INITRANS 4”

Parameter Name	Value	Parameter Description	Default Value
XML_IDX_TEXT_UPDATE_METHOD	<string>	Oracle Text index change tracking method: NONE: Manual update by running Oracle Text package BUFFERED: ArcSDE updates when stream is closed IMMEDIATE: ArcSDE updates on row insert or update	"NONE"
XML_INDEX_TAGS_INDEX	<string>	Storage clause for *xml_indextags_pk* index (used under DATA_DICTIONARY keyword)	"INITRANS 5 STORAGE (INITIAL 1M)"
XML_INDEX_TAGS_TABLE	<string>	Storage clause for *sde_xml_index_tags* table (used under DATA_DICTIONARY keyword)	"INITRANS 4 STORAGE (INITIAL 1M)"
XML_TAGS_PK_INDEX	<string>	Storage clause for *xml_tag_pk* index (used under DATA_DICTIONARY keyword)	"INITRANS 5 STORAGE (INITIAL 1M)"
XML_TAGS_TABLE	<string>	Storage clause for *sde_xml_tags* table (used under DATA_DICTIONARY keyword)	"INITRANS 4 STORAGE (INITIAL 1M)"
XML_TAGS_UK_INDEX	<string>	Storage clause for *xml_tag_uk1* and *xml_tag_uk2* indexes (used under DATA_DICTIONARY keyword)	"INITRANS 5 STORAGE (INITIAL 1M)"

CHAPTER 4

Managing tables, feature classes, and raster columns

A fundamental part of any database is creating and loading the tables. Tables with spatial columns are called standalone feature classes. Attribute-only (nonspatial) tables are also an important part of any database. This chapter will describe the table and feature class creation and loading process.

Data creation

There are numerous applications that can create and load data within an ArcSDE Oracle database. These include:

1. ArcSDE administration commands located in the bin directory of SDEHOME:
   - sdelayer—Creates and manages feature classes.
   - sdetable—Creates and manages tables.
   - sdeimport—Takes an existing sdeexport file and loads the data into a feature class.
   - shp2sde—Loads an ESRI shapefile into a feature class.
   - cov2sde—Loads a coverage, Map LIBRARIAN layer, or ArcStorm™ layer into a feature class.
   - tbl2sde—Loads an attribute-only dBASE® or INFO™ file into a table.
   - sdegroup—A specialty feature class creation command that combines the features of an existing feature class into single multipart features and stores them in a new feature class for background display. The generated feature class is used for rapid display of a large amount of geometry data. The attribute information is not retained, and spatial searches cannot be performed on these feature classes.

- sderaster—Creates, inserts, modifies, imports, and manages raster data stored in an ArcSDE database.

These are all run from the operating system prompt. Command references for these tools are in *ArcSDE Developer Help*.

Other applications include:

2. ArcGIS Desktop—Use ArcCatalog or ArcToolbox to manage and populate your database.
3. ArcInfo Workstation—Use the Defined Layer interface to create and populate the database.
4. ArcView® 3.2—Use the Database Access extension.
5. MapObjects®—Custom Component Object Model (COM) applications can be built to create and populate databases.
6. ArcSDE CAD Client extension—For AutoCAD® and MicroStation® users.
7. Other third party applications built with either the C or Java™ APIs.

This document focuses primarily on the ArcSDE administration tools but does provide some ArcGIS Desktop examples as well. In general, most people prefer an easy-to-use graphic user interface like the one found in ArcGIS Desktop. For details on how to use ArcCatalog or ArcToolbox (another desktop data loading tool), please refer to the ArcGIS books:

- *Using ArcCatalog*
- *Building a Geodatabase*

Creating and populating a feature class

The general process involved with creating and loading a feature class is:

1. Create the business table.
2. Record the business table and the spatial column in the ArcSDE LAYERS and GEOMETRY_COLUMNS system tables, thus adding a new feature class to the database.
3. Switch the feature class to load_only_io mode—an optional step to improve bulk data loading performance. It is OK to leave the feature class in normal_io mode to load data.
4. Insert the records (load data).
5. Switch the feature class to normal_io mode (builds the indexes).
6. Version the data (optional).
7. Grant privileges on the data (optional).

In the following sections, this process is discussed in more detail and illustrated with some examples of ArcSDE administration commands usage and ArcInfo data-loading utilities through the ArcCatalog and ArcToolbox interfaces.

Creating a feature class “from scratch”

There are two basic ways to create a feature class. You can create a feature class from scratch (requiring considerably more effort), or you can create a feature class from existing data such as a coverage or ESRI shapefile. Both methods are reviewed below with the “from scratch” method being first.

Creating a business table

You may create a business table with either the SQL CREATE TABLE statement or the ArcSDE sdetable command. The sdetable command allows you to include a DBTUNE configuration keyword containing the storage parameters of the table.

Although the table may be up to 256 columns, ArcSDE requires that only one of those columns be defined as a spatial column.

In this example, the sdetable command is used to create the ‘roads’ business table.

```
sdetable -o create -t roads -d 'road_id integer, name string(32), shape
integer' -k roads -u beetle -p bug
```

The table is created using the DBTUNE configuration keyword (-k) ‘roads’ by user beetle.

The same table could be created with a SQL CREATE TABLE statement using the Oracle SQL*Plus interface.

```
create table roads
  (road_id    integer,
   name       varchar(32),
   shape       integer)
   tablespace beetle_data
   storage (initial 16K next 8K);
```

At this point you have created a table in the database. ArcSDE does not yet recognize it as a feature class. The next step is to record the spatial column in the ArcSDE LAYERS and GEOMETRY_COLUMNS system tables and thus add a new feature class to the database.

Adding a feature class

After creating a business table, you must add an entry for the spatial column in the ArcSDE LAYERS system tables before the ArcSDE server can reference it. Use the sdelayer command with the ‘-o add’ operation to add the new feature class.

In the following example, the roads feature class is added to the ArcSDE database. Note that to add the feature class, the roads table name and the spatial column are combined to form a unique feature class reference. To understand the purpose of the –e, –g, and –x options, refer to the sdelayer command reference in *ArcSDE Developer Help*.

```
sdelayer -o add -l roads,shape -e l+ -g 256,0,0 -x 0,0,100 -u beetle -p
bug -k roads
```

If the spatial column of the feature class is stored in either LONG RAW or BLOB ArcSDE compressed binary format, the feature and spatial index tables are created. The feature class tables and indexes are stored according to the storage parameters of the roads configuration keywords in the DBTUNE table. Upon successful completion of the previous sdetable command—to create a table—and the sdelayer command—to record the feature class in the ArcSDE system tables—you have an empty feature class in normal_io mode.

Switching to load-only mode

Switching the feature class to load-only mode drops the spatial index and makes the feature class unavailable to ArcSDE clients. Bulk loading data into the feature class in this state is much faster due to the absence of index maintenance. Use the sdelayer command to switch the feature class to load-only mode by specifying the -o load_only_io operation.

```
sdelayer -o load_only_io -l roads,shape -u beetle -p bug
```

Note: A feature class, registered as multiversioned, cannot be placed in the load-only I/O mode.

Inserting records into the feature class

Once the empty feature class exists, the next step is to populate it with data. There are several ways to insert data into a feature class, but probably the easiest method is to convert an existing shapefile or coverage or import a previously exported ArcSDE sdeexport file directly into the feature class.

In this first example, shp2sde is used with the init operation. The init operation is used on newly created feature classes or can be used on feature classes when you want to overwrite data that's already there. Don't use the init operation on feature classes that already contain data unless you want to remove the existing data. Here, the shapefile, rdshp, will be loaded into the feature class, roads. Note that the name of the spatial column (shape in this case) is included in the feature class (-l) option.

```
shp2sde -o init -l roads,shape -f rdshp -u beetle -p bug
```

Similarly, you can also use the cov2sde command:

```
cov2sde -o init -l roads,shape -f rdcov -u beetle -p bug
```

Switching the table to normal_io mode

After data has been loaded into the feature class, you must switch the feature class to normal_io mode to re-create all indexes and make the feature class available to clients. For example:

```
sdelayer -o normal_io -l roads,shape -u beetle -p bug
```

Versioning your data

Optionally, you may enable your feature class as multiversioned. Versioning is a process that allows multiple representations of your data to exist without requiring duplication or copies of the data. For further information on versioning data, refer to the *Building a Geodatabase* book.

In this example, the feature class “states” will be registered as multiversioned using the sdetable alter_reg operation.

```
sdetable -o alter_reg -t states -c ver_id -C SDE -V multi -k GEOMETRY_TYPE
```

Granting privileges on the data

Once you have the data loaded, it is often necessary for other users to have access to the data for update, query, insert, or delete operations. Initially, only the user who has created the business table has access to it. In order to make the data available to others, the owner of the data must grant privileges to other users. The owner can use the sdelayer command to grant privileges. Privileges can be granted to either another user or to a role.

In this example, a user “beetle” grants SELECT privileges on a feature class “states” to a user “spider”.

```
sdelayer -o grant -l states,feature -U spider -A SELECT -u beetle -p bug
```

The full list of -A keywords are:

SELECT. The user may query the selected object(s) data.

DELETE. The user may delete the selected object(s) data.

UPDATE. The user may modify the selected object(s) data.

INSERT. The user may add new data to the selected object(s) data.

If you include the -I grant option, you also grant the recipient the privilege of granting other users and roles the initial privilege.

Creating and loading feature classes from existing data

The method for creating a schema from scratch and loading it has been reviewed. This next section reviews how to create feature classes from existing data. This method is simpler since the creation and load processes are completed at once.

Each of the ArcSDE administration commands, shp2sde, cov2sde, and sdeimport, includes a “-o create” operation, which allows you to create a new feature class within the ArcSDE database. The create operation does all of the following:

- Creates the business table using the input data as the template for the schema
- Adds the feature class to the ArcSDE system tables
- Puts the feature class into load-only mode

- Inserts data into the feature class
- When all the records are inserted, puts the feature class into normal_io mode

shp2sde

shp2sde converts shapefiles into ArcSDE feature classes. The spatial column definition is read directly from the shapefile. You can use the shpinfo command to display the shapefile column definitions. As part of the create operation, you can specify which spatial storage format you want to adopt for the data storage by including a –k option that references a configuration keyword containing a GEOMETRY_STORAGE parameter.

In this example, Oracle Spatial geometry type is chosen as the storage format by including the configuration keyword SDO_GEOMETRY. The keyword SDO_GEOMETRY, defined in the DBTUNE table, also sets a number of additional parameters for this storage type, which will be used to create the Oracle Spatial index. See Appendix D, 'Oracle Spatial geometry type', for additional information on Oracle Spatial and specifics about how it can be configured.

```
shp2sde -o create -f rdshp -l roads,shape -k SDO_GEOMETRY -u beetle -p bug
```

cov2sde

The cov2sde command converts ArcInfo coverages, ArcInfo Librarian™ library feature classes, and ArcStorm library feature classes into ArcSDE feature classes. The create operation derives the spatial column definition from the coverage's feature attribute table. Use the ArcInfo Workstation "describe" command to display the ArcInfo data source column definitions.

In this example, an ArcStorm library, "roadlib", is converted into the feature class, "roads".

```
cov2sde -o create -l roads,shape -f roadlib,arcstorm -g 256,0,0 -x 0,0,100
-e l+ -u beetle -p bug
```

sdeimport

The sdeimport command converts ArcSDE export files into ArcSDE feature classes. In this example, the roadexp ArcSDE export file is converted into the feature class "roads".

```
sdeimport -o create -l roads,shape -f roadexp -u beetle -p bug
```

After using these commands to create and load data, you may optionally need to enable multiversioning on the feature class and grant privileges on the feature class to other users.

Appending data to an existing feature class

A common requirement for data management is to be able to append data to existing feature classes. The data loading commands described thus far have a –o append operation for appending data. A feature class must exist prior to using the append operation. If the feature class is multiversioned, it must be in an "open" state. It is also advisable to change the feature class to load-only I/O mode and pause the spatial indexing operations before loading the data to improve the data-loading performance. The spatial indexes will be re-created when the feature class is put back into normal I/O mode. Because the feature class has been defined, the metadata exists and is not altered by the append operation.

In the shp2sde example below, a previously created “roads” feature class appends features from a shapefile, “rdshp2”. All existing features, loaded from the “rdshp” shapefile, remain intact, and ArcSDE updates the feature class with the new features from the rdshp2 shapefile.

```
sdelayer -o load_only_io -l roads,shape -u beetle -p bug
shp2sde -o append -f rdshp2 -l roads,shape -u beetle -p bug
sdelayer -o normal_io -l roads,shape -u beetle -p bug
sdetable -o update_dbms_stats -t roads -u beetle -p bug
```

Note the last command in the sequence. The sdetable update_dbms_stats operation updates the table and index statistics required by the Oracle cost-based optimizer. Without the statistics, the optimizer may not be able to select the best execution plan when you query the table. For more information on updating statistics, see Chapter 2, ‘Essential Oracle configuring and tuning’.

Creating and populating raster columns

Raster columns are created from ArcGIS Desktop using ArcCatalog or ArcMap. To create a raster column, you will first need to convert the image file into a format acceptable to ArcSDE. Then, after the image has been converted to the ESRI raster file format, you can convert it into a raster column.

For more information on creating raster columns using either ArcCatalog or ArcToolbox, refer to *Building a Geodatabase.*

To estimate the size of your raster data, refer to Appendix A, ‘Estimating the size of your tables and indexes’.

To understand how ArcSDE stores rasters in Oracle, refer to Appendix B, ‘Storing raster data’.

Creating views

There are times when a DBMS view is required in your database schema. ArcSDE provides the sdetable create_view operation to accommodate this need. The view creation is much like any other Oracle view creation. If you want to create a view using a layer and you want the resulting view to appear as a feature class to client applications, include the feature class’s spatial column in the view definition. As with the other ArcSDE commands, see *ArcSDE Developer Help* for more information.

Exporting data

As with importing data, there are also client applications that export data from ArcSDE. With ArcSDE, the following command line tools exist:

sdeexport—creates an ArcSDE export file to easily move feature class data between Oracle instances and to other supported DBMSs

sde2shp—creates an ESRI shapefile from an ArcSDE feature class

sde2cov—creates a coverage from an ArcSDE feature class

sde2tbl—creates a dBASE or INFO file from a DBMS table

Schema modification

There will be occasions when it is necessary to modify the schema of some tables. You may need to add or remove columns from a table. The ArcSDE command to do this is sdetable with the –o alter option. ArcCatalog offers an easy-to-use tool for this and other schema operations such as modifying the spatial index (grids) and adding and dropping column indexes.

Choosing an ArcSDE log file configuration

ArcSDE allows you to configure the allocation of ArcSDE log files to your users. You can allow your users to own their own log files, or they can check out a log file from a pool of log files owned by the SDE user. Log files can be shared, session-based, or standalone. A shared log file is the default and is used by all sessions that connect as a given user. Also, if the ArcSDE server is configured to use standalone log files and all available log files of this type are exhausted, ArcSDE will attempt to create a session-based log file if allowed; otherwise a shared log file will be created. If the shared log file cannot be created, ArcSDE returns an error.

Shared ArcSDE log files

Shared log files are shared by all sessions that connect as the same user. Essentially, all sessions are inserting and deleting records from the same log file data table. The log files are created the first time any session connects and remains in the user's schema. To configure your server to use only shared log files, set the log file server configuration parameters as follows:

```
MAXSTANDALONELOGS      0
ALLOWSESSIONLOGFILE    FALSE
```

Session-based ArcSDE log files

For session-based log files, each session that connects to the server creates a log file.

A session-based log file is dropped when a session disconnects. To configure your server to use session-based log files, set the server configuration parameter ALLOWSESSIONLOGFILE to true.

```
ALLOWSESSIONLOGFILE    TRUE
```

You need to make sure that you configure enough space for the tables and indexes of the session-based log files. The DBTUNE SESSION_STORAGE and SESSION_INDEX storage parameters control the storage of session-based log files.

Standalone ArcSDE log files

Standalone log files are created by a session for each log file the application needs to store. When an application deletes the log file, the standalone log file is truncated. The standalone log files are dropped when the session disconnects. To configure your server to use standalone log files, set the server configuration parameter MAXSTANDALONELOGS to the number of standalone log files you want them to be able to create.

For instance, set MAXSTANDALONELOGS to 6 if you want to allow each ArcSDE session to create a maximum of six standalone log files:

```
MAXSTANDALONELOGS 6
```

Keep in mind that you need to configure enough space to store all of these log files. The DBTUNE parameters, SESSION_STORAGE and SESSION_INDEX, allocate space for the tables and indexes of standalone log files.

If the application exhausts the number of allowable standalone log files—if the application needs to simultaneously create more logical log files than MAXSTANDALONELOGS allows—ArcSDE will attempt to create a session-based log file, but only if ALLOWSESSIONLOGFILE is set to TRUE; otherwise ArcSDE will use a shared log file. The shared log file is created if it does not already exist. If the shared log file cannot be created, ArcSDE returns an error.

Using an SDE user pool of ArcSDE log files

The SDE user can create a pool of log files that can be checked out and used as either session-based or standalone log files by other users. Using a pool of SDE log files avoids having to grant users CREATE TABLE privileges. Shared log files cannot be checked out from an SDE log file pool.

To create a pool of log files, set the configuration parameter LOGPOOLSIZE to the number of log files that need to be created. This number should reflect the number of sessions that will connect to your server in addition to the standalone log files, if allowed. To calculate the total number of log files that can be checked out of the pool, use the following formulas:

If session log files are allowed but not standalone log files:

```
LOGPOOLSIZE = total sessions expected
```

If standalone log files are allowed, but not session log files:

```
LOGPOOLSIZE = MAXSTANDALONELOGS * total sessions expected
```

If both standalone log files and session log files are allowed:

```
LOGPOOLSIZE = (MAXSTANDALONELOGS + 1) * total sessions expected
```

For instance, if you compute that 100 log files are needed, the LOGPOOLSIZE parameter would be set as follows:

```
LOGPOOLSIZE 100
```

If the pool is exhausted and another log file is needed, ArcSDE will attempt to create it in the user's schema. If the log file cannot be created, an error is returned.

The pooled log file tables are created or dropped whenever the LOGFILESIZE parameter is changed.

Set the HOLDLOGPOOLTABLES server configuration parameter to TRUE if you want the sessions to retain checked-out log files. If set to FALSE, the log files are released whenever the application deletes all of its log files, as in the case of a session log file, or whenever the log file occupying a standalone log file is deleted.

The storage of the tables and indexes of the log file pool is controlled by the DBTUNE storage parameters SESSION_STORAGE and SESSION_INDEX.

Using the ArcGIS Desktop applications

So far the discussion has focused on ArcSDE command line tools that create feature class schemas and load data into them. While robust, these commands can be daunting for the first-time user. In addition, if you are using ArcGIS Desktop, you may have to use ArcCatalog to create feature datasets and feature classes within those feature datasets to use specific ArcGIS Desktop functionality. For that reason, a glimpse of how to use ArcCatalog and other ArcGIS tools to load data is provided. Refer to the ArcGIS Desktop documentation for a full discussion of these tools.

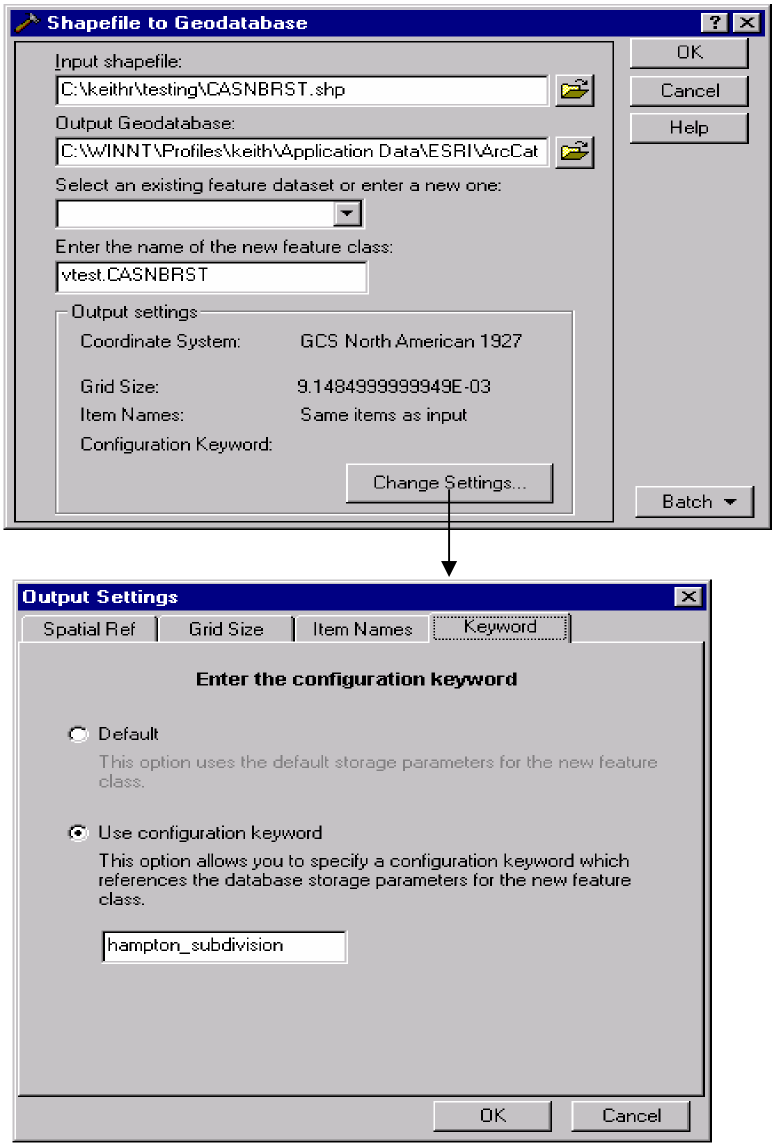
Loading data

You can convert ESRI shapefiles, coverages, Map LIBRARIAN layers, and ArcStorm layers into geodatabase feature classes with the ArcToolbox and ArcCatalog applications. ArcToolbox provides a number of tools that enable you to convert data from one format to another.

ArcToolbox operations, such as the ArcSDE administration commands shp2sde, cov2sde, and sdeimport, accept configuration keywords. By using a configuration keyword with the GEOMETRY_STORAGE parameter, the user can choose one of the three Oracle spatial storage methods supported by ArcSDE.

In the ArcToolbox Shapefile to Geodatabase wizard, you can see that a configuration keyword has been specified for the loading of the hampton_streets shapefile into the geodatabase. Since the geodatabase is maintained by an ArcSDE service operating on an

Oracle database, you can store the resulting feature class using the Oracle Spatial geometry type format. This keyword has set the GEOMETRY_STORAGE parameter to SDO_GEOMETRY, indicating to ArcSDE that the resulting feature class should be stored as an Oracle Spatial geometry type.



Shapefile to Geodatabase

Input shapefile: C:\keithr\testing\CASNBRST.shp

Output Geodatabase: C:\WINNT\Profiles\keith\Application Data\ESRI\ArcCat

Select an existing feature dataset or enter a new one:

Enter the name of the new feature class: vtest.CASNBRST

Output settings

Coordinate System:	GCS North American 1927
Grid Size:	9.14849999999949E-03
Item Names:	Same items as input
Configuration Keyword:	

Output Settings

Enter the configuration keyword

Default [✓ Use configuration keyword]

Default: This option uses the default storage parameters for the new feature class.

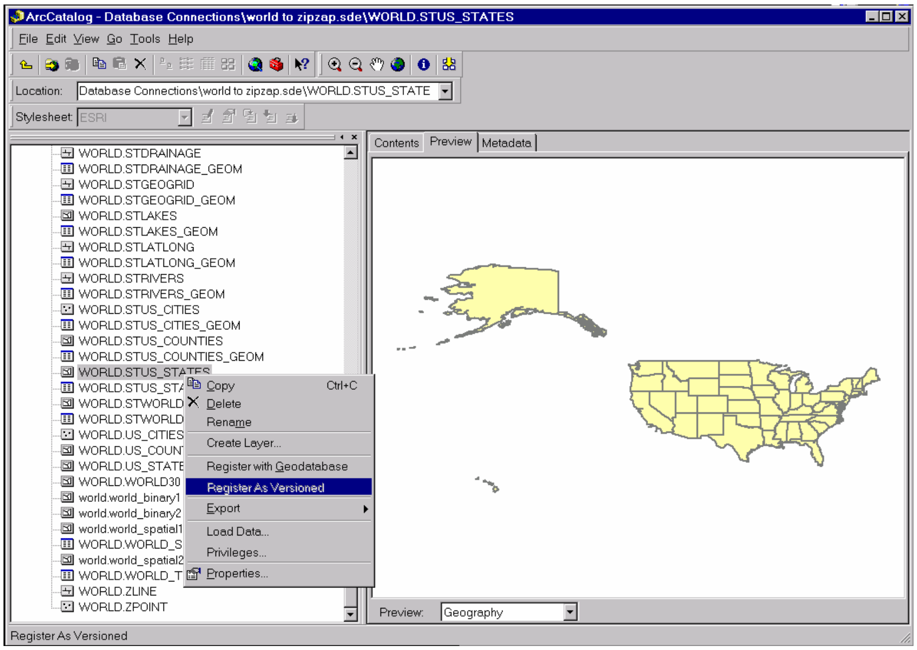
Use configuration keyword: This option allows you to specify a configuration keyword which references the database storage parameters for the new feature class.

hampton_subdivision

The shapefile CASNBRST.shp is converted to feature class vtest.CASNBRST using Toolbox.

Versioning your data

ArcCatalog also provides a means for registering data as multiversioned. Simply right-click the feature class to be registered as multiversioned and click the Register As Versioned context menu item.


ArcCatalog - Database Connections\world to zipzap.sde\WORLD.STUS_STATES
File Edit View Go Tools Help
Contents Preview Metadata
Copy Ctrl+C
Delete
Rename
Create Layer...
Register with Geodatabase
Register As Versioned
Export
Load Data...
Privileges...
Properties...

Location: Database Connections\world to zipzap.sde\WORLD.STUS_STATE

Stylesheet: ESRI

WORLD.STDRAINAGE
WORLD.STDRAINAGE_GEOM
WORLD.STGEOGRID
WORLD.STGEOGRID_GEOM
WORLD.STLAKES
WORLD.STLAKES_GEOM
WORLD.STLATLONG
WORLD.STLATLONG_GEOM
WORLD.STRIVERS
WORLD.STRIVERS_GEOM
WORLD.STUS_CITIES
WORLD.STUS_CITIES_GEOM
WORLD.STUS_COUNTIES
WORLD.STUS_COUNTIES_GEOM
WORLD.STUS_STATES
WORLD.STUS_STA
WORLD.STWORLD
WORLD.STWORLD
WORLD.US_CITIES
WORLD.US_COUN
WORLD.US_STATE
WORLD.WORLD30
world.world_binary1
world.world_binary2
world.world_spatial1
WORLD.WORLD_S
world.world_spatial2
WORLD.WORLD_T
WORLD.ZLINE
WORLD.ZPOINT

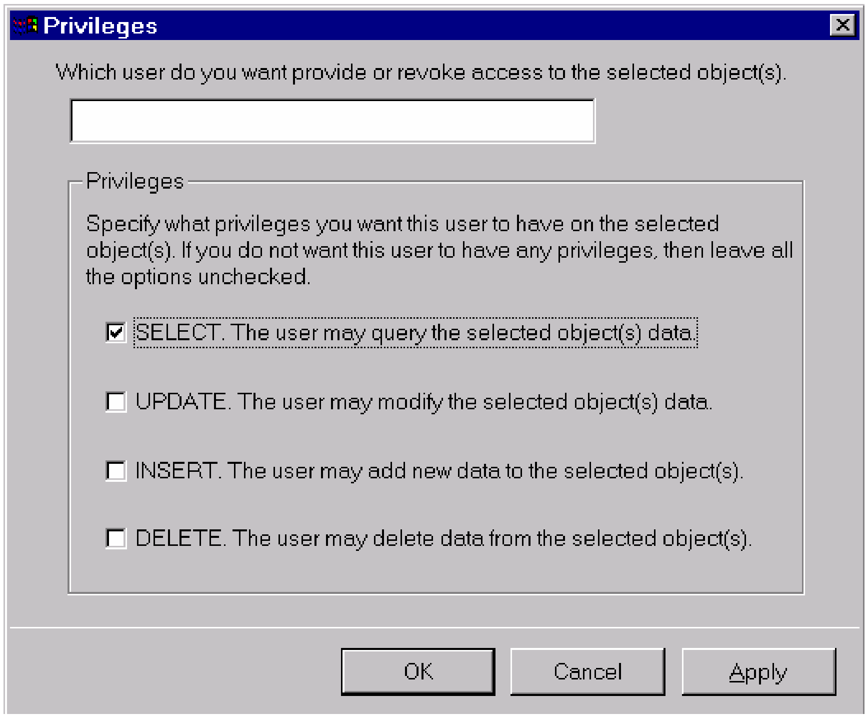
Preview: Geography

Register As Versioned

A feature class is registered as multiversioned from within ArcCatalog.

Granting privileges

Using ArcCatalog, right-click the data object class and click the Privileges context menu. From the Privileges context menu, assign privileges specifying the username and the privilege you want to grant to or revoke from a particular user.



Privileges

Which user do you want provide or revoke access to the selected object(s).

Privileges

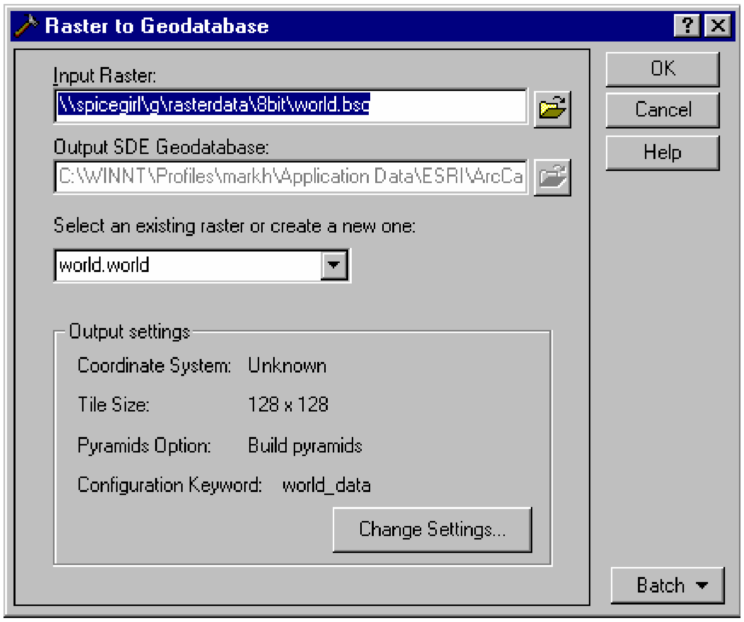
Specify what privileges you want this user to have on the selected object(s). If you do not want this user to have any privileges, then leave all the options unchecked.

- [✓] SELECT. The user may query the selected object(s) data.
- [ ] UPDATE. The user may modify the selected object(s) data.
- [ ] INSERT. The user may add new data to the selected object(s).
- [ ] DELETE. The user may delete data from the selected object(s).

The ArcCatalog Privileges dialog box allows the owner of an object class, such as a feature dataset, feature class, or table, to assign privileges to other users or roles.

Creating a raster column with ArcCatalog

Using ArcCatalog, right-click the database connection, point to Import, and click Raster to Geodatabase. Navigate to the raster file to import. Click Change Settings if you want to change the coordinate reference system, tile size, pyramids option, or configuration keyword. Click OK to import the raster file into the Oracle database.



Raster to Geodatabase

Input Raster: \\spicegirl\g\rasterdata\8bit\world.bsq

Output SDE Geodatabase: C:\WINNT\Profiles\markh\Application Data\ESRI\ArcCa

Select an existing raster or create a new one: world.world

Output settings

Coordinate System: Unknown

Tile Size: 128 x 128

Pyramids Option: Build pyramids

Configuration Keyword: world_data

Efficiently registering large business tables with ArcSDE

When you create a business table in an ArcSDE database using an ArcSDE client application, for example, ArcCatalog or sdetable, ArcSDE registers the business table in the SDE.TABLE_REGISTRY. During the registration process, ArcSDE performs a number of tasks depending on the type of registration requested. The duration of the registration process is dependent on the type of registration, whether the business table has a spatial column, and the table's number of rows. For large business tables, the registration process can take a long time to complete. This section provides the most efficient method to register tables with a large number of records.

Registering a table as NONE or USER maintained

Tables registered as NONE are registered without a row ID column.

Tables registered as USER are registered with a row ID column whose values you must maintain.

If the registration type is NONE or USER, ArcSDE merely adds a record to the SDE.TABLE_REGISTRY that references the business table. For tables registered as type USER the name of the row ID is also added to the SDE.TABLE_REGISTRY entry. In the case of user maintained row IDs, ArcSDE will ensure that the column exists in the business table before completing the registration.

The preceding two registration types complete the registration process rather quickly.

Registering a table with an SDE maintained row ID

Tables registered with an SDE maintained row ID must have a row ID column that uniquely identifies the rows of the table.

Note: Tables registered by the geodatabase must be registered by ArcSDE with an SDE maintained row ID. If the geodatabase determines that the table has been registered with an SDE maintained row ID, the geodatabase registration process is relatively inexpensive.

If a table was registered with a USER maintained row ID, the geodatabase alters its row ID registration to be SDE maintained.

By default, the geodatabase adds a column called objectid to the table and registers it as an SDE maintained row ID. If the objectid column already exists and is not currently registered as SDE maintained, the geodatabase will add a new column to the table called objectid_1.

Creating a new SDE maintained row ID column

If the row ID column does not exist when the table is registered, ArcSDE adds a column of type INTEGER, with a NOT NULL constraint. If the table contains rows, ArcSDE populates the column with unique ascending values starting at your specified minimum ID value. The minimum ID value defaults to 1 if left unspecified. It then creates a unique index on the column R<registration_id>_SDE_ROWID_UK, where registration_id is the registration identifier ArcSDE assigns the table when it was registered.

ArcSDE creates a sequence generator called R<registration_id> and uses it to generate the next value of the row ID column whenever a value is added to the column.

Adding a column to an Oracle table that already contains rows can result in row migration, when the free space of the Oracle data block (as defined by PCTFREE) is consumed by the newly added column values.

Therefore, if at all possible, it is better to include the row ID column as part of the table’s original definition, or add it before data is inserted into the table. If, however, you must add the row ID column to a table that contains data, consider exporting the table afterwards, truncating the rows, and importing the data back into the table. Doing so eliminates the excessive disk I/O generated by fetching the migrated rows.

Using an existing column

If the row ID column already exists, ArcSDE confirms that the column was defined as an integer. If it does not, the registration fails.

Next, ArcSDE confirms that the column has a unique index. If the column was defined with a nonunique index, ArcSDE drops the index.

In the event that the column does not have a unique index, ArcSDE attempts to create a unique index on the column. If the index creation fails because the column contains non-unique values, ArcSDE repopulates the column with ascending values, beginning at 1, and creates the unique index. ArcSDE names the unique index R<registration_id>_SDE_ROWID_UK.

Next, ArcSDE verifies that the column has been defined as NOT NULL.

If the column was defined as NULL, ArcSDE attempts to redefine it as NOT NULL. If this action fails, ArcSDE repopulates the column and defines it as NOT NULL.

Repopulating the column either because it contained null values or because it contained nonunique values is an expensive process, especially if the table contains more than 100,000 records.

Therefore, don't rely on ArcSDE to perform this operation. Instead, define the row ID column as not null when the table is created and create your own unique index on it. At the very least, you should ensure that the column is populated with unique integer values.

Registering a table as multiversioned

To perform versioned edits on a business table, the table must be registered as multiversioned. These tables, as the name implies, store the records of the business that are added and deleted. They are named A<registration_id> and D<registration_id>. When tables are registered as multiversioned, the associated adds and deletes tables are created, and along with the business table, ArcSDE updates their DBMS statistics.

For the business table, ArcSDE queries the LAST_ANALYZED value from the USER_TABLES view to determine if it has DBMS statistics. If the value is NULL, ArcSDE generates statistics for the table with COMPUTE STATISTICS.

Note: This is an expensive operation for tables with more than a million rows. Therefore, to speed up the registration process, you should analyze large tables before you register them as multiversioned. For tables containing approximately 100,000 rows or more, sampling from 1 to 10 percent of the rows should be adequate. The larger the table, the smaller the sampling percentage can safely be.

If the business table contains a spatial column that is stored as binary (see Appendix C for more information), the feature table and spatial index table must also have statistics. ArcSDE also queries the LAST_ANALYZED value of the USER_TABLES view for these tables.

If the value is not null, these tables and their indexes are analyzed.

Again, if they are very large (more than 100,000 rows), you should analyze them yourself using a percentage of the total number of records.

Use the Oracle ANALYZE statement to analyze the tables.

```
ANALYZE TABLE <table_name>
ESTIMATE STATISTICS SAMPLE <percent_value> PERCENT;
```

Under cost-based optimization, Oracle uses the statistics of the tables it is querying to select the optimal execution plan.

Note: The registration of an SDE maintained row ID column is the only time ArcSDE automatically updates the statistics of the business, adds and deletes tables and their indexes as well as the feature table and the spatial index table, if they exist.

CHAPTER 5

National language support

Storing and retrieving data in an ArcSDE Oracle database using character sets other than the Oracle default requires some extra configuration on both the client and the server. This section provides guidelines for configuring both the Oracle database and the ArcSDE client environment, to enable the use of character sets other than the default.

Oracle database character sets

If you are using UNIX, by default an Oracle database is created with a US7ASCII character set. On Windows, the default character set is WE8ISO8859P1. The character set is selected when the database is created with the CREATE DATABASE statement and cannot be changed afterward. To change a character set, the database must be re-created and the data reloaded. Consult the *Oracle National Language Support Guide* for your Oracle release to determine the character set that is right for your data. References to *character sets* in Oracle documentation are synonymous with the ArcSDE term *character encoding*.

Setting the NLS_LANG variable on the client

Once the ArcSDE Oracle database has been created with the desired character encoding, data can be loaded into it using a variety of applications such as ArcGIS Desktop and the ArcSDE administration tools shp2sde and cov2sde as well as applications written with the ArcSDE C and Java API. To properly convert and preserve the characters, you must set the Oracle NLS_LANG variable in the client application's system environment.

For instance, if the Oracle database, installed on Windows, has been created with the Shift Japanese industrial standard (JIS), Oracle notation ja16sjis, character encoding and you want to access this database from a UNIX client running in the Extended UNIX character Japanese (EUC JP), Oracle notation ja16euc, character encoding, you will have to set the NLS_LANG variable to ja16euc. This ensures that all character data transferred between the ArcSDE server and the ArcSDE client uses the ja16euc character encoding.

For C API client applications, set the NLS_LANG as an environment variable.

```
setenv NLS_LANG Japanese_Japan.ja16euc
```

For Java API client applications, set the NLS_LANG as a Java Virtual Machine (JVM) system property.

```
java -DNLS_LANG= Japanese_Japan.ja16euc [Java program name]
```

If the situation is reversed, and the Oracle database is on UNIX and created with the EUC JP character encoding but the client is running on Windows, you will have to set the NLS_LANG environment variable on the Windows client to ja16sjis.

For the Windows clients, such as ArcCatalog or ArcMap, open the Control Panel. Double-click the System icon. For Windows 2000 and lower, select the Environment tab. Click the System Variables scrolling list and enter NLS_LANG in the Variable: input line and Japanese_Japan.ja16sjis in the Value: input line. Click Set, then OK. For Windows XP and higher, select the Advanced tab and click on the Environment Variables button. Under the System Variables list, click New. Enter NLS_LANG in the Variable: input line and Japanese_Japan.ja16sjis in the Value: input line and click OK.

For C API and ArcSDE client utility applications, such as shp2sde, that run from the DOS environment, set the NLS_LANG as an environment variable.

```
set NLS_LANG=Japanese_Japan.ja16sjis
```

For Java API client applications, set the NLS_LANG as a JVM system property.

```
java -DNLS_LANG= Japanese_Japan.ja16sjis [Java client program]
```

Setting the NLS_LANG variable for Windows clients

Be careful setting the NLS_LANG on Windows because there are actually two different code page environments on this platform. The character encoding standards supported by the Windows environment are different from those supported by the MS–DOS environment. Windows applications, such as ArcGIS Desktop, run in the Windows American National Standards Institute (ANSI) code page environment, while ArcSDE administration tools and C and Java API applications invoked from the MS–DOS Command Prompt run in the original equipment manufacturer (OEM) code page environment. Some languages require two different NLS_LANG settings for the language character set component for each of these code page environments.

Setting the NLS_LANG variable for a remote ArcSDE setup

If you use direct connect to connect to an Oracle server from a remote computer or if you start the ArcSDE application server on a computer that is remote from the Oracle server, you will need to set the NLS_LANG variable for ArcSDE.

In the case of the direct connect client, that NLS_LANG variable must be set in the environment. If you are using an ArcSDE administration tool, such as shp2sde, that is started from an MS–DOS command tool, set the NLS_LANG variable in the MS–DOS environment before executing the command.

Direct connections from a Windows application, such as ArcMap, require that the NLS_LANG variable be set in the Windows environment. Open the Control Panel and double-click the System icon. For Windows 2000 and lower, select the Environment tab. Click the System Variables scrolling list and enter NLS_LANG in the Variable: input line and Japanese_Japan.ja16sjis in the Value: input line. Click Set, then OK. For Windows XP and higher, select the Advanced tab and click on the Environment Variables button. Under the System Variables list, click New. Enter NLS_LANG in the Variable: input line and Japanese_Japan.ja16sjis in the Value: input line and click OK.

To set the NLS_LANG for a remote application server, set the variable in the dbinit.sde file. When the application server is started, the variable is read from the file. To ensure that the variable is set in the ArcSDE application server environment, check the ArcSDE application server's variable settings with the ArcSDE sdemon –o info –I vars command.

The following ArcSDE/Oracle configuration illustrates how the NLS_LANG variable should be set in a remote setup. Consider the case where the language environment is Eastern European, Oracle is installed on a UNIX server, the ArcSDE application server is running on a Windows server, and a user is connecting from Windows on yet another computer. The Oracle database was created with the iso.8859-2 UNIX ISO code page. Before the administrator starts the ArcSDE application server, the following Windows ANSI cp1250 code page NLS_LANG variable must be added to the dbinit.sde file:

```
set NLS_LANG=SLOVAK_SLOVAKIA.EE8ISO8859P2
```

The user connects to the application server with ArcMap, but before doing so sets the NLS_LANG variable to the Windows ANSI cp1250 code page value as:

```
NLS_LANG=SLOVAK_SLOVAKIA.EE8ISO8859P2
```

The user wishes to use SQL*Plus from an MS–DOS prompt to query a table. In the command window the user enters the following DOS OEM 852 NLS_LANG variable:

```
set NLS_LANG=SLOVAK_SLOVAKIA.EE8PC852
```

Setting the NLS_LANG variable for Java clients

ArcSDE—Java API client-based applications, run from the DOS command window, require setting the file.encoding Java System property if the DOS OEM file.encoding property is different from the file.encoding used by applications run in the Windows environment. Consider, for example, that the Windows client machine is set up in the French Locale (CP1252) and that the Oracle database contains French data, created with the NLS_CHARACTERSET parameter set to WE8MSWIN1252. The OEM codepage required to read this data is CP850. To read and display all French data, the file.encoding Java System Property must be set at the DOS command prompt as shown below:

```
java -Dfile.encoding=CP850 -DNLS_LANG=French_France.we8pc850 <Java program
name>
```

Setting the NLS_LANG for the ArcSDE server

For Windows, the NLS_LANG variable setting must be registered before the ArcSDE server is started.

```
sdeservice -o register -r NLS_LANG -v <value> -p <DB_Admin_password> [-i
<service>]
```

On UNIX, the NLS_LANG variable must be set as an environment variable before the ArcSDE server is started, preferably in the dbinit.sde file.

Character encoding standards supported by ArcSDE

For a complete list of character encoding standards supported by your Oracle database and their naming conventions, please refer to the *Oracle National Language Support Guide*.

CHAPTER 6

Backup and recovery

Data protection is ensured by implementing, testing, and verifying an appropriate database backup and recovery procedure. Oracle provides a number of choices for backing up a database including exporting data, selective backup of tablespaces, or copying database files.

This chapter presents a summary of the Oracle concepts that are essential for understanding backup and recovery, plus specific guidelines that may be applied to your ArcSDE installation.

For details, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Recording database changes

Oracle manages changes to its contents and structure in such a manner as to guarantee recovery of the database to the last committed transaction after any single point of failure. This guarantee is at the heart of Oracle's architecture, and this statement says a great deal.

To the last committed transaction means that once control is returned to the user after executing a COMMIT statement, Oracle guarantees that the committed data has been written to disk in some form and is recoverable.

Single point of failure means that any single file or process can fail without losing the contents of any committed transaction. If a data file is lost or corrupted, the contents of the redo logs guarantee that the data is recoverable. If a control file breaks, the other control files ensure that the information remains safe. A process may be killed, but committed data is never lost.

Such a guarantee requires a high level of coordination between database processes and database files. Files “know about” each other and may not be deleted or changed without properly coordinating through the Oracle instance.

This section briefly describes the role of the various Oracle database files in backup and recovery to achieve this guarantee.

Control files

The physical structure of the database—the arrangement of physical files on disk—is recorded in the *control file*. The control file is so important for the purpose of database consistency that Oracle maintains multiple identical copies of it.

If the database loses one of its control files, the database will halt. You will have to do a SHUTDOWN ABORT on the database instance prior to recovering the database.

Oracle recommends that any database instance should use *at least three* identical control files and further recommends that each control file be placed on a physically separate disk drive under a separate disk controller. While a single hardware failure may cause the loss of one control file, this strategy ensures the safety of the remaining control files.

The locations and names of the control files are controlled by the init.ora parameter:

CONTROL_FILES

Note: If you add or drop a data file or an online redo log file or alter file locations, the contents of the control file change as this information is recorded. Since the backup version of the control file no longer matches the current database structure, you should take an immediate backup of the new control file.

Redo log files

Every change that is made to an Oracle database—such as creating or altering schema objects or inserting or manipulating data—is recorded in the *online redo log file*. This happens automatically in addition to the normal action of writing the changes to the appropriate *data file* or control file.

Changes to the database are recorded in the order that they occur. If, for example, you create a table to store GIS data, then insert some data into that table, the create table action will be recorded first, and the insert data action will be recorded second.

Recording the order of the actions is important, and individual steps may not be neglected. In this example, if the table creation step is skipped or performed at the wrong time, the data insertion step will fail.

Eventually, the database change information recorded in a redo log file accumulates to some preset maximum. Oracle then performs a *log switch*, closing the current online redo log file and opening the next one.

Oracle uses the online redo log files in a circular fashion, reusing the first log file after it closes the last log file. If three online log files are specified, for example, Oracle will use the log files in order: 1, 2, 3, 1, 2, 3, 1, … This means that earlier information stored in any of the online redo log files will be lost when the file is used again.

Archiving redo log files

Oracle provides for the online redo log files to be copied to an offline location immediately after a log switch. This is called *archiving* the redo log files.

When an Oracle database is run in ARCHIVELOG mode, redo log information is accumulated indefinitely by copying the online redo log file to an offline location with a unique name based on the current *log sequence number*.

For example, as Oracle switches through online redo log files in order (1, 2, 3, 1, 2,…) the following archived redo log files (reflecting log sequence numbers 15, 16, 17, 18, 19, …) might be created:

```
LOG0015.ARC
LOG0016.ARC
LOG0017.ARC
LOG0018.ARC
LOG0019.ARC
```

In ARCHIVELOG mode, the total amount of redo log information is limited only by the total file space available at the archive destination. When an Oracle database is run in NOARCHIVELOG mode, the online redo log files are not copied so the amount of usable redo log information is limited to the contents of the current online redo log files.

Which archive mode should you use?

Often databases are created and initially loaded under the NOARCHIVELOG mode because, should a media failure occur, the same effort is required to restore the database regardless of whether the database was reloaded using the archived redo log files or the original source data. In addition, a huge amount of log file data is created, and maintaining this data is unnecessary since the original source data is readily available.

After the initial data load is complete and the database goes into production, the changes become difficult to re-create. Switch to ARCHIVELOG mode to ensure that point-in-time recovery of the database is available in the event of a media failure.

Operate the database in NOARCHIVELOG mode and make frequent backups *only* if you understand the risk of data loss and can afford to reenter the changes made since the last backup.

Note: Because of the risk of losing committed transactions in the event of media failure, ESRI does *not* recommend using NOARCHIVELOG mode for normal operation of an ArcSDE database.

Switching from NOARCHIVELOG to ARCHIVELOG

An Oracle database created in NOARCHIVELOG mode can be switched to ARCHIVELOG mode. To make this change, shut down the database using the NORMAL, IMMEDIATE, or TRANSACTIONAL priority. (Do not use ABORT.)

```
SQL> connect sys/<password> as sysdba
Connected.
SQL> shutdown normal
Database closed.
Database dismounted.
Oracle instance shut down.
```

Before switching to ARCHIVELOG mode, Oracle recommends that you make a full backup of the database.

To the init.ora file, add the LOG_ARCHIVE_START and LOG_ARCHIVE_DEST parameters.

```
LOG_ARCHIVE_START = TRUE
LOG_ARCHIVE_DEST = <pathname_to_archives>/arch%t_%s.dbf
```

Setting LOG_ARCHIVE_START to true causes the ARCH process to archive the online redo log files automatically. If you do not set this parameter or set it to false, you will have to manually archive the log files. For information on how to perform a manual archive, consult the *Oracle Server Administrator's Guide* for your Oracle release.

After you have backed up the Oracle database, switch the database to ARCHIVELOG mode:

```
C:\>sqlplus

SQL*Plus: Release 9.2.0.3.0 - Production on Thu Jul 10 18:55:38 2003

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Enter user-name: sys/manager as sysdba
```

```
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.3.0 - Production
With the OLAP and Oracle Data Mining options
JServer Release 9.2.0.3.0 - Production

SQL> startup mount
Total System Global Area          245317008 bytes
Fixed Size                            64912 bytes
Variable Size                     103677952 bytes
Database Buffers                  131072000 bytes
Redo Buffers                       10502144 bytes
Database mounted.
SQL> alter database archivelog;
Statement Processes
SQL> alter database open;
Statement processed.
SQL>
```

Note: After the database successfully reopens in ARCHIVELOG mode, you should shut it down again and make another full backup. If you don't, your last full backup will have recorded that the database was in NOARCHIVELOG mode, and you will not be able to apply the online redo log files following the restoration of the backup.

Sizing the redo log files

If you have created small online redo log files, you should consider creating new, larger redo log files.

Every time that Oracle switches the log file, it first performs a *checkpoint* in which the contents of changed blocks within the database buffer cache are written to disk. Changes to the database are always suspended while checkpoint processing is taking place.

In a large multiuser system where many changes are being made, the checkpoint process may involve practically the entire database buffer cache and should, therefore, be put off for as long as possible. Where the number of database changes is small, checkpoint processing will be insignificant.

If the database is configured with only three large online redo log files without mirroring—the default configuration—you should create more log file groups and mirror them before you switch the database to ARCHIVELOG mode.

Additional log file groups help smooth out the archiving process by enabling the archive process to finish copying a log file to the archive destination before the database needs to make the log file current again. If the database must make a log file current before the archive process has finished copying it, the database will wait until the archive is finished before continuing processing.

Refer to Chapter 2, ‘Essential Oracle configuring and tuning’ for guidance on configuring the online redo logs.

For details on commands used to manage redo log files, refer to the *Administrator’s Guide* for your Oracle release.

Database backup

To back up your ArcSDE Oracle database, you must copy these files to an offline location:

- Control files
- Datafiles
- giomgr.defs, dbinit.sde, and services.sde files
- Archived redo log files

You should make at least three copies of the control files with each backup because of their importance in ensuring database consistency. Because control files are comparatively small, there is negligible cost in doing so.

Remember that the redo log files are essential for bringing the data files from an earlier state to a later state. Between any two points in time, the redo logs must be found in an *unbroken* sequence if database recovery is to succeed.

ESRI recommends that you maintain *at least two* copies of all archived redo logs for as far back in time as may be reasonably necessary for database recovery. The two copies should be stored on physically distinct media—separate disk drives, for example, or a disk drive and a tape drive.

If you intend to purge the archived redo log files from their location on disk, be sure that you have a *second* backup copy of each archived redo log file before purging.

This strategy of multiple backups of the archived redo log files helps guard against multiple media failure, which is not as rare as it might seem. Some tape drives, for example, fail to detect bit errors until you attempt to restore a file, when it may be too late.

You only need to make a single copy of each data file with each backup as long as you carefully maintain multiple copies of archived redo logs.

Note: The online redo log files are *not* necessary for backup. If the current online redo log file fails, committed information still exists in memory, which Oracle writes to the data files when a checkpoint is issued. Oracle issues a checkpoint automatically when you shut down a database instance with NORMAL, IMMEDIATE, or TRANSACTIONAL priority. Before shutting down a database with ABORT priority, you should force a checkpoint with the ALTER SYSTEM … CHECKPOINT command if at all possible.

Hot backup

Backing up an Oracle database while the database instance is running is called a *hot backup*. If you plan to take a hot backup, you must operate your database in ARCHIVELOG mode.

Enter the ALTER TABLESPACE … BEGIN BACKUP command prior to the backup of *each* tablespace, which tells Oracle that a hot backup is being taken. If this command is not issued, the hot backup will appear to succeed but it may be useless for restoring the database. To finish the hot backup, for each tablespace enter the ALTER TABLESPACE … END BACKUP command.

Changes to data are recorded and held in the rollback segment until they are no longer needed by any outstanding transaction. Taking a hot backup prevents the release of rollback segment data until the ALTER TABLESPACE … END BACKUP command is issued.

Therefore, the rollback segment must be large enough to accommodate changes made during the hot backup. If the rollback segment runs out of space, a transaction will fail with an ORA-1555 error:

```
ORA-1555: snapshot too old (rollback segment too small)
```

While the hot backup will succeed despite this error, changes made to the database may need to be reentered.

You may avoid this error by taking a hot backup during times of low database activity or by ensuring that your rollback segments are large enough to accommodate data changes made during backup.

You do not need to shut down the ArcSDE server process (giomgr) prior to making a hot backup.

For details on hot backup, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Cold backup

Backing up an Oracle database while the database instance is shut down is called a *cold backup*. Managing a cold backup is simpler than a hot backup and less prone to error.

If you run the database in NOARCHIVELOG mode, a cold backup is your only option. Running the database in ARCHIVELOG mode will enable you to use a cold backup to recover a database to the latest committed transaction.

Database export

You can use Oracle's export utility to supplement a full backup. If changes are made to a known set of data objects between full backups, you can export the objects.

However, the export utility should only be used on data objects that are not changing during the export and on all data objects that are closely related. For example, if you use export to back up a business table, you should also capture the related spatial index, feature, and delta tables in the same backup.

ESRI does *not* recommend that you use the export utility as your only backup method.

You can also back up the entire Oracle database with the Oracle export utility, then make cumulative and incremental backups.

For more information on the export utility, refer to the *Oracle Utilities* manual for your Oracle release.

Frequency of backups

Base the frequency of your backups on the rate at which the data in your database is changing. The more changes that occur, the more frequently backups should occur.

If your Oracle database is operating under ARCHIVELOG mode, you have several variations that you may add to your backup strategy in addition to the periodic full backup. Databases operating under the NOARCHIVELOG mode are restricted to full backups with the possible addition of Oracle export files.

For more information on different Oracle database backup strategies, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Regardless of which archiving mode you are using, you should make regular full backups of the Oracle database. A full backup should include the Oracle database and, if you have started an ArcSDE service, the giomgr.defs, dbinit.sde, and services.sde files.

Scripting the backup

Once you have decided on your backup strategy, the commands that are entered through the system prompt or through SQL*Plus may often be scripted.

ESRI recommends using a script for backup purposes whenever possible.

Executing the backup commands from a UNIX shell script or a Windows batch file will help ensure that the database is being backed up consistently and that all intended steps are being followed.

Be sure to update the script whenever the physical structure of the database changes, for instance, whenever you add a data file or move a redo log file.

For examples of backup scripts, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Database recovery

To recover a database after any failure, Oracle takes these steps:

1. Reads the init.ora file to determine the names and locations of the control files.
2. Reads the control files to verify their consistency with each other and to determine the physical file structure of the database.
3. Opens each data file mentioned in the control file to determine whether that data file is current and reflects the latest committed change or is in need of recovery.
4. Opens each redo log file *in sequence* and applies the information found there to each data file, as necessary, to bring each data file to the state where it contains all its committed transactions.

It is possible to recover a database by combining current data files with older data files restored from backup, since Oracle selectively applies changes from the redo log files to all files that are not current. However, this strategy succeeds only if an *unbroken* sequence of redo log files is available to bring the older data files up to date.

Database failure and recovery modes

If the database has lost a control file, the database is recovered by replacing the lost control file with a copy of a current control file.

If the database has lost one or more data files, the database is recovered by first replacing the lost data file or data files with backup copies, then using the redo logs (online or archived) to make the restored copies current. If the backup copies are restored to different locations from the original files they are intended to replace, you must use the ALTER DATABASE RENAME FILE command to tell the Oracle instance where the restored files are to be found.

If the database has lost the current online redo log, the database instance will halt when it attempts to commit more transactions. No data will have been lost, except that the latest transaction will not be committed and may need to be reentered when the database comes back up. However, the current online redo log file will have to be replaced and a backup of the database should immediately be taken.

If the database has lost any archived redo logs, the database instance will continue to function, as it will have no knowledge of the loss. However, the ability to recover the database in the event of a second media failure or file loss may be compromised. A fresh backup should be taken if the archived redo logs are lost.

For detailed information on the recovery of an Oracle database, refer to the *Oracle Backup and Recovery Guide* for your Oracle release.

Note: ESRI *strongly* recommends that you test your backup procedure by building a functioning database instance from backup files *before* an emergency arises. If you have just loaded your database, you should make a full backup, then recover the database from the backed up files to ensure the recovery process will work when you need it.

APPENDIX A

Estimating the size of your tables and indexes

The formulas provided in this appendix provide approximations of the actual sizes of the Oracle tables and indexes created by ArcSDE.

The business table

The following calculation of business table size is based on the calculations provided in *Oracle Administrator's Guide* for your Oracle release.

1. Determine the data block size. Do this by connecting to the database as the DBA and querying the db_block_size parameter stored in the v$parameter file.

```
$ sqlplus system/<password>
```

```
SQL> select value from v$parameter where name = 'db_block_size';
```

2. Remove the header from the data block space.

```
space after headers(sph) = db_block_size - 86 - ((INITRANS -1) * 24)
```

3. Calculate the available space by removing the percent free from the space after headers.

```
available space = ceil(sph * (1 - PCTFREE / 100)) - 4
```

4. Reduce the available space further by 5 percent.

```
available space = (available space * 0.95)
```

5. Calculate the row size. You may use the formulas described in Step 3 of Appendix A in the *Oracle Administrator's Guide* to calculate the row size. However, if you can create the table and load approximately 100 sample records into it, you can obtain the most accurate estimate of row size by analyzing the table, then querying user_tables for the table's avg_row_len.

```
$ sqlplus <username>/<password>
```

```
SQL> create table <table_name> . . .
SQL> insert into table <table_name> . . .
SQL> analyze table <table_name> compute statistics;
SQL> select avg_row_len from user_tables where table_name = <table_name>;
```

6. Compute the number of rows that can be stored by each data block.

```
rows per data block = floor(available space / row size)
```

7. Compute the number of data blocks required to store the table. You will need to estimate the number of rows you expect the table to contain.

```
total data blocks = total rows / rows per data block
```

8. Compute tablespace required to store the table in bytes.

```
tablespace size = data block size * total data blocks
```

The feature table

Use the steps listed in the business table size calculation to calculate the size of the feature table. If you cannot load a sample feature class, use the following formula to calculate the row size:

row size = feature header + (average points per geometry) * 0.586 * coordinate factor

The feature header varies depending on the type of geometry the feature class stored and whether z-values and/or measures are stored. The following table lists the feature header sizes for the various combinations.

	x,y	**x,y,z**	**x,y,m**	**x,y,z,m**
points	85	103	121	139
lines	93	111	129	147
polygons	101	119	137	155

Feature headers vary according to the geometry type of the feature class and whether z-values, measures, or both are stored along with each x,y coordinate pair. The geometry type is listed by row, while the coordinate type is listed by column.

The coordinate factor varies depending on whether it will store z-values, measures, or both. The following table lists the possible coordinate factors.

Coordinate type	**Coordinate factor**
x,y	8
x,y,z	12
x,y,m	12
x,y,z,m	16

Coordinate factors vary depending on the type of coordinates stored.

If the features store annotation, the row size will be larger. Calculating the space required to store annotation can be rather complex with the annotation text, leader line, placement, and metadata. Most annotation will occupy between 100 and 300 bytes of space per feature. Therefore, increase the row size by 200 bytes if your features include annotation.

The spatial index table

You can use the steps listed in calculating the business table to calculate the size of the spatial index table except for the row size. The row size for the spatial index table is always 43 bytes.

However, the size of the spatial index table can be fairly accurately estimated given the number of records in a feature class’s business table.

spatial index table size = number of business table records * 1.3

This formula assumes that the spatial index table contains an average of 1.3 grid cells for each feature.

Use steps 6, 7, and 8, listed in the business table, to calculate the amount of space required to store the spatial index table.

The version delta tables

When you register a business table as multiversioned, three tables are created to maintain the deltas: the adds table, the deletes table, and the equivalency table. Depending on how the versions are managed, these tables may remain relatively small, or they may become quite large. The adds table receives one record for each record that is added to a business table version. The deletes table receives a record each time a record is deleted from a business table version.

The most difficult part of estimating space for the delta tables is trying to predict how many rows they will contain. It will depend partly on how many versions the table is involved in. In other words, the number of records added or deleted throughout the life of a version will affect the size of the adds and deletes tables.

The size of the delta tables also depends on how often records are removed. These tables shrink only when the states preceding the level 0 version are compressed. This occurs only after a version branching directly off the root of the version tree completes and is removed from the system. The compression of states that follows will cause the changes of the states between the level 0 version and the next version following the one removed to be written to the business table and deleted from the delta tables.

For most applications the delta tables should not be expected to grow beyond 10,000 records. However, there are applications where several versions of the database must be maintained over a long period. In these cases, as many as 30,000 records per delta table can be expected.

The adds table

The table definition of the adds table is identical to that of the business table with the addition of the SDE_STATE_ID column. Therefore, to compute the space required to store the adds table, use the calculations for the business table and increase the row size of the adds table by four bytes to account for the addition of the SDE_STATES_ID column.

The record count for the adds table will be an estimate of the number of records added to the versioned business table after it has been registered as multiversioned. Records are removed from the adds table only when a version next to the 0 version is removed and its states are compressed.

The deletes table

The record count depends on the number of records deleted from the business table after it was registered as multiversioned. Records are removed from the deletes table only when a version next to the 0 version is removed and its states are compressed.

Calculate the size of the deletes table with the following formula:

deletes table size = 5.4 * number of rows

For more information on versioning, refer to *Building a Geodatabase*.

The network tables

There are 10 standard network tables created whenever you create a geodatabase network class. Optionally, the application designer may also create weights for the network, in which case two tables are created for each weight.

The size of the network tables depends on the number of rows in the base tables. A network class is basically the logical combination of point and linestring feature classes. In network terms, linestring features are referred to as edges.

The first step in determining the size of the network tables is to establish the total number of edges and junctions.

The total number of edges is calculated by adding up all the linestring features in the network.

A junction must exist at the endpoint of each linestring. If a point feature does not exist at the endpoint of a linestring, a logical junction is created within the network tables. For endpoints of two or more linestrings that share the same coordinate position, only one junction exists. So some of the endpoints will share the same coordinate position, and some will not. Some will have a point feature at the endpoint, and some will not. As a rough guide assume that half of your linestring endpoints share the same coordinate position and that the total number of junctions required by the network is equal to the number of edges.

Once you have the junction and edge totals, you can determine the size of the network tables using the following formulas.

Description table (N_<n>_DESC)

The number of rows in the description table is equal to the sum of the junctions and edges.

The size of the description table is calculated as

size in bytes = number of rows * 7.4

Edge description table (N_<n>_EDESC)

The number of rows in the edge description table is calculated as

number of rows = number of edges /1,638

The size of the edge description table is calculated as

size in bytes = number of rows * 6,250

Edge status table (N_<n>_ESTATUS)

The number of rows in the edge status table is calculated as the

number of rows = number of edges/65,536

size in bytes = number of rows * 10,000

Edge topology table (N_<n>_ETOPO)

The number of rows in the edge topology table is calculated as the

number of rows = number of edges/4,096

The size of the edge topology table is calculated as

size in bytes = number of rows * 6,336

Flow direction table (N_<n>_FLODIR)

The number of rows in the flow direction table is calculated as

number of rows = number of edges/65,536

The size of the flow direction table is calculated as

size in bytes = number of rows * 40,960

Junction description table (N_<n>_JDESC)

The number of rows in the junction description table is calculated as

number of rows = number of junctions/1,638

The size of the junction description table is calculated as

size in bytes = number of rows * 24,986

Junction status table (N_<n>_JSTATUS)

The number of rows in the junction status table is calculated as

number of rows = number of junctions/65,536

The size of the junction status table is calculated as

size in bytes = number of rows * 40,960

Junction topology table (N_<n>_JTOPO)

The number of rows in the junction topology table is calculated as

number of rows = number of junctions/2,048

The size of the junction topology table is calculated as

size in bytes = number of rows * 24,865

Junction overflow topology table (N_<n>_JTOPO2)

The number of rows in the junction overflow topology table is calculated as

number of rows = number of junctions/20,480

The size of the junction overflow topology table is calculated as size in bytes = number of rows * 29,257

Property table (N_<n>_PROP)

The number of rows in the network properties table is always 10.

number of rows = 10

The size of the network properties table is approximately 250 bytes.

size in bytes ≈ 250

The network properties table will always occupy one data block.

Edge weight table (N_<n>_E<w>)

The number of rows in an edge weight table is calculated as

number of rows = number of edges/2,048

The size of the junction overflow topology table is calculated as

size in bytes = number of rows * 24,865

Junction weight table (N_<n>_J<w>)

The number of rows in the junction overflow topology table is calculated as

number of rows = number of junctions/2,048

The size of the junction overflow topology table is calculated as

size in bytes = number of rows * 24,865

The raster data tables

Four raster data tables are created whenever a user adds a raster column to a business table. A row is added to the sde raster_columns system table for each raster column created in the database. Each new raster column is given a unique ID called a rastercolumn_id. The rastercolumn_id is used in the naming of the raster data tables.

Raster data schema consists of four tables and their associated indexes. The four tables and their associated indexes are:

1. The raster table (SDE_RAS_<raster_column_id>) stores one record for each raster. The number of raster table rows is always less than or equal to the number of business table rows. If each row of the business table has an image stored in the raster tables, then the number of rows is equal. If some of the business table rows do not have an image associated with them, then the number for raster tables is less.

   The raster table has a single unique index on the raster_id column (SDE_RAS_<raster_column_id>_UK).

2. The raster bands table (SDE_BND_<raster_column_id>) stores one record for each raster band. The one-to-many relationship between the raster table and the raster bands table is maintained by the primary/foreign key raster_id column. The number of rows in the raster bands table can be determined by multiplying the number of image bands by the number of raster table rows.

   The raster bands table has two unique indexes, one on the rasterband_id column (SDE_BND_<raster_column_id>_UK1) and the other a composite of the raster_id and sequence_nbr columns (SDE_BND_<raster_column_id>_UK2).

3. The data stored in the raster auxiliary table (SDE_AUX_<raster_column_id>) is optional. Therefore, a zero-to-many relationship exists between the raster bands table and the raster auxiliary table. The tables are joined on the primary/foreign key relationship of the rasterband_id column. The raster auxiliary table stores one record of metadata about each band. For example, if the image has a color map associated with it, one record will be added to the auxiliary table for each record in the raster band table. If you elect to create statistics for the raster, one record will be added for each record in the raster bands table. If you are not sure how much metadata will be stored for each raster, you don’t need to be overly concerned since the size of this table is small.

   The raster auxiliary table has one unique index, a composite of the type, and rasterband_id columns (SDE_AUX_<raster_column_id>_UK).

4. The raster blocks table (SDE_BLK_<raster_column_id>) stores the raster band pixels as blocks defined by the block’s dimensions. A one-to-many relationship exists between the raster bands table and the raster blocks table. The number of rows in the raster blocks table can be estimated by first determining the pixels required to complete a block, dividing this number into the total number of pixels within a band, and multiplying the result by the number of bands in the raster. If you have elected to store a resolution pyramid, multiply the result by 1.33 to account for the resolution pyramid blocks.

The raster blocks table has one unique index—a composite of the rasterband_id, rrd_factor, row_nbr, and col_nbr columns (SDE_BLK_<raster_column_id>_UK).

Raster table (SDE_RAS_<raster_column_id>)

Roughly 2,400 raster table rows fit in a 16 KB Oracle data block, given an Oracle data block that is 10 percent free with four initial transactions. To roughly estimate the number of data blocks required to store the rows of the raster table, divide the total number of rows you expect the table to have by 2,400.

Raster bands table (SDE_BND_<raster_column_id>)

Approximately 190 raster band table rows fit within a 16 KB Oracle data block, given an Oracle data block that is 10 percent free with four initial transactions. To roughly estimate the number of data blocks required to store the rows of the raster bands table, divide the total number of rows you expect the table to have by 190.

Raster auxiliary table (SDE_AUX_<raster_column_id>)

Approximately six raster auxiliary table rows fit within a 16 KB Oracle data block, given an Oracle data block that is 10 percent free with four initial transactions. To roughly estimate the number of data blocks required to store the rows of the raster auxiliary table, divide the total number of rows you expect the table to have by six.

Raster blocks table (SDE_BLK_<raster_column_id>)

The size of the raster blocks table varies depending on whether you have built a resolution pyramid and whether an image compression method has been applied. The addition of the pyramid will increase the number of rows in the raster blocks table from the base level by a factor of one-third.

The pixel depth affects the row size of the raster block. Single-bit pixels require less storage space than do 4-bit pixels, which in turn require less space than 8-bit pixels, and so on.

The pixel dimension of the raster blocks also affects the row size of the raster blocks. Larger dimensions require more space. For example, the storage space of a 64 x 64 pixel raster block requires a quarter of the space of a 128 x 128 raster block.

Compression reduces the total size of the stored image in a manner similar to the algorithmic compression of ASCII files. The software interrogates rows of pixels searching for equal valued groups and storing a single value and the pixel count, rather than a string of equal values. Compression reduces the row size of the raster blocks.

Compression results vary from image to image; however, the table below provides a general guideline for the number of Oracle 16 KB data blocks required to store raster block table rows, given the image's pixel depth. To compute the total number of 16 KB Oracle data blocks required, multiply the expected number of rows in the raster block's table by the appropriate Oracle data blocks per row factor in the following table. If you created the database with an Oracle data block size of 8, multiply the result by 2.

	Pixel Depth			
Compression	**1-bit**	**8-bit**	**16-bit**	**32-bit**
None	0.17	1.5	2.6	4.5
LV77	0.01	0.55	0.78	1.08

Number of 16 KB Oracle data blocks required to store a row of the raster blocks table for a given pixel depth

The indexes

All of the indexes, with the exception of the spatial index table's S<n>_IX1, index single-integer columns. The spatial index table's S<n>_IX1 indexes all of the columns of the table. Since the definition of the indexes is fixed, the size of the indexes can be based on the number of rows they index. The space for all indexes, except the spatial index table's S<n>_IX index, can be calculated using the following formula:

adjusted size = 500 rows per data block * (1 - PCTFREE)

total data blocks = table rows/adjusted size

tablespace size = total data blocks * data block size

For the S<n>_IX1 index, the formula is

adjusted size = 150 rows per data block * (1 - PCTFREE)

total data blocks = table rows/adjusted size

tablespace size = total data blocks * data block size

APPENDIX B

Storing raster data

A raster is a rectangular array of equally spaced cells that, taken as a whole, represent thematic, spectral, or picture data. Raster data can represent everything from qualities of land surface, such as elevation or vegetation, to satellite images, scanned maps, and photographs.

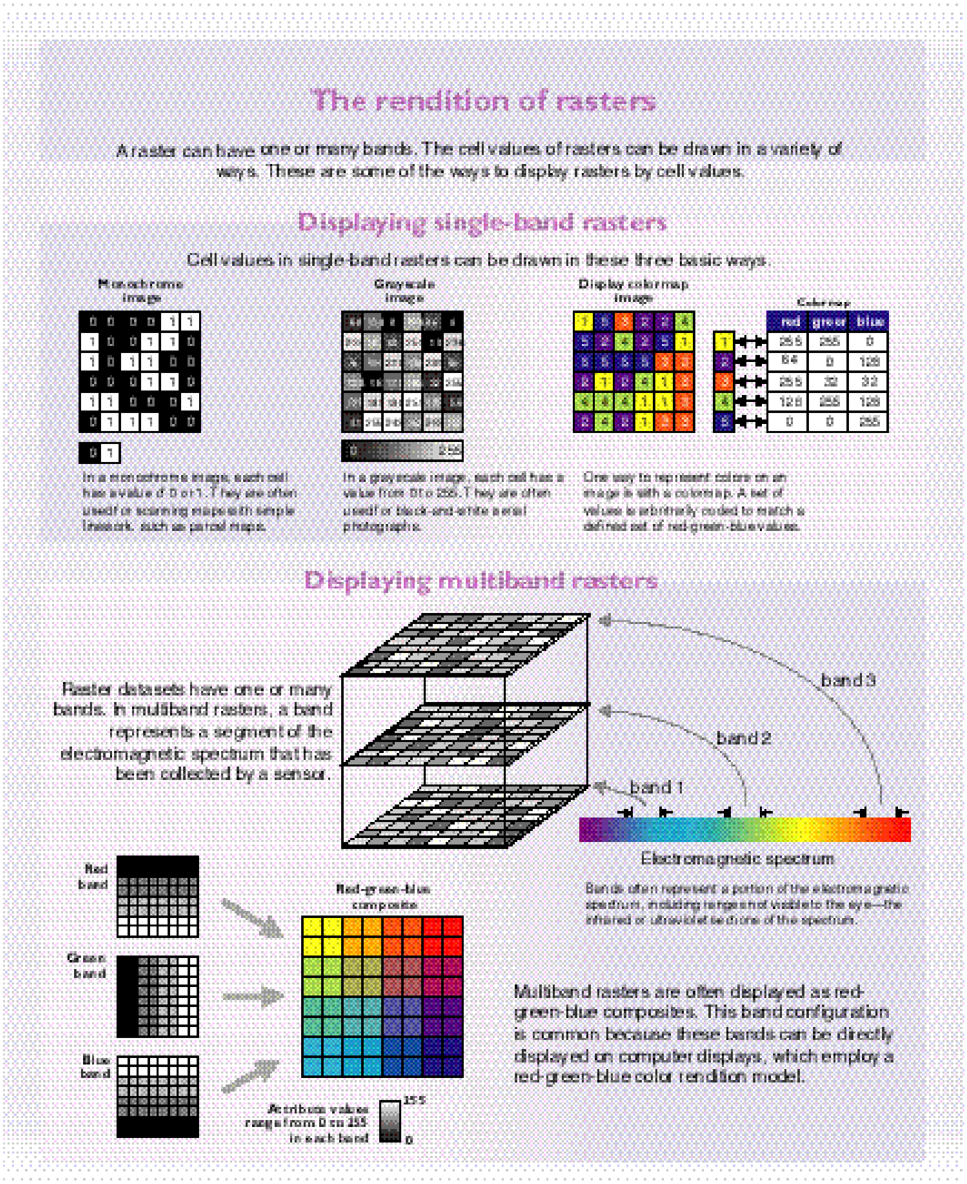
You are probably familiar with raster formats such as tagged image file format (TIFF), Joint Photographic Experts Group (JPEG), and Graphics Interchange Format (GIF) that your Internet browser renders. These raster images are composed of one or more bands. Each band is segmented into a grid of square pixels. Each pixel is assigned a value that reflects the information it represents at a particular position.

For an expanded discussion of the type of raster data supported by ESRI products, review Chapter 9, 'Cell-based modeling with rasters', in *Modeling Our World*.

ArcSDE stores raster datasets similar to the way it stores compressed binary feature classes (see Appendix C, 'ArcSDE compressed binary'). A raster column is added to a business table, and each cell of the raster column contains a reference to a raster stored in a separate raster table. Therefore, each row of a business table references an entire raster.

ArcSDE stores the raster bands in the raster bands table. ArcSDE joins the raster band table to the raster table on the raster_id column. The raster band table's raster_id column is a foreign key reference to the raster table's raster_id primary key.

ArcSDE automatically stores any existing image metadata, such as image statistics, color maps, and coordinate transformations in the raster auxiliary table. The rasterband_id column of the raster auxiliary table is a foreign key reference to the primary key of the raster band table. ArcSDE joins the two tables on this primary/foreign key reference when accessing a raster band's metadata.


The rendition of rasters

A raster can have one or many bands. The cell values of rasters can be drawn in a variety of ways. These are some of the ways to display rasters by cell values.

Displaying single-band rasters

Cell values in single-band rasters can be drawn in these three basic ways.

In a monochrome image, each cell has a value of 0 or 1. They are often used for scanning maps with simple linework, such as parcel maps.

In a grayscale image, each cell has a value from 0 to 255. They are often used for black-and-white aerial photographs.

One way to represent colors on an image is with a colormap. A set of values is arbitrarily coded to match a defined set of red-green-blue values.

Displaying multiband rasters

Raster datasets have one or many bands. In multiband rasters, a band represents a segment of the electromagnetic spectrum that has been collected by a sensor.

Bands often represent a portion of the electromagnetic spectrum, including ranges not visible to the eye—the infrared or ultraviolet sections of the spectrum.

Multiband rasters are often displayed as red-green-blue composites. This band configuration is common because these bands can be directly displayed on computer displays, which employ a red-green-blue color rendition model.

The raster blocks table stores the pixels of each raster band. ArcSDE tiles the pixels into blocks according to a user-defined dimension. ArcSDE does not have a default dimension; however, applications that store raster data in ArcSDE do. ArcCatalog, for example, uses a default raster block dimensions of 128 x 128 pixels per block. The dimensions of the raster block, along with any specified compression method, determine the storage size of each raster block. You should choose raster block dimensions that, combined with the compression method, allow each row of the raster block table to fit within an Oracle data block. For Oracle databases, storing raster data should be created with a 16 KB Oracle data block size. See Appendix A, 'Estimating the size of your tables and indexes', for more information on estimating the size of your raster tables and indexes.

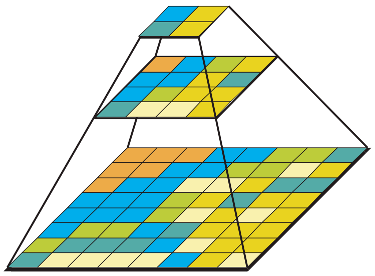
Using a compression method, such as lossless lz77, almost always results in improved performance. The savings in disk space and network I/O offset the additional CPU cycles required for the application to decompress the image.

The raster blocks table contains the rasterband_id column, which is a foreign key reference to the raster band table's rasterband_id primary key. ArcSDE joins these tables together on the primary/foreign key reference when accessing the blocks of the raster band.

ArcSDE populates the raster blocks table according to a declining resolution pyramid. The number of levels specified by the application determines the height of the pyramid. The application, such as ArcCatalog, may allow you to define the levels, request that ArcSDE calculate them, or offer both possible choices.

The pyramid begins at the base, or level 0, which contains the original pixels of the image. The pyramid proceeds toward the apex by coalescing four pixels from the previous level into a single pixel at the current level. This process continues until less than four pixels remain or until ArcSDE exhausts the defined number of levels.

The apex of the pyramid is reached when the uppermost level has less than four pixels. The additional levels of the pyramid increase the number of raster block table rows by one-third. However, since it is possible for the user to specify the number of levels, the true apex of the pyramid may not be obtained, limiting the number of records added to the raster blocks table.



When you build a pyramid, more rasters are created by progressively downsampling the previous level by a factor of two until the apex is reached. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The purpose of the pyramid is to optimize display performance.

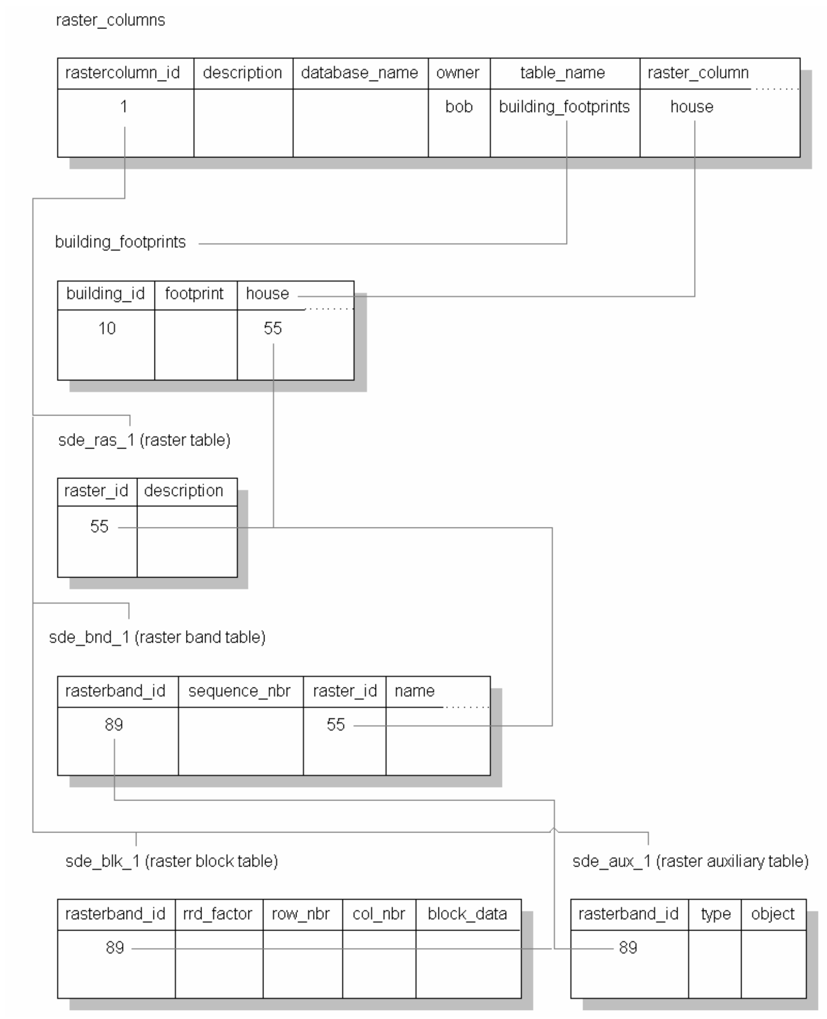
The pyramid allows ArcSDE to provide the application with a constant resolution of pixel data regardless of the rendering window's scale. Data of a large raster transfers more quickly to the client when a pyramid exists since ArcSDE can transfer fewer cells of a reduced resolution.

Raster schema

When you import a raster into an ArcSDE database, ArcSDE adds a raster column to the business table of your choice. You may name the raster column whatever you like, so long as it conforms to Oracle's column naming convention. ArcSDE restricts one raster column per business table.

The raster column is a foreign key reference to the raster_id column of the raster table created during the addition of the raster column. Also joined to the raster table's raster_id primary key, the raster band table stores the bands of the image. The raster auxiliary table, joined one to one to the raster band table by rasterband_id, stores the metadata of each raster band. The rasterbands_id also joins the raster band table to the raster blocks table in a many-to-one relationship. The raster blocks table rows store blocks of pixels, determined by the dimensions of the block.

The sections that follow describe the schema of the tables associated with the storage of raster data. Refer to the following figure for an illustration of these tables and the manner in which they are associated with one another.


raster_columns
rastercolumn_id | description | database_name | owner | table_name | raster_column
1 | bob | building_footprints | house
building_footprints
building_id | footprint | house
10 | 55
sde_ras_1 (raster table)
raster_id | description
55
sde_bnd_1 (raster band table)
rasterband_id | sequence_nbr | raster_id | name
89 | 55
sde_blk_1 (raster block table)
rasterband_id | rrd_factor | row_nbr | col_nbr | block_data
89
sde_aux_1 (raster auxiliary table)
rasterband_id | type | object
89

When ArcSDE adds a raster column to a table, it records that column in the SDE user's raster_columns table. The rastercolumn_id table is used in the creation of the table names of the raster, raster band, raster auxiliary, and raster blocks table.

RASTER_COLUMNS table

When you add a raster column to a business table, ArcSDE adds a record to the RASTER_COLUMNS system table maintained in the SDE user's schema. ArcSDE also creates four tables to store the raster images and metadata associated with each one.

NAME	DATA TYPE	NULL?
rastercolumn_id	NUMBER(38)	NOT NULL
description	VARCHAR2(65)	NULL
database_name	VARCHAR2(32)	NULL
owner	VARCHAR2(32)	NOT NULL
table_name	VARCHAR2(160)	NOT NULL
raster_column	VARCHAR2(32)	NOT NULL
cdate	NUMBER(38)	NOT NULL
config_keyword	VARCHAR2(32)	NULL
minimum_id	NUMBER(38)	NULL
base_rastercolumn_id	NUMBER(38)	NOT NULL
rastercolumn_mask	NUMBER(38)	NOT NULL
srid	NUMBER(38)	NULL

Raster columns table

- rastercolumn_id (SE_INTEGER_TYPE)—The table's primary key.
- description (SE_STRING_TYPE)—The description of the raster table.
- database_name (SE_STRING_TYPE)—Field is always NULL for Oracle.
- owner (SE_STRING_TYPE)—The owner of the raster column's business table.
- table_name (SE_STRING_TYPE)—The business table name.
- raster_column (SE_STRING_TYPE)—The raster column name.
- cdate (SE_INTEGER_TYPE)—The date the raster column was added to the business table.
- config_keyword (SE_STRING_TYPE)—The DBTUNE configuration keyword whose storage parameters determine how the tables and indexes of the raster are stored in the Oracle database. For more information on DBTUNE configuration keywords and their storage parameters, review Chapter 3, 'Configuring DBTUNE storage parameters'.
- minimum_id (SE_INTEGER_TYPE)—Defined during the creation of the raster, establishes value of the raster table's raster_id column.
- base_rastercolumn_id (SE_INTEGER_TYPE)—If a view of the business table is created that includes the raster column, an entry is added to the RASTER_COLUMNS table. The raster column entry of the view will have its own rastercolumn_id. The base_rastercolumn_id will be the rastercolumn_id of the business table used to create the view. This base_rastercolumn_id maintains

referential integrity to the business table. It ensures that actions performed on the business table raster column are reflected in the view. For example, if the business table's raster column is dropped, it will also be dropped from the view (essentially removing the view's raster column entry from the RASTER_COLUMNS table).

- rastercolumn_mask (SE_INTEGER_TYPE)—Currently not used; maintained for future use.
- srid (SE_INTEGER_TYPE)—The spatial reference ID is a foreign key reference to the SPATIAL_REFERENCES table. For images that can be georeferenced, the SRID references the coordinate reference system the image was created under.

Business table

In the example that follows, the fictitious BUILDING_FOOTPRINTS business table contains the raster column house_image. This is a foreign key reference to the raster table created in the user's schema. In this case the raster table contains a record for each raster of a house. It should be noted that images of houses cannot be georeferenced. Therefore, the SRID column of the RASTER_COLUMN record for this raster is NULL.

NAME	DATA TYPE	NULL?
building_id	NUMBER(38)	NOT NULL
building_footprint	NUMBER(38)	NOT NULL
house_picture	NUMBER(38)	NOT NULL

BUILDING_FOOTPRINTS business table with house image raster column

- building_id (SE_INTEGER_TYPE)—The table's primary key
- building_footprints (SE_INTEGER_TYPE)—A spatial column and foreign key reference to a feature table containing the building footprints
- house_image (SE_INTEGER_TYPE)—A raster column and foreign key reference to a raster table containing the images of the houses located on each building footprint

Raster table (SDE_RAS_<rastercolumn_id>)

The raster table, created as SDE_RAS_<raster_column_id> in the Oracle database, stores a record for each image stored in a raster column. The raster_column_id is assigned by ArcSDE whenever a raster column is created in the database. A record for each raster column in the database is stored in the ArcSDE RASTER_COLUMNS system table maintained in the SDE user's schema.

NAME	DATA TYPE	NULL?
raster_id	NUMBER(38)	NOT NULL
raster_flags	NUMBER(38)	NULL
description	VARCHAR2(65)	NULL

Raster description table schema (SDE_RAS_<raster_column_id>)

- raster_id (SE_INTEGER_TYPE)—The primary key of the raster table and unique sequential identifier of each image stored in the raster table
- raster_flags (SE_INTEGER_TYPE)—A bit map set according to the characteristics of a stored image
- description (SE_STRING_TYPE)—A text description of the image (not implemented at ArcSDE 8.1)

Raster band table (SDE_BND_<rastercolumn_id>)

Each image referenced in a raster may be subdivided into one or more raster bands. The raster band table, created as SDE_BND_<rastercolumn_id>, stores the raster bands of each image stored in the raster table. The raster_id column of the raster band table is a foreign key reference to the raster table's raster_id primary key. The rasterband_id column is the raster band table's primary key. Each raster band in the table is uniquely identified by the sequential rasterband_id.

NAME	DATA TYPE	NULL?
rasterband_id	NUMBER(38)	NOT NULL
sequence_nbr	NUMBER(38)	NOT NULL
raster_id	NUMBER(38)	NOT NULL
name	VARCHAR2(65)	NULL
band_flags	NUMBER(38)	NOT NULL
band_width	NUMBER(38)	NOT NULL
band_height	NUMBER(38)	NOT NULL
band_types	NUMBER(38)	NOT NULL
block_width	NUMBER(38)	NOT NULL
block_height	NUMBER(38)	NOT NULL
block_origin_x	NUMBER(64)	NOT NULL
block_origin_y	NUMBER(64)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	NOT NULL

```
cdate            NUMBER(38)           NOT NULL
mdate            NUMBER(38)           NOT NULL
```

Raster band table schema

- rasterband_id (SE_INTEGER_TYPE)—The primary key of the raster band table that uniquely identifies each raster band.
- sequence_nbr (SE_INTEGER_TYPE)—An optional sequential number that can be combined with the raster_id as a composite key as a second way to uniquely identify the raster band.
- raster_id (SE_INTEGER_TYPE)—The foreign key reference to the raster table's primary key. Uniquely identifies the raster band when combined with the sequence_nbr as a composite key.
- name (SE_STRING_TYPE)—The name of the raster band.
- band_flags (SE_INTEGER_TYPE)—A bit map set according to the characteristics of the raster band.
- band_width (SE_INTEGER_TYPE)—The pixel width of the band.
- band_height (SE_INTEGER_TYPE)—The pixel height of the band.
- band_types (SE_INTEGER_TYPE)—A bitmap band compression data.
- block_width (SE_INTEGER_TYPE)—The pixel width of the band's tiles.
- block_height (SE_INTEGER_TYPE)—The pixel height of the band's tiles.
- block_origin_x (SE_FLOAT_TYPE)—The leftmost pixel.
- block_origin_y (SE_FLOAT_TYPE)—The bottommost pixel.

If the image has a map extent, the optional eminx, eminy, emaxx, and emaxy will hold the coordinates of the extent.

- eminx (SE_FLOAT_TYPE)—The band's minimum x coordinate
- eminy (SE_FLOAT_TYPE)—The band's minimum y coordinate
- emaxx (SE_FLOAT_TYPE)—The band's maximum x coordinate
- emaxy (SE_FLOAT_TYPE)—The band's maximum y coordinate

- cdate (SE_FLOAT_TYPE)—The creation date
- mdate (SE_FLOAT_TYPE)—The last modification date

Raster blocks table (SDE_BLK_<rastercolumn_id>)

Created as SDE_BLK_<rastercolumn_id>, the raster blocks table stores the actual pixel data of the raster images. ArcSDE evenly tiles the bands into blocks of pixels. Tiling the raster band data enables efficient storage and retrieval of the raster data. The raster blocks can be configured so that the records of the raster block table fit with an Oracle data block, avoiding the adverse effects of data block chaining.

The rasterband_id column of the raster block table is a foreign key reference to the raster band table's primary key. A composite unique key is formed by combining the rasterband_id, rrd_factor, row_nbr, and col_nbr columns.

NAME	DATA TYPE	NULL?
rasterband_id	NUMBER(38)	NOT NULL
rrd_factor	NUMBER(38)	NOT NULL
row_nbr	NUMBER(38)	NOT NULL
col_nbr	NUMBER(38)	NOT NULL
block_data	LONG RAW or BLOB	NOT NULL

Raster block table schema

- rasterband_id (SE_INTEGER_TYPE)—The foreign key reference to the raster band table's primary key.
- rrd_factor (SE_INTEGER_TYPE)—The reduced resolution dataset factor determines the position of the raster band block within the resolution pyramid. The resolution pyramid begins at 0 for the highest resolution and increases until the raster band's lowest resolution level has been reached.
- row_nbr (SE_INTEGER_TYPE)—The block's row number.
- col_nbr (SE_INTEGER_TYPE)—The block's column number.
- block_data (SE_BLOB_TYPE)—The block's tile of pixel data.

Raster band auxiliary table (SDE_AUX_<rastercolumn_id>)

The raster band auxiliary table, created as SDE_AUX_<rastercolumn_id>, stores optional raster metadata such as the image color map, image statistics, and coordinate transformations used for image overlay and mosaicking. The rasterband_id column is a foreign key reference to the primary key of the raster band table.

NAME	DATA TYPE	NULL?
rasterband_id	NUMBER(38)	NOT NULL
type	NUMBER(38)	NOT NULL
object	LONG RAW or BLOB	NOT NULL

Raster auxiliary table schema

- rasterband_id (SE_INTEGER_TYPE)—The foreign key reference to the raster band table's primary key
- type (SE_INTEGER_TYPE)—A bit map set according to the characteristics of the data stored in the object column
- object (SE_BLOB_TYPE)—May contain the image color map, image statistics, or coordinate transformation

APPENDIX C

ArcSDE compressed binary

ArcSDE uses a compressed binary format to store geometry in either an Oracle binary LONG RAW or BLOB data type. ArcSDE stores compressed binary spatial data in the POINTS columns of the feature table. Compressing the geometry offers efficient storage and retrieval of spatial data by reducing the size of the geometry. Compressed binary stored in the LONG RAW data type is the default storage format for ArcSDE feature classes.

Compressed binary

ArcSDE verifies the geometry, compresses it, and sends it to the Oracle instance, where it is inserted into a feature table in compressed binary format. Compressing the geometry on the client offloads the task from the ArcSDE server and reduces the transmission time to send the geometry to the ArcSDE server. Storing compressed geometry data reduces the space required to store data by as much as 40 percent.

Compressed binary schema

A compressed binary feature class comprises three tables: the business table, the feature table, and the spatial index table.

The business table contains attributes and a spatial column. The spatial column is a key to the feature and spatial index tables.

The relationship between the business table and the feature table is managed through the spatial column and the FID column. This key, which is maintained by ArcSDE, is unique.

NAME	DATA TYPE	NULL?
fid	NUMBER(38)	NOT NULL
numofpts	NUMBER(38)	NOT NULL
entity	NUMBER(38)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	NOT NULL
eminz	NUMBER(64)	NULL
emaxz	NUMBER(64)	NULL
min_measure	NUMBER(64)	NULL
max_measure	NUMBER(64)	NULL
area	NUMBER(64)	NOT NULL
len	NUMBER(64)	NOT NULL
points	LONG RAW or BLOB	NULL
anno_text	VARCHAR2(256)	NULL

Feature table schema

The feature table stores the geometry, annotation, and computer-aided design (CAD) elements in the POINTS column. The POINTS column may be defined as either LONG RAW or BLOB depending on the setting of the GEOMETRY_STORAGE DBTUNE storage parameter. Set the GEOMETRY_STORAGE DBTUNE storage parameter to SDEBINARY if you want to store the compressed binary spatial data in a column defined as LONG RAW; otherwise, set the storage parameter to SDELOB if you want to store the compressed binary spatial data in a column defined as BLOB.

For an expanded discussion of the GEOMETRY_STORAGE DBTUNE storage parameter, see Chapter 3, 'Configuring DBTUNE storage parameters'.

The ArcSDE datatype for each column is defined below.

- fid (SE_INTEGER_TYPE)—Contains the unique ID that joins the feature table to the business table
- entity (SE_INTEGER_TYPE)—The type of geometric feature stored in the spatial column (e.g., point, linestring)
- numofpts (SE_INTEGER_TYPE)—The number of points defining the geometry
- eminx, eminy, emaxx, emaxy (SE_FLOAT_TYPE)—The envelope of the geometry
- eminz (SE_FLOAT_TYPE)—The minimum z-value in the geometry

- emaxz (SE_FLOAT_TYPE)—The maximum z-value in the geometry
- min_measure (SE_FLOAT_TYPE)—The minimum measure value in the geometry
- max_measure (SE_FLOAT_TYPE)—The maximum measure value in the geometry
- area (SE_FLOAT_TYPE)—The area of the geometry
- len (SE_FLOAT_TYPE)—The length or perimeter of the geometry
- points (SE_SHAPE_TYPE)—Contains the byte stream of point coordinates that define the geometry
- anno_text (SE_STRING_TYPE)—Contains the feature annotation string

NAME	DATA TYPE	NULL?
sp_fid	NUMBER(38)	NOT NULL
gx	NUMBER(38)	NOT NULL
gy	NUMBER(38)	NOT NULL
eminx	NUMBER(64)	NOT NULL
eminy	NUMBER(64)	NOT NULL
emaxx	NUMBER(64)	NOT NULL
emaxy	NUMBER(64)	NOT NULL

Spatial index table schema

The spatial index table defines the grid range and extent of all geometry in an ArcSDE feature class.

- sp_fid—Contains the unique ID that joins the feature table to the business table
- gx/gy—Defines the feature's extent in grid cells
- eminx/eminy/emaxx/emaxy—Defines the extent of the feature in system units

In this example the FEATURE-ID column from the WELLS business table references features from the feature and spatial index tables:

WELL_ID	DEPTH	ACTIVE	FEATURE-ID
1	30029	Yes	101
2	13939	No	102
3	92891	No	103
…	…		…

FID	AREA	LEN	EMINX,EMINY,…	POINTS
101				<compressed feature>
102				<compressed feature>
103				<compressed feature>
...				...

SP_FID	GX	GY	{EMINX,EMINY,EMAXX,EMAXY}
101	70	100	
102	70	100	
103	71	100	
...			

A business/feature/spatial index key reference

The spatial grid index

The spatial grid index is a two-dimensional index that spans a feature class, like the reference grid you might find on a common road map. You may assign the spatial grid index one, two, or three grid levels, each with its own distinct cell size. The mandatory first grid level has the smallest cell size. The optional second and third grid cell levels are disabled by setting them to 0. If enabled, the second grid cell size must be at least three times larger than the first grid cell size and the third grid cell size must be three times larger than the second grid cell size.

The spatial index table (S<feature class_id>) has seven integer columns that store the grid cell values, feature envelopes, and corresponding feature IDs. Adding a feature to a feature class adds one or more grid cells to the spatial index table. The number of records added to the spatial index table depends on the number of grid cells the feature spans.

The spatial index table contains two indexes. One index is on the SP_FID column, which contains the feature ID. The other is a composite index that includes all of the columns of the spatial index table. Since all of the columns of the spatial index table are indexed, the values of the table are read from the leaf blocks of the index and not the table data blocks. The result is less I/O and better performance. In addition, the spatial index table is not accessed whenever the feature class is queried. Therefore, when considering how to position the tables and indexes to reduce disk I/O contention, you should be concerned about the positioning of the indexes of the spatial index table but not the table itself.

Building the spatial index

Every time a feature class is added to a business table, a persistent spatial index is built for it.

The ArcSDE server manages the spatial index throughout the life of the feature class. As features are inserted, updated, or deleted, the spatial index is automatically updated.

A load-only mode disables spatial index management until loading completes. This boosts loading performance substantially and is imperative for bulk loading efforts. No queries are allowed in the load-only mode except native SQL-based queries.

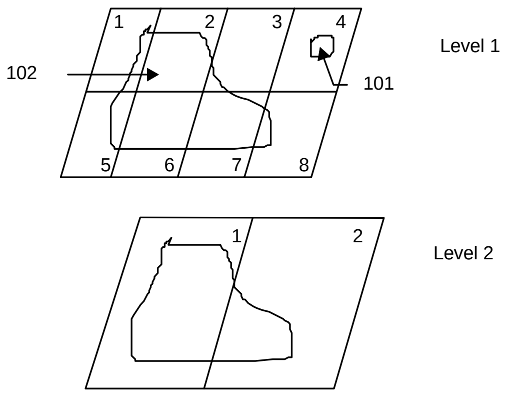
Once loading has been completed, the spatial index is enabled by returning it to normal I/O mode. The conversion from normal I/O mode to load-only I/O mode reconstructs the spatial index.

Inserting, updating, or deleting a feature updates the spatial index when the feature class is in normal I/O mode.

ArcSDE overlays the extent of each feature onto the lowest grid level to obtain the number of grid cells. If the feature exceeds four cells, ArcSDE promotes the feature to the next highest grid level, if you have defined one. ArcSDE will continue to promote the feature until it fits within four cells or less or until the highest defined grid level is reached. On the highest defined grid level, geometries can be indexed by more than four grid cells.

ArcSDE adds the feature’s grid cells to the spatial index table with their corresponding shape ID and feature envelope. The grid level is encoded with each grid cell.

In the example below, the feature class has two grid levels. Area shape 101 is located in grid cell 4 on level 1. A record is added to the spatial index table because the feature resides within four grid cells (in this case it is one). The envelope for area feature 102 is located in cells 1 through 8 on level 1. Because the feature’s envelope resides in more than four grid cells, the feature is promoted to level 2, where its envelope fits within two grid cells. Feature 102 is indexed at level 2, and two records are added to the spatial index table.



Shape 101 is indexed on grid level 1, while shape 102 is indexed on grid level 2, where it is in only two grid cells.

Spatial queries and the spatial index

Spatial queries, such as finding all the lakes within a state boundary, use the spatial index. The spatial index is used unless the search order has been set to SE_ATTRIBUTE_FIRST in the SE_stream_set_spatial_constraints function. When the search order is set to SE_ATTRIBUTE_FIRST, ArcSDE ignores the spatial index and the criteria of the attribute's where clause determines which records of the feature class the query returns.

Whenever the spatial index is used, the ArcSDE service generally uses the following decision process to perform the query:

1. Define the envelope. The envelope could be defined directly by the application such as the extent defined by the ArcMap Zoom In tool. Alternatively, the envelope may be defined as the envelope of another feature.
2. Join the spatial index table with the feature table and return all features whose grid cells intersect the envelope.
3. Join the feature table with the business table and apply the criteria of the attribute's where clause to further refine the features returned.

Grid cell size impacts the size of the spatial index table. Setting up the spatial index means balancing the cell sizes—smaller cell sizes mean more cells per shape, which requires more entries in the spatial index table.

Guidelines for tuning the spatial index

Because client applications and spatial data profiles vary from one system to another, no single solution fits all. Experienced users of ArcSDE often experiment with the spatial index, trying different cell sizes and different grid-level configurations.

The sdelayer command has several operations that can help you optimize the spatial index by changing the grid cell sizes and adding new grid levels with the “alter” operation. The “stats” and “si_stats” operations profile your spatial data and current spatial index.

The following guidelines can help improve the performance of spatial queries.

- Consider how many grid levels are needed and remember the ArcSDE server scans the spatial index table once per grid level. Often a single grid level is the best solution for a feature class, despite the notion of distributing geometries evenly across many grid levels to minimize the spatial index entries.

- Use one grid level for pure point type feature class and consider making the cell sizes large. Spatial queries generally process point geometries faster than other geometry types.

- Monitor the spatial index. Tuning a spatial index is difficult if the data changes frequently. Tuning depends on the structure of the spatial data. Periodically assess the spatial index as your spatial data changes.

- Base the spatial index on the application. Match the spatial index grid cell sizes to the extent of the application window. By doing so, the application is probably viewing exact entries in the spatial index table. This helps to size the spatial index table suitably and reduces the amount of processing because fewer candidate feature IDs must be evaluated against the feature table (see ‘Spatial queries and the spatial index’ above).

- For unknown or variable application windows, start by defining one grid level with a cell size three times the average feature extent size. Query the feature table to obtain the average feature size with the following SQL statement:

  ```
  select (avg(emaxx - eminx) + avg(emaxy - eminy)) / 2
  from f<N>;
  ```

  ```
  (where <N> is the layer number of the feature class)
  ```

  Such spatial index configuration minimizes the number of rows in the spatial index table while maintaining the proficiency of the index because the majority of the features can be referenced by less than one or two grid cells.

- Design the feature class around spatial data categories such as type, geometry size, and distribution. Sometimes a carefully designed feature class, using these categories, can substantially boost the performance of spatial queries.

Displaying spatial index statistics

The sdelayer command's spatial index statistics operation, 'si_stats', can help you determine optimum spatial index grid sizes. Optimum grid cell sizes depend on the spatial extent of all feature geometries, the variation in feature geometry spatial extent, and the types of searches to be performed on the map feature class. Below is a sample output generated by si_stats:

```
$ sdelayer -o si_stats -l victoria,parcels -u av -p mo -i sde81
ArcSDE   8.1          Wed Jan 17 22:43:09 PST 2000
Layer    Administration Utility
--------------------------------------------------------
Layer 1 Spatial Index Statistics:
Level 1,   Grid Size 200
|-------------------------------------------------------|
| Grid Records: 978341                                  |
| Feature Records: 627392                               |
| Grids/Feature Ratio: 1.56                             |
| Avg. Features per Grid: 18.26                         |
| Max. Features per Grid: 166                           |
| % of Features Wholly Inside 1 Grid: 59.71             |
|-------------------------------------------------------|
|             Spatial Index Record Count By Group       |
| Grids:     <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes:   627392   0    0    0    0     0     0     0 |
| % Total:     100%  0%   0%   0%   0%    0%    0%    0%|
|-------------------------------------------------------|
Level 2,   Grid Size 1600 (Meters)
|-------------------------------------------------------|
| Grid Records: 70532                                   |
| Feature Records: 36434                                |
| Grids/Feature Ratio: 1.94                             |
| Avg. Features per Grid: 18.21                         |
| Max. Features per Grid: 82                            |
| % of Features Wholly Inside 1 Grid: 45.35             |
|-------------------------------------------------------|
|             Spatial Index Record Count By Group       |
| Grids:     <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes:   35682  752   87   17    3     0     0     0 |
| % Total:     97%   2%   0%   0%   0%    0%    0%    0%|
|-------------------------------------------------------|
```

As the output shows, for each defined spatial index level, the following values and statistics are printed:

- Grid level and cell size.

- Total spatial index records for the current grid level.
- Total geometries stored for the current grid level.
- Ratio of spatial index records per geometry.
- Geometry counts and percentages by group that indicate how geometries are grouped within the spatial index at this grid level. The column headings have the following meaning (where “N” is the number of grid cells):

<=N Number of geometries and percentage of total geometries that fall within <= N grid cells
>N Number of geometries and percentage of total geometries that fall within > N grid cells

Notice that the “>” groupings include count values from the next group. For instance, the “>4” group count represents the number of geometries that require more than four grid records as well as more than 10, and so on.

- Average number of geometries per grid.
- Maximum number of geometries per grid. This is the maximum number of geometries indexed into a single grid.
- Percentage of geometries wholly inside one grid. This is the percentage of all geometries wholly contained by one grid record.

The output sample shows spatial index statistics for a map feature class that uses two grid levels: one that specifies a grid size of 200 meters, the other a grid size of 1,600 meters. When a geometry requires more than four spatial index records, it is automatically promoted to the next grid level, if one is defined. A geometry will not generate more than four records if a higher grid level is available.

In the example above, 627,392 features are indexed through grid level 1. Because the system automatically promotes geometries that need more than four spatial index records to the next defined grid level, all 627,392 geometries for grid level 1 are indexed with four grid records or less. Grid level 2 is the last defined grid level, so geometries indexed at this level are allowed to be indexed with more than four grid records. At grid level 2, there are a total of 36,434 geometries and 70,532 spatial index records. There are 35,682 geometries indexed with four grid records or fewer, 752 geometries indexed with more than four grid records, 87 geometries indexed with more than 10, 17 geometries with more than 25, and three geometries with more than 50. Percentage values below each column show how the geometries are dispersed through the eight groups.

Creating tables with compressed binary schema

The geometry storage format for ArcSDE feature classes defaults to LONG RAW compressed binary. If you always store your geometry in this format, you do not need to adjust the geometry storage format.

If you wish to have a mix of geometry types in your schema, add configuration keywords to the DBTUNE table with the desired geometry storage format and assign the configuration keywords to the feature classes.

The DBTUNE table storage parameter GEOMETRY_STORAGE defines the geometry storage format of a feature class. The GEOMETRY_STORAGE value for the default, LONG RAW compressed binary, is SDEBINARY. In the following example, a dbtune file has a PARCELS configuration keyword that contains the value SDEBINARY.

```
##PARCELS
GEOMETRY_STORAGE           SDEBINARY
<other parameters>
END
```

Additional storage parameters precisely define the storage configuration of the parcel's feature class.

```
##PARCELS

GEOMETRY_STORAGE    "SDEBINARY"
B_STORAGE           "TABLESPACE btabsp STORAGE (INITIAL 500M)"
B_INDEX_ROWID       "TABLESPACE bindex_tsp STORAGE (INITIAL 100M)"
B_INDEX_SHAPE       "TABLESPACE bindex_tsp STORAGE (INITIAL 100M)"
A_STORAGE           "TABLESPACE atabsp STORAGE (INITIAL 100M)"
A_INDEX_ROWID       "TABLESPACE aindex_tsp STORAGE (INITIAL 50M)"
A_INDEX_SHAPE       "TABLESPACE aindex_tsp STORAGE (INITIAL 50M)"
A_INDEX_STATEID     "TABLESPACE aindex_tsp STORAGE (INITIAL 50M)"
D_STORAGE           "TABLESPACE atabsp STORAGE (INITIAL 50M)"
D_INDEX_STATE_ROWID "TABLESPACE dindex_tsp STORAGE (INITIAL 50M)"
D_INDEX_DELETED_AT  "TABLESPACE dindex_tsp STORAGE (INITIAL 50M)"
F_STORAGE           "TABLESPACE ftabsp STORAGE (INITIAL 500M)"
F_INDEX_FID         "TABLESPACE findex_tsp STORAGE (INITIAL 50M)"
F_INDEX_AREA        "TABLESPACE findex_tsp STORAGE (INITIAL 50M)"
F_INDEX_LEN         "TABLESPACE findex_tsp STORAGE (INITIAL 50M)"
S_STORAGE           "TABLESPACE stabsp STORAGE (INITIAL 300M)"
S_INDEX_ALL         "TABLESPACE sindex_tsp STORAGE (INITIAL 100M)"
S_INDEX_SP_FID      "TABLESPACE sindex_tsp STORAGE (INITIAL 100M)"
END
```

In this example, the storage schema is compressed binary; it defines the storage and location for the business table, adds table, deletes table, feature table, and spatial index table. Chapter 3, 'Configuring DBTUNE storage parameters', describes these storage parameters as well as many others.

Tuning LOB storage

Large Object (LOB) data types are used by Oracle to store large, unstructured datasets. ArcSDE uses this data type for large datasets that are processed by specialized ArcSDE algorithms.

LOB data may be stored *in-line*, meaning that LOB and non-LOB data from the same row in a table are stored in the same Oracle data block. Alternately, LOB data may be stored *out-of-line* meaning that the LOB data is stored separately from the rest of the row data in a special area set aside for it by the database. Oracle stores LOB data in-line or out-of-line depending upon the size of the LOB and the storage parameters associated with creation of the table holding the LOB data.

If the total size of a LOB datum plus the storage locator is less than 4,000 bytes and the ENABLE STORAGE IN ROW clause is specified, the LOB datum is stored in the same block as the rest of the row fields. If possible to achieve, this configuration will result in less I/O to read a single LOB datum.

If the total size of a LOB datum plus the storage locator exceeds 4,000 bytes or the DISABLE STORAGE IN ROW clause is specified, Oracle stores the LOB datum out-of-line. This configuration will result in two I/Os to fetch a single LOB datum—the first to fetch row data plus the LOB locator and a second to fetch LOB itself.

To create a new row with a LOB datum, Oracle effectively performs an INSERT of the new row storing the row data and the LOB locator followed by an UPDATE of the same row to add the actual LOB data. If several rows having LOB data are inserted in a single SQL statement, *all* INSERTs are performed before any UPDATE.

This has important consequences on the storage parameters chosen for a table holding LOB data. If, for example, you insert nine rows with LOB data and Oracle inserts them into a single block, then performs the UPDATE, it may happen that none of the LOB data fits into the original block and that all must chain or migrate to another block. Setting a high value for PCTFREE will reduce this undesirable effect.

To calculate an appropriate value for PCTFREE, first estimate the average size of the LOB data and the average size of the non-LOB data. If datasize is the amount of space available in a block:

```
datasize = blocksize - blockheadersize
```

Then

```
num_rows = TRUNC ( datasize / (aveLOB + aveNonLOB) )
```

will be the maximum number of rows expected to fit completely into a block. Set PCTFREE to a value that will limit the amount of non-LOB data to be inserted, leaving space for LOB data, as follows:

```
PCTFREE = 1 – (num_rows * aveNonLOB / datasize )
```

The F_STORAGE configuration string within the DBTUNE table controls the allocation of space for feature tables. By default, PCTFREE is set to 30 in the F_STORAGE configuration strings. A larger value of PCTFREE is necessary to reduce the problem of row chaining that may occur when data is stored as a BLOB data type.

Referential integrity

Maintaining the referential integrity between the business and feature table is important. You should not edit the records of either the feature table or the spatial index table. Several indexes and constraints have been added to the business, feature, and spatial index table to ensure referential integrity is maintained. However, these indexes and constraints are removed when the feature class is converted to the load-only I/O mode, a state that allows for rapid insertion of data into the feature class.

When the feature class is placed back into normal I/O mode—the state that allows users to query the feature class through an ArcSDE client—the indexes are created and the constraints are enabled. The conversion to normal I/O mode will fail if the unique indexes cannot be built on the business table’s spatial column or the feature table’s FID column. It will also fail if a value exists in the business table’s spatial column that is not in the feature table’s FID column. In this case a reference to the offending business table record is loaded into the SDE_EXCEPTIONS table.

APPENDIX D

Oracle Spatial geometry type

ArcSDE supports Oracle Spatial's Object Relational Model as a method to store spatial data. Oracle Spatial's Object Relational Model was introduced in Oracle8*i* using object-relational types and methods to define, index, and perform spatial analysis on spatial data. For additional information on Oracle Spatial, see the *Oracle Spatial User's Guide and Reference* for your Oracle release.

What is Oracle Spatial?

Oracle Spatial extends the Oracle database with the addition of spatial data management functions. Oracle Spatial provides a SQL geometry type, spatial metadata schema, indexing methods, functions, and implementation rules.

Oracle Locator is a subset of Oracle Spatial, available starting with Oracle9*i*. Oracle Locator includes the SDO_GEOMETRY data type along with some of the functionality provided with Oracle Spatial. Oracle Locator may be used with ArcSDE. Refer to Oracle's documentation for an explanation of the difference between Oracle Spatial and Oracle Locator. Throughout this appendix, "Oracle Locator" can be used in place of references to "Oracle Spatial".

The *Oracle Spatial User's Guide and Reference* for your Oracle release is a valuable resource for more detailed information about Oracle Spatial and Oracle Locator. Refer to this document if you have questions about Oracle Spatial architecture and functionality.

SDO_GEOMETRY

The Oracle Spatial geometry type SDO_GEOMETRY is implemented using Oracle's extensible object-relational type system. The SDO_GEOMETRY type stores information about a geometry including its geometry type, spatial reference ID, interpolation type (straight versus curved), and coordinate values.

The SDO_GEOMETRY type supports single and multipart point, line, and area geometry. Geometries may be defined as having linear interpolation between coordinates as defined by the OpenGIS Simple Feature Specification. Geometries may also be constructed from circular curves or a combination of both interpolation methods.

Application programs are responsible for properly inserting, updating, and fetching the contents of the SDO_GEOMETRY type using Oracle's object-relational SQL interface. Applications are also responsible for ensuring that the content of each geometry adheres to the rules defined in the Oracle Spatial documentation. Oracle provides geometry validation routines that can be executed after inserting geometries.

Note: Oracle's geometry validation routines do not implement precisely the same set of rules as the ArcSDE geometry validation. ArcSDE is designed to store SDO_GEOMETRY that satisfies Oracle's validation rules.

Metadata schema

Information about every SDO_GEOMETRY column should be recorded in the Oracle Spatial metadata schema, though Oracle Spatial does not do this automatically. It is the responsibility of software that uses Oracle Spatial data (such as ArcSDE) to insert and update the metadata for each SDO_GEOMETRY column it creates or modifies. The metadata contains the spatial column name, the name of the table it resides in and its owner, the Oracle Spatial Reference Identifier (SRID), the number of dimensions, the range of each dimension, and its identical coordinate tolerance.

Spatial index

Spatial indexes provide fast access to records based on the location of geometry. Oracle Spatial provides three different spatial indexing methods: Fixed Quadtree, Hybrid Quadtree, and R-tree indexes. R-tree spatial indexes are generally the most efficient and easiest to create of the three. Oracle recommends using R-tree spatial indexes in most situations.

Oracle Spatial provides the Spatial Index Advisor utility to assist in determining the best type of spatial index for a given table. In addition, consult your *Oracle Spatial User's Guide and Reference* for detailed information on supported spatial index types, how to create them, and the trade-offs of different spatial index methods.

Spatial functions

Oracle Spatial extends SQL with spatial search functions for primary and secondary filtering. Using the SDO_FILTER function in a SQL query will perform a primary spatial search utilizing the spatial index. Spatial predicates, such as SDO_RELATE and SDO_CONTAINS, return secondary relationships between pairs of SDO_GEOMETRY objects.

Oracle Spatial has spatial transformation functions that change the form of an SDO_GEOMETRY value. For example, the SDO_BUFFER function computes the coordinates of a new SDO_GEOMETRY object as a buffer polygon at a given distance surrounding the original geometry. Other spatial transformation functions include SDO_DIFFERENCE and SDO_INTERSECTION.

Coordinate reference

Oracle Spatial provides access to many predefined coordinate reference systems using SRID values. The SRID value, stored in the SDO_GEOMETRY object, specifies the coordinate reference for the geometry stored in that object. The SRID in the SDO_GEOMETRY object, if not NULL, is a foreign key in a table containing details about each SRID. This table is called MDSYS.CS_SRS.

The SDO_TRANSFORM function uses the spatial reference ID to establish coordinate reference transformations. ArcSDE may also use this information to create ArcSDE spatial references.

How does ArcSDE use Oracle Spatial?

ArcSDE supports Oracle Spatial or Oracle Locator for storing and managing geometry in an Oracle database.

Making Oracle Spatial your default geometry schema

When you install ArcSDE, the SDE compressed binary geometry schema is set as the default storage type. The default settings for ArcSDE storage are defined in the DBTUNE table—one of these parameters is the GEOMETRY_STORAGE method.

Changing the default ArcSDE geometry storage to use Oracle Spatial is easy. For the DEFAULTS keyword, simply change the GEOMETRY_STORAGE parameter from SDEBINARY to SDO_GEOMETRY. After the default GEOMETRY_STORAGE setting has been changed to SDO_GEOMETRY, ArcSDE creates feature classes with SDO_GEOMETRY columns by default.

Note: Use the sdedbtune administration command to alter DBTUNE settings.

ArcSDE for Oracle supports a number of different geometry storage schemas—these different schemas can all be used together in the same database. While there can only be one default geometry schema, individual tables can be created using different geometry schemas. See Chapter 3, ‘Configuring DBTUNE storage parameters’, for instructions on using DBTUNE to define different geometry schemas.

SDO_GEOMETRY columns

ArcSDE creates a feature class by adding a geometry column to the specified business table. When the GEOMETRY_STORAGE parameter is set to SDO_GEOMETRY, ArcSDE adds an SDO_GEOMETRY column to the business table. In this example, a business table called COUNTRIES has name and population properties—after adding a geometry column, it also has an SDO_GEOMETRY column named BORDERS. A unique feature identifier column (OBJECTID in this example) is added.

NAME	DATA TYPE	NULL?
NAME	VARCHAR2(32)	
POPULATION	NUMBER(11)	
BORDERS	MDSYS.SDO_GEOMETRY	
OBJECTID	NUMBER(38)	NOT NULL

COUNTRIES business table schema

A geometry column can be added to the business table using ArcCatalog, the *sdelayer* administration utility, or the ArcSDE C and Java APIs.

Spatial index

When an SDO_GEOMETRY column is added to a business table, a spatial index on that geometry column is usually created. By default, ArcSDE creates an R-tree index on an SDO_GEOMETRY column. ArcSDE also includes support for creating Oracle Spatial Fixed and Hybrid indexes instead. Alternatively, ArcSDE can create feature classes with no spatial index; however, spatial queries cannot be supported until a spatial index is created. Oracle’s SDO_FILTER, used by ArcSDE, requires the presence of a spatial index.

You can create a spatial index in several ways—in ArcCatalog; with *sdelayer* administration utility, including appropriate DBTUNE configuration parameters; using the Oracle Spatial Index Advisor; using SQL; or programmatically using the ArcSDE C and Java APIs.

ArcSDE automatically drops and recreates Oracle Spatial indexes created by ArcSDE whenever the feature class is switched between LOAD_ONLY_IO and NORMAL_IO mode. Spatial indexes defined by the Oracle Spatial Index Advisor application or created using SQL are not dropped when ArcSDE switches the feature class to LOAD_ONLY_IO mode.

Oracle Spatial metadata

When ArcSDE adds an SDO_GEOMETRY column to a business table, it also adds the required Oracle Spatial metadata record to the USER_SDO_GEOM_METADATA view. This metadata includes the table name, SDO_GEOMETRY column name, spatial reference ID, and coordinate dimension information.

When an ArcSDE client application is used to delete Oracle Spatial tables, it also deletes the Oracle Spatial metadata record for the table. If the ArcSDE command *sdelayer –o delete* is used to unregister a table as a layer, ArcSDE deletes the Oracle Spatial metadata record for the table, unless the table was autoregistered and autoregistration is currently disabled.

Coordinate dimension

You can create ArcSDE geometry as 2D, 2D with measures, 3D, or 3D with measures. When creating new feature classes or adding an SDO_GEOMETRY column to an existing table, ArcSDE defines the Oracle Spatial dimension information (DIMINFO) as:

- The X ordinate is the first dimension.
- The Y ordinate is the second dimension.
- The Z ordinate is the third dimension if the feature class is defined as having elevations.
- The M ordinate is the last dimension (third or fourth, depending on the presence or absence of a Z ordinate) if the feature class is defined as having measures.

The dimension extents for X range, Y range, Z range, and M range may be calculated from the actual data or based on fixed values in the DBTUNE table. See Chapter 3, 'Configuring DBTUNE storage parameters', for DBTUNE storage parameters to establish predefined Oracle Spatial dimension properties and to change the name of any dimension.

Coordinate reference

Oracle Spatial provides predefined coordinate references in the MDSYS.CS_SRS table. When creating Oracle Spatial tables with ArcSDE, you can choose an Oracle Spatial SRID through a DBTUNE configuration keyword. Identify the appropriate Oracle Spatial coordinate reference description and set the feature class's SDO_SRID DBTUNE storage parameter to that value. For example:

```
SDO_SRID 8307
```

If the SDO_SRID storage parameter is not set in DBTUNE, the SRID of each SDO_GEOMETRY value will be NULL, as will the corresponding SRID in the metadata record in the USER_SDO_GEOM_METADATA view.

ArcSDE does not require setting the Oracle Spatial SRID. ArcSDE maintains the coordinate reference information for each feature class in its own SPATIAL_REFERENCES table independently from Oracle Spatial. However, setting the Oracle SRID is useful if you intend to access the data from other applications.

Consult your *Oracle Spatial User's Guide and Reference* for information on supported coordinate references.

Storing geometry in SDO_GEOMETRY

When storing geometry in the database, ArcSDE populates the SDO_GEOMETRY value from an ArcSDE API object called SE_SHAPE. The SE_SHAPE object can contain simple and complex geometry that may include elevations, measures, CAD data, annotation, and surface patches. The SDO_GEOMETRY data type supports a subset of these geometric properties. Because there is not a one-to-one mapping of the components in the SDO_GEOMETRY and the SE_SHAPE object, ArcSDE follows this set of rules when storing ArcSDE data in Oracle Spatial tables.

- Create four-digit SDO_GTYPE based on the geometry's entity type.
- The SDO_SRID is set to NULL if no Oracle Spatial coordinate reference is provided in a DBTUNE parameter.
- Coordinate values are written using the appropriate coordinate reference system.
- Coordinates are written as X, Y, <Z>, and <M>, where the elevation and measure ordinates are only defined if present in the source SE_SHAPE object.
- If elevations and/or measure ordinates are present in the source SE_SHAPE object, all coordinates are stored with an elevation and/or measure ordinate.
- Elevations and measure ordinates are set to NaN in a specific geometry if they do not contain an elevation or measure.
- Measures may be present on any geometry type (not just linestrings), and the first and last coordinates need not contain measure values.
- Measures are not restricted to ascending or descending order.
- Circular curves are written to the SDO_GEOMETRY type when the layer is registered with the CAD entity mask ("c"); otherwise, the curves are converted to densified linestrings that closely represent the curve.
- Other nonlinear interpolated shapes (cubic spline, Bézier curves) are converted to densified linestrings. (See "CAD and annotation properties" below.)

- All ArcSDE-generated SDO_GEOMETRY values are generated from SE_SHAPE objects that have passed the ArcSDE rigorous geometry validation. The validation rules are described in the ArcSDE Developer Guide.

- Single part, non-nil (-e p) 2D or 3D point layers are stored in the SDO_POINT member of the SDO_GEOMETRY object. Other variations of point features are stored in the SDO_ORDINATE_ARRAY.

- ArcSDE does not support heterogeneous geometry collection in the SDO_GEOMETRY object.

- ArcSDE does not encode SDO_ETYPE 0 elements in the SDO_GEOMETRY object. SDO_ETYPE 0 elements are application-specific. See 'Interoperability considerations' below for more information.

CAD and annotation properties

The SDO_GEOMETRY type cannot store all types of data that ArcSDE storage must support. When storage of this data may be required (as determined from the geometry type flags specified when the layer is created), ArcSDE adds a column called SE_ANNO_CAD_DATA to the business table.

NAME	DATA TYPE	NULL?
NAME	VARCHAR2(32)	
POPULATION	NUMBER(11)	
BORDERS	MDSYS.SDO_GEOMETRY	
SE_ANNO_CAD_DATA	BLOB	
OBJECTID	NUMBER(38)	NOT NULL

COUNTRIES business table schema with the addition of CAD data

Whenever ArcSDE detects that the data source has CAD data, ArcSDE writes a simple geometric representation of the CAD data into the SDO_GEOMETRY value and writes the unmodified CAD data into the SE_ANNO_CAD_DATA value. If the data source does not have CAD data, ArcSDE sets the SE_ANNO_CAD_DATA value to NULL.

The SE_ANNO_CAD_DATA property contains data from numerous ArcGIS components:

- AutoCAD or MicroStation data from ArcSDE CAD Client

- Parametric objects, such as cubic splines and Bézier curves from ArcMap

- Surface patches from ArcMap Spatial Analyst

- Annotation from ArcInfo Workstation (pre-ArcGIS 8)

Spatial queries

ArcSDE uses the Oracle Spatial SDO_FILTER function to perform the primary spatial query. ArcSDE performs secondary filtering of the SDO_GEOMETRY based on the spatial relationship requested by the application.

Applications may also include Oracle Spatial primary and secondary filter functions in the SQL Where clause supplied to ArcSDE. Using spatial filters in the Where clause, applications can distribute the spatial query to the database server, the ArcSDE application server, and the application itself.

How ArcSDE uses existing Oracle Spatial tables

ArcSDE is designed to use tables containing SDO_GEOMETRY columns that were created by other applications or using SQL (sometimes referred to as third party tables), so long as the tables meet certain prerequisites.

Manual registration

You can use the ArcSDE administration command *sdelayer –o register* to manually register a table as a layer. Manually registering tables as layers gives you more control over how a table is registered than autoregistration does. Here is an example of registering a table called TBL which contains point features (-e p) in a spatial column called SHAPE. The table has an integer column, PID, that will be the user-maintained unique feature identifier column (-C pid,USER).

```
sdelayer -o register -l tbl,shape -e p -C pid,USER -u <user> -p <pw>
```

Keep in mind, for a table to be registered with ArcSDE, it must meet at least the following criteria:

- It must be owned by the user registering the table.
- It must have a single SDO_GEOMETRY column.
- It must have no other columns of a user-defined type.
- It must have a valid entry in the view USER_SDO_GEOM_METADATA.
- It must have a single type of geometry (points, lines, or polygons). Geometry can be multipart.

- It must have an integer, unique, not-NULL column suitable as a registered row ID column. If the row ID column will be maintained by ArcSDE, it must be NUMBER(38) UNIQUE NOT NULL.
- It should have a spatial index; otherwise, spatial queries are not possible.
- It should pass Oracle's geometry validation tests; otherwise, accessing these geometries may have unexpected results.

Autoregistration

Autoregistration is a feature of ArcSDE that can discover and register Oracle Spatial tables, allowing ArcSDE to access these tables as ArcSDE layers.

Autoregistration is controlled by the system configuration parameter DISABLEAUTOREG, which is set to TRUE by default. This means that the default state for autoregistration is off. To turn on autoregistration, use the following administration command:

```
sdeconfig -o alter -v DISABLEAUTOREG=FALSE -u sde -p <sde_password>
```

Note: As of ArcSDE 9, the environment and dbinit.sde variable SDEDISABLEAUTOREG is no longer used to disable autoregistration.

ArcSDE provides a list of registered layers to client applications at certain times, such as when a user connects to the server using ArcCatalog or when running the command *sdelayer –o describe*. In the process of providing this list, ArcSDE autoregistration queries the Oracle Spatial metadata view, ALL_SDO_GEOM_METADATA, for tables containing SDO_GEOMETRY columns in which the user has SELECT privileges and that are not already registered as ArcSDE layers. When a table is discovered that matches these criteria, ArcSDE automatically registers it, making the table available to ArcSDE client applications.

To register the table, ArcSDE must know the geometry type. Autoregistration looks at the first row in a newly discovered table to establish the ArcSDE geometry type. Autoregistration will not register empty tables. If you know that the table contains rows with differing geometry types, you can use the *sdelayer* administration utility to alter the geometry type definition after autoregistration completes. Layers with multiple geometry types are not visible in ArcGIS 8 or later.

ArcSDE with Oracle Spatial requires a column to uniquely identify each row. See the section ‘Registered row ID column,’ later in this appendix. Autoregistration searches for a column in the table to use as a unique feature identifier column. To qualify, the column must be defined as NUMBER(38) UNIQUE NOT NULL. If such a column is found, it is recorded in the ArcSDE table registry as the unique feature identifier column for the table. If a suitable unique feature identifier column is not found, the table is registered, but operations requiring a unique feature identifier are unavailable.

For SDO_GEOMETRY columns that have an Oracle Spatial SRID, ArcSDE stores the information in the ArcSDE SPATIAL_REFERENCES table. ArcSDE sets its spatial reference AUTH_NAME field to ORACLE and the AUTH_SRID field to the SRID value. ArcSDE tests the coordinate reference description and, if it is valid, sets the SRTEXT field to the Oracle Spatial coordinate reference description.

Interoperability considerations

There is a common misconception that applications can interoperate simply because they support the same underlying geometry type. The geometry type is only one aspect of the interoperability picture—a common understanding of rules, constraints, schema, and implementation is also required.

Multiple SDO_GEOMETRY columns in a table

ArcSDE does not support multiple geometry columns in a table. Tables with multiple SDO_GEOMETRY columns should be accessed through views that contain only one SDO_GEOMETRY column.

Use the sdetable create_view operation to create views of business tables, or create a view with SQL and manually register it with ArcSDE. Be sure to include the row ID column from the same table from which you included the SDO_GEOMETRY column.

Single geometry type in an SDO_GEOMETRY column

While ArcSDE supports multiple geometry types in a geometry column, many applications do not. For example, ArcGIS requires that a geometry column be restricted to a single geometry type.

Oracle Spatial does not necessarily enforce geometry type constraints on an SDO_GEOMETRY column. Without this enforcement, one application may want to enforce a single geometry type constraint, but another application could insert different geometry types.

One option to enforce geometry type constraints is to create an insert–update trigger that checks the SDO_GEOMETRY GTYPE property to enforce geometry types. Using Oracle9*i*, you have an additional option to constrain the geometry type on insert into a table with a spatial index that was created with a special parameter.

Geometry validation

ArcSDE validates geometry when inserting or updating geometries. However, Oracle Spatial itself does not automatically enforce geometry validation on the insert or update of an SDO_GEOMETRY value if you insert or update geometries using methods other than ArcSDE clients. You can create an insert–update trigger to fire the SDO_VALIDATE function to enforce validation of SDO_GEOMETRY types.

Note: As of ArcSDE 9.1, SDO_GEOMETRY features are not validated by ArcSDE each time they are fetched. They are still validated when stored.

Problems would occur if illegal or poorly formed geometry values were passed to ArcSDE client applications. To reduce the occurrences of these potential problems, you should validate any geometries created or changed by any method other than an ArcSDE client application. This validation should be done using the *sdelayer –o feature_info* administration command. The feature_info operation of the *sdelayer* command optionally includes, as part of its output information, whether or not a shape's geometry is valid for ArcSDE. For details on the use of this operation, refer to the Administration Command Reference documentation included with your ArcSDE installation. Here is an example of using *sdelayer –o feature_info* to find features that fail ArcSDE shape validation.

```
sdelayer -o feature_info -l tbl,shape -r invalid . . .
```

ArcSDE geometry validation is not the same as Oracle Spatial geometry validation. While it is important that geometries pass Oracle Spatial validation, successfully validating geometries with Oracle's validation routines does not guarantee that the geometries will be suitable for ArcSDE.

Heterogeneous geometry collections

ArcSDE does not support heterogeneous geometry collections in an SDO_GEOMETRY object.

SDO_ETYPE 0 elements

Oracle Spatial allows applications to insert application-specific data into an SDO_GEOMETRY object. Applications do this by embedding this data using an SDO_ETYPE value of 0. This allows applications great flexibility to store many types of unconventional geometry and other data. However, the nature of the application-specific data is known only to the application that generated that special SDO_GEOMETRY object. Such application-specific data cannot be reliably supported in an interoperable environment. Applications reading SDO_GEOMETRY objects probably would not know how to interpret SDO_ETYPE 0 data created by other applications. Applications updating SDO_GEOMETRY objects would not know how to edit or preserve the SDO_ETYPE 0 data.

When reading SDO_GEOMETRY objects containing SDO_ETYPE 0 elements, ArcSDE will ignore the SDO_ETYPE 0 data and will pass only the geometry elements it supports to the application.

When updating SDO_GEOMETRY objects containing SDO_ETYPE 0 elements, ArcSDE will not preserve the SDO_ETYPE 0 data. Therefore, applications that need to ensure that SDO_ETYPE 0 data is preserved should ensure that users do not have UPDATE access to the table.

Coordinate reference

ArcSDE requires that all SDO_GEOMETRY values in a column be in the same coordinate reference system. If the coordinate reference is undefined, the SRID value should be NULL as defined in the *Oracle Spatial User's Guide and Reference.* With Oracle9*i*, Oracle Spatial requires that the SRID in each geometry match each other and the SRID in the spatial metadata, even if it the SRID is NULL.

Identification of elevations and measures

Oracle Spatial and ArcSDE both assume that the first and second dimensions of the SDO_GEOMETRY are X and Y, respectively.

If an SDO_GEOMETRY object has three dimensions, ArcSDE assumes the third dimension is a measure if the dimension name (in the spatial metadata) starts with an "M"; otherwise, the third dimension is assumed to be an elevation. You can control how ArcSDE interprets the third dimension by setting the feature class's entity flag to have either elevations or measures.

Oracle9*i* Release 2 extended the SDO_GTYPE member of the SDO_GEOMETRY type to allow encoding of which dimension contains a measure ordinate. The second digit of the four-digit SDO_GTYPE may be 0, 3, or 4. If it is 3 or 4, this indicates which dimension contains the measure ordinate. ArcSDE 9 recognizes this encoding.

In this example, *sdelayer* sets the entity type of a feature class to store linestrings and elevations:

```
sdelayer -o add -e l3
```

In this example, *sdelayer* sets the entity type of a feature class to store linestrings and measures:

```
sdelayer -o add -e lM
```

If the SDO_GEOMETRY object has four dimensions, the measure is expected to be the last ordinate.

Measures and linear reference system

Oracle Spatial (but not Oracle Locator) provides functions for linear reference system (LRS) calculations using measure values. The Oracle Spatial LRS functions require that all measure values in a geometry be monotonically ascending or descending without NaN values. Oracle Spatial LRS also restricts measure values to linestrings.

ArcSDE allows measures and LRS calculations on all geometric types with support for arbitrarily ordered measure values and NaN values.

If you use Oracle Spatial LRS functions, design your application and database to operate within the Oracle Spatial LRS constraints.

Registered row ID column

ArcSDE needs a unique feature identifier in the spatial table to perform spatial queries, logfile queries, single-row operations, and multiversioned database operations. With SDE compressed binary schema, the geometry column can serve this purpose because it is a foreign key into the feature table and is defined as a non-NULL unique integer value. SDO_GEOMETRY does not include a unique identifying integer value, so when ArcSDE adds the SDO_GEOMETRY column to an existing table, it may also add a unique identifying column. This column is often called OBJECTID, but it can have another name. Or, an existing column can be used for the registered row ID column so long as it is indexed and is an integer, UNIQUE, and NOT NULL. Row ID columns maintained by ArcSDE must be NUMBER (38) UNIQUE NOT NULL.

A unique feature identifier column can be added using the *sdetable* administration command or the ArcCatalog application. Many applications, such as ArcGIS Desktop, require a unique feature identifier column in a table. Each application has its own requirements and limitations.

Multiversioned database

ArcGIS Desktop uses a multiversioned database implemented through ArcSDE for editing operations. The multiversioned database provides long transaction support for multiple simultaneous design alternatives.

Multiversioned views are available for SQL access to the multiversioned database, including Oracle Spatial tables registered as versioned. See the *ArcSDE Developer Guide* for more information.

Networks and topology feature classes

ArcGIS Desktop can create and maintain networks and integrated topological feature classes from simple feature classes that use the SDO_GEOMETRY type. ArcGIS

Desktop manages the relationships and maintains the topological integrity of the data—modifications to the underlying features through ArcGIS Desktop are reflected in these integrated networks.

Modification of Oracle Spatial feature classes participating in these networks should be restricted to ArcGIS Desktop applications. Other applications can freely read the data, but any modifications they make are not reflected in the networks.

Relationships and constraints

ArcGIS Desktop enforces relationships and constraints across many different data sources. Feature classes containing an SDO_GEOMETRY type may be included in relationships and may have constraints defined on them.

Modification of Oracle Spatial feature classes participating in relationships and constraints should be restricted to ArcGIS Desktop applications. Other applications can freely read the data, but their edits are not properly handled.

ArcSDE Oracle Spatial Support

Visit the ESRI Online Support Center (*http://support.esri.com*) for help with Oracle Spatial issues.

APPENDIX E

The well-known binary representation

The well-known binary representation for OGC geometry (WKBgeometry) provides a portable representation of a geometry value as a contiguous stream of bytes. It permits geometry values to be exchanged between an ODBC client and a SQL database in binary form.

The well-known binary representation for geometry is obtained by serializing a geometry instance as a sequence of numeric types drawn from the set {Unsigned Integer, Double} and serializing each numeric type as a sequence of bytes using one of two well-defined, standard binary representations for numeric types, NDR or XDR. The specific binary encoding used for a geometry byte stream is described by a one-byte tag that precedes the serialized bytes. The only difference between the two encodings of geometry is byte order. The XDR encoding is big-endian, while the NDR encoding is little-endian.

Numeric type definitions

An unsigned integer is a 32-bit (4 byte) data type that encodes a nonnegative integer in the range [0, 4294967295].

A double is a 64-bit (8 byte) double-precision data type that encodes a double-precision number using the IEEE 754 double-precision format.

The above definitions are common to both XDR and NDR.

XDR (big endian) encoding of numeric types

The XDR representation of an unsigned integer is big endian (most significant byte first).

The XDR representation of a double is big endian (sign bit is first byte).

NDR (little endian) encoding of numeric types

The NDR representation of an unsigned integer is little endian (least significant byte first).

The NDR representation of a double is little endian (sign bit is last byte).

Conversion between the NDR and XDR representations of WKB geometry

Conversion between the NDR and XDR data types for unsigned integers and doubles is a simple operation involving the reversal of bytes within each unsigned integer or double in the byte stream.

Description of WKB geometry byte streams

The well-known binary representation for geometry is described below. The basic building block is the byte stream for a point that consists of two doubles. The byte streams for other geometries are built using the byte streams for geometries that have already been defined.

```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer  (4 bytes)
// double : double precision number (8 bytes)


// Building Blocks : Point, LinearRing
Point {
      double x;
      double y;
      };
```

```
LinearRing   {
      uint32 numPoints;
      Point  points[numPoints];
      }
```

```
enum wkbGeometryType {
      wkbPoint = 1,
      wkbLineString = 2,
      wkbPolygon = 3,
      wkbMultiPoint = 4,
      wkbMultiLineString = 5,
      wkbMultiPolygon = 6,
      wkbGeometryCollection = 7
      };
enum wkbByteOrder {
   wkbXDR = 0,            // Big Endian
   wkbNDR = 1             // Little Endian
};


WKBPoint {
      byte            byteOrder;
      uint32      wkbType;                  // 1
      Point           point;
      }
WKBLineString {
      byte            byteOrder;
      uint32      wkbType;                  // 2
      uint32      numPoints;
      Point           points[numPoints];
      }
WKBPolygon   {
      byte            byteOrder;
      uint32      wkbType;                  // 3
      uint32      numRings;
      LinearRing      rings[numRings];
      }
WKBMultiPoint {
      byte            byteOrder;
      uint32      wkbType;                  // 4
      uint32      num_wkbPoints;
      WKBPoint        WKBPoints[num_wkbPoints];
      }
```

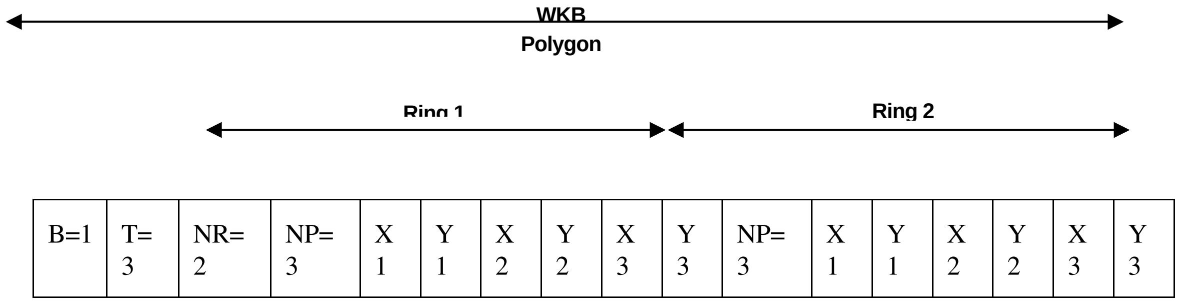
```
WKBMultiLineString  {
      byte            byteOrder;
      uint32      wkbType;                  // 5
      uint32      num_wkbLineStrings;
      WKBLineString WKBLineStrings[num_wkbLineStrings];
      }
```

```
wkbMultiPolygon {
       byte             byteOrder;

       uint32       wkbType;                      // 6
       uint32       num_wkbPolygons;
       WKBPolygon       wkbPolygons[num_wkbPolygons];
       }


WKBGeometry  {
       union {
       WKBPoint                   point;
       WKBLineString           linestring;
       WKBPolygon                 polygon;
       WKBGeometryCollection  collection;
       WKBMultiPoint          mpoint;
       WKBMultiLineString     mlinestring;
       WKBMultiPolygon        mpolygon;
       }
       };
```

```
WKBGeometryCollection {
      byte            byte_order;
      uint32       wkbType;                    // 7
      uint32       num_wkbGeometries;
      WKBGeometry     wkbGeometries[num_wkbGeometries]
      }
```



Well-known binary representation for a geometry object in NDR format (B = 1) of type polygon (T = 3) with two linears (NR = 2) and each ring having three points (NP = 3).

Assertions for well-known binary representation for geometry

The well-known binary representation for geometry is designed to represent instances of the geometry types described in the geometry object model and in the *OpenGIS Abstract Specification.*

These assertions imply the following for rings, polygons, and multipolygons:

Linear rings—Rings are simple and closed, which means that linear rings may not self-intersect.

Polygons—No two linear rings in the boundary of a polygon may cross each other. The linear rings in the boundary of a polygon may intersect, at most, at a single point but only as a tangent.

Multipolygons—The interiors of two polygons that are elements of a multipolygon may not intersect. The boundaries of any two polygons that are elements of a multipolygon may touch at only a finite number of points.

APPENDIX F

Storing locators

A locator is an object that you can use to convert textual descriptions of locations into geographic features. The most common locator is an address locator, which you can use to geocode addresses. For additional documentation on creating and using locators in ArcGIS, see *Geocoding in ArcGIS* in the ArcGIS documentation set.

ArcSDE stores locator definitions in the SDE_locators table. Three main types of locators can be stored in an ArcSDE database:

- Locator styles are used as templates on which to base new locators.
- Locators define the inputs, outputs, logic, and one or more reference datasets that are used to find locations. Locators are usually created by adding some properties to a locator style that specify which reference datasets and which columns in those reference datasets to use to find locations. Using ArcCatalog to create a locator based on a locator style is the easiest way to create a new locator.
- Attached locators are copies of locators that are used to create a geocoded feature class. When you create a geocoded feature class by geocoding a table of addresses using an address locator, ArcSDE stores a copy of the locator that was used to create the geocoded feature class. ArcSDE uses this attached locator when you rematch addresses in the geocoded feature class.

Each locator style, locator, and attached locator has a number of properties that define the locator. ArcSDE stores each property of a locator as a record in the SDE.METADATA table.

Address locators use a set of geocoding rules that define how addresses are parsed, standardized, and matched to the reference data used by the address locator. ArcSDE stores geocoding rules in the SDE.GCDRULES table. Each row in the SDE.GCDRULES table corresponds to a single file in a set of geocoding rules. For information on geocoding rule files, see the *Geocoding Rule Base Developer Guide* in the ArcGIS documentation set.

Many address locators require a geocoding index table for each reference data table. Geocoding index tables are tables used by a locator to quickly search for records in the corresponding reference datasets that may be matches for an address. The XID column in a geocoding index table is a foreign key to the OBJECTID column in the corresponding reference dataset. When you create a new address locator that requires a geocoding index table for a reference dataset, ArcSDE creates the geocoding index table if it does not already exist.

When a locator is instantiated, ArcSDE reads the locator record from the SDE.LOCATORS table and all of the corresponding locator properties from the SDE.METADATA table. Some of the locator properties specify which set of geocoding rules to use, which are read from the SDE.GCDRULES table. Other locator properties specify which feature classes or tables in the ArcSDE database are used as reference datasets and which geocoding index tables, if any, correspond to these reference datasets.

When you use a locator to geocode an address, the locator uses the specified geocoding rules to parse the given address into its components. If the locator uses geocoding index tables to index the reference data, the locator properties specify which of these address components to use to search for matches in the geocoding index table or tables and which transformations (usually the Soundex function) to apply to the address components when searching for records in the geocoding index table. ArcSDE searches for records matching the geocoding index query in the geocoding index table. The resulting set of records from the geocoding index table is joined to the corresponding reference data table to generate a set of candidates for the address. ArcSDE uses the locator's properties to determine which columns in the reference data feature class or table correspond to address components used by the locator and uses the geocoding rules to assign a score to each candidate.

Locator schema

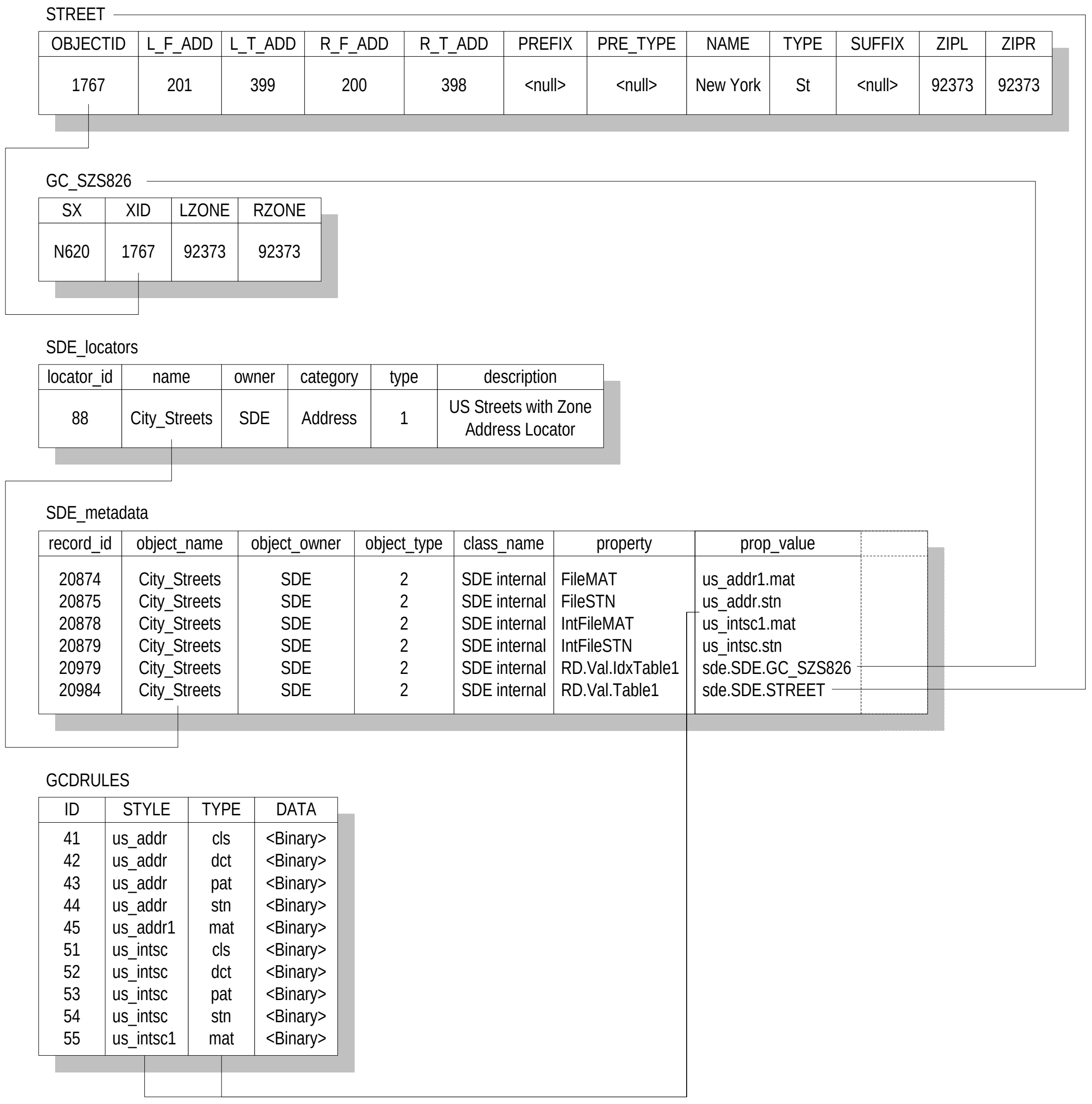
When you create a locator in an ArcSDE database, ArcSDE adds a record to the SDE.LOCATORS table that defines the locator. ArcSDE also adds a record to the SDE.METADATA table for each property of the locator. The object_name column in the SDE.METADATA table is a foreign key to the Name column in the SDE.LOCATORS table that ArcSDE uses to associate locators with their properties.

Each locator has associated FileMAT and FileSTN properties in the SDE.METADATA table that define which geocoding rules the locator uses. The values of these properties are in the format *style.type*, and define which geocoding rule files, stored in the

SDE.GCDRULES table, the locator uses to match addresses. The locator uses the value of these properties in the SDE.METADATA table to query the SDE.GCDRULES table on the STYLE and TYPE columns to retrieve the correct set of geocoding rules. Locators that support intersection geocoding have associated IntFileMAT and IntFileSTN properties that define the geocoding rules to use for intersection geocoding.

When you create an address locator, ArcSDE may create one or more geocoding index tables for the reference datasets used by the locator, depending upon the locator style on which the address locator is based. Geocoding index table names are prefixed with "GC_" and include characters identifying the type of geocoding index table and the Geodatabase object class ID of the table or feature class that it indexes. The XID column in a geocoding index table is a foreign key to the OBJECTID column in the table or feature class that the geocoding index table indexes.

In the example that follows, an ArcSDE database contains a STREET feature class that represents street centerlines for a particular geographic area, such as a city. In addition to the geometry for the street centerlines, the STREET feature class contains attributes for the address ranges that can be found along the street and the components of the street name. The ArcSDE table schema required to store a locator to allow address geocoding on this feature class is shown below.



STREET

OBJECTID	L_F_ADD	L_T_ADD	R_F_ADD	R_T_ADD	PREFIX	PRE_TYPE	NAME	TYPE	SUFFIX	ZIPL	ZIPR
1767	201	399	200	398	<null>	<null>	New York	St	<null>	92373	92373

GC_SZS826

SX	XID	LZONE	RZONE
N620	1767	92373	92373

SDE_locators

locator_id	name	owner	category	type	description
88	City_Streets	SDE	Address	1	US Streets with Zone Address Locator

SDE_metadata

record_id	object_name	object_owner	object_type	class_name	property	prop_value
20874	City_Streets	SDE	2	SDE internal	FileMAT	us_addr1.mat
20875	City_Streets	SDE	2	SDE internal	FileSTN	us_addr.stn
20878	City_Streets	SDE	2	SDE internal	IntFileMAT	us_intsc1.mat
20879	City_Streets	SDE	2	SDE internal	IntFileSTN	us_intsc.stn
20979	City_Streets	SDE	2	SDE internal	RD.Val.IdxTable1	sde.SDE.GC_SZS826
20984	City_Streets	SDE	2	SDE internal	RD.Val.Table1	sde.SDE.STREET

GCDRULES

ID	STYLE	TYPE	DATA
41	us_addr	cls	<Binary>
42	us_addr	dct	<Binary>
43	us_addr	pat	<Binary>
44	us_addr	stn	<Binary>
45	us_addr1	mat	<Binary>
51	us_intsc	cls	<Binary>
52	us_intsc	dct	<Binary>
53	us_intsc	pat	<Binary>
54	us_intsc	stn	<Binary>
55	us_intsc1	mat	<Binary>

Business table

In this example, the STREET feature class represents street centerlines within a particular geographic area and contains attributes that allow address locators to geocode addresses using this feature class.

NAME	DATA TYPE	NULL?
OBJECTID	NUMBER(38)	NOT NULL
L_F_ADD	NUMBER(38)	NULL
L_T_ADD	NUMBER(38)	NULL
R_F_ADD	NUMBER(38)	NULL
R_T_ADD	NUMBER(38)	NULL
PREFIX	VARCHAR2(2)	NULL
PRE_TYPE	VARCHAR2(5)	NULL
NAME	VARCHAR2(30)	NULL
TYPE	VARCHAR2(5)	NULL
SUFFIX	VARCHAR2(2)	NULL
ZIPL	VARCHAR2(5)	NULL
ZIPR	VARCHAR2(5)	NULL
Shape	NUMBER(38) or SDO_GEOMETRY	NULL

STREET business table

- OBJECTID (SE_INTEGER_TYPE)—The table's primary key
- L_F_ADD (SE_INTEGER_TYPE)—The address at the start node on the left side of the street feature
- L_T_ADD (SE_INTEGER_TYPE)—The address at the end node on the left side of the street feature
- R_F_ADD (SE_INTEGER_TYPE)—The address at the start node on the right side of the street feature
- R_T_ADD (SE_INTEGER_TYPE)—The address at the end node on the right side of the feature
- PREFIX (SE_STRING_TYPE)—The prefix direction component of the street's name
- PRE_TYPE (SE_STRING_TYPE)—The prefix type component of the street's name
- NAME (SE_STRING_TYPE)—The base component of the street's name
- TYPE (SE_STRING_TYPE)—The suffix type component of the street's name
- SUFFIX (SE_STRING_TYPE)—The suffix direction component of the street's name

- ZIPL (SE_STRING_TYPE)—The ZIP Code on the left side of the street feature
- ZIPR (SE_STRING_TYPE)—The ZIP Code on the right side of the street feature
- Shape (SE_SHAPE_TYPE)—Contains the geometry for the feature class

Geocoding index table (GC_SZS<objectclass_id>)

When you create a locator that uses an ArcSDE feature class as reference data, the locator style on which the locator is based may specify that a geocoding index table is used when performing geocoding queries against the feature class. The locator style defines the format of the name of the geocoding index table, as well as the contents. In this example, a locator based on the US Streets with Zone locator style was created on the STREETS feature class. Geocoding index tables created by locators based on this style contain a Soundex value for the street name, as well as attributes for the zones on each side of the street feature.

The size of the delta tables also depends on how often records are removed. These tables shrink only when the states preceding the level 0 version are compressed. This occurs only after a version branching directly off the root of the version tree completes and is removed from the system. The compression of states that follows will cause the changes of the states between the level 0 version and the next version following the one removed to be written to the business table and deleted from the delta tables.

NAME	DATA TYPE	NULL?
SX	VARCHAR2(4)	NULL
XID	NUMBER(38)	NULL
LZONE	VARCHAR2(5)	NULL
RZONE	VARCHAR2(4)	NULL

Geocoding index table

- SX (SE_STRING_TYPE)—The Soundex value for the street name
- XID (SE_INTEGER_TYPE)—A foreign key to the OBJECTID column in the business table
- LZONE (SE_STRING_TYPE)—The zone on the left side of the street feature
- RZONE (SE_STRING_TYPE)—The zone on the right side of the street feature

SDE.LOCATORS table

When you add a locator to an ArcSDE database, ArcSDE adds a row to the SDE.LOCATORS table. Each row in the SDE.LOCATORS table defines a locator or locator style.

NAME	DATA TYPE	NULL?
Locator_id	NUMBER(38)	NOT NULL
Name	VARCHAR2(32)	NOT NULL
Owner	VARCHAR2(32)	NOT NULL
Category	VARCHAR2(32)	NOT NULL
Type	NUMBER(38)	NOT NULL
Description	VARCHAR2(64)	NULL

SDE locators table

- locator_id (SE_INTEGER_TYPE)—The table's primary key
- name (SE_STRING_TYPE)—The name of the locator
- owner (SE_STRING_TYPE)—The name of the ArcSDE user that owns the locator
- category (SE_STRING_TYPE)—The category of the locator; address locators have a category value of "Address"
- type (SE_INTEGER_TYPE)—The type of locator; values in this column are represented as follows:
    - 0—Defines locator styles
    - 1—Defines locators (that is, locators that can be used to find locations)
    - 2—Defines attached locators (that is, locators that are attached to a geocoded feature class and are a copy of the locator and the geocoding options that were used to create the geocoded feature class)
- description (SE_STRING_TYPE)—The description of the locator

SDE.METADATA table

When you add a locator to an ArcSDE database, ArcSDE adds a row to the SDE.METADATA table for each property of the locator. Each row in the SDE.METADATA table defines a single property for a locator. The object_name column is a foreign key to the name column in the SDE.LOCATORS table that ArcSDE uses to associate a locator with its properties.

NAME	DATA TYPE	NULL?
record_id	NUMBER(38)	NOT NULL
object_name	VARCHAR2(32)	NOT NULL
object_owner	VARCHAR2(32)	NOT NULL
object_type	NUMBER(38)	NOT NULL
class_name	VARCHAR2(32)	NULL
property	VARCHAR2(32)	NULL
prop_value	VARCHAR2(255)	NULL
description	VARCHAR2(65)	NULL
creation_date	DATE	NOT NULL

SDE metadata table

- record_id (SE_INTEGER_TYPE)—The table's primary key
- object_name (SE_STRING_TYPE)—The name of the locator to which the property belongs
- object_owner (SE_STRING_TYPE)—The name of the ArcSDE user that owns the record
- object_type (SE_INTEGER_TYPE)—Always a value of 2 for locator properties
- class_name (SE_STRING_TYPE)—Always a value of "SDE_internal" for locator properties
- property (SE_STRING_TYPE)—The name of the locator property
- prop_value (SE_STRING_TYPE)—The value of the locator property
- description (SE_STRING_TYPE)—Not used for locator properties
- creation_date (SE_DATE_TYPE)—The date and time at which the locator property was created

SDE.GCDRULES table

The SDE.GCDRULES table stores the geocoding rules that are used by address locators to match addresses. Each record in the SDE.GCDRULES table corresponds to a geocoding rule file. For descriptions of each of the geocoding rule files and their contents, see the *Geocoding Rule Base Developer Guide* in the ArcGIS documentation set.

NAME	DATA TYPE	NULL?
ID	NUMBER(9)	NOT NULL
STYLE	VARCHAR2(32)	NULL
TYPE	VARCHAR2(3)	NULL
DATA	BLOB or LONG RAW	NULL

Geocoding rules table

- ID (SE_INTEGER_TYPE)—The table's primary key
- STYLE (SE_STRING_TYPE)—The name of the geocoding rule set
- TYPE (SE_STRING_TYPE)—The type of geocoding rule file
- DATA (SE_BLOB_TYPE)—The contents of the geocoding rule file

APPENDIX G

Making a direct connection

Direct connect is another configuration option for ArcSDE, and all the ArcSDE concepts and prerequisites also apply to direct connect. The main difference between the ArcSDE application server and direct connect is where the ArcSDE processing takes place. The purpose of this appendix is to provide information to administrators on how to set up and configure direct connect configurations for the database as well as client machines. If using the application server exclusively, you do not need this appendix.

What files do you need?

There are two sets of ESRI-supplied files required for direct connect:

- Direct connect driver—These are dynamically linked libraries in the bin or lib directory (depending on your operating system) of your client application that provide the functionality to connect and use spatial data in a DBMS. There are drivers for the following databases:
  - IBM DB2
  - IBM Informix
  - Microsoft SQL Server
  - Oracle8*i* and Oracle9*i*

  These drivers are automatically installed for ArcGIS (the whole product suite), ArcView GIS 3.x Database Access, ArcIMS, ArcInfo Workstation, and MapObjects 2. If you are using a non-ESRI custom application built from the ArcSDE C API, you may need to install the direct connect drivers from the

ArcSDE Developer Kit CD–ROM located in the ArcSDE media kit. Check with the supplier of your non-ESRI custom application.

- Database setup files—These are files needed by an administrator to set up and configure a DBMS for direct connect and include files such as sdesetup<dbms>. The setup is exactly the same as it is for the ArcSDE application server. These setup files are located on the platform CD–ROM of choice in the ArcSDE media kit. To get them, you must install ArcSDE for your database. You do not have to create an application server; you only need the files on disk, so you can use them against your database.

DBMS considerations are as follows:

Oracle8*i*, Oracle9*i*
To facilitate network communication to an Oracle database, each client machine where direct connect is used must have Oracle Net installed.

Microsoft SQL Server 7, Microsoft SQL Server 2000
SQL Server requires Microsoft® Data Access Components (MDAC).

If you intend to use ArcCatalog 9 or ArcView GIS 3.3 with Database Access 2.1f, MDAC version 2.6 (SP1) or greater is required. If using ArcIMS 9 or ArcGIS 9 to direct connect, you must have MDAC 2.6 or higher.

DB2
Each client machine must be configured for remote database access. Use the DB2 Configuration Assistant on the database host to connect to a remote database.

Informix
Each client machine where direct connect will be used must have the Informix Client Software Development Kit (SDK) 2.8 or the Informix I-connect 2.8 application installed. The client machine must also have the SetNet32 application installed, which comes with both the Informix Client SDK 2.8 and the Informix I-connect 2.8 applications.

How to get your database setup files

You will need to get your database setup files from one of the CD–ROMs in the ArcSDE media kit. The ArcSDE media kit has CD–ROMs by platform with the exception of the *ArcSDE Developer Kit* CD–ROM. To get your database setup files, you will need to install the software for the ArcSDE application server for your database or platform. For example, if you are using IBM DB2 on a Sun Solaris server, you will use the Sun Solaris CD–ROM from the ArcSDE media kit and install the DB2 version of ArcSDE on your Sun Solaris server. Be sure to follow the postinstallation configuration instructions in the database-specific install guide, but ignore any instructions about creating the application server. You don't need to do that. Install guides are html files on each CD–ROM. Please read them carefully.

Why do I need to install the ArcSDE application server software?

Installation of the ArcSDE application server is to get the database setup and administration files only. If you are a direct connect only site, you do not need to start an ArcSDE application server. All you need to do is install the ArcSDE files and follow the postinstallation configuration instructions. The administration files that are installed (for example, sdesetup<dbms>, sdeconfig, sdedbtune, sdelayer) are useful for managing your connection parameters, dbtune table, and manual registration or unregistration of third party layers. See *Managing ArcSDE Services* and the ArcSDE Configuration and Tuning Guides for more information.

If you use both the application server and direct connect at your site, you already have or soon will have ArcSDE setup and administration files installed anyway. It is important to note that once your database is configured for use with the ArcSDE application server, it is also ready for direct connect usage.

Environment variables

For each client machine, there are environment variables you must set. If necessary, ask your Windows or UNIX system administrator to find out how to set environment variables on your system.

The SDEHOME environment variable

You must set the SDEHOME variable to tell the client application:

- Where the direct connect driver files are stored. For ESRI client applications, the direct connect files are located in the same directory where the client application's other dynamically linked library files are installed. For Windows applications, this is normally in the bin directory of your client application's install location. For UNIX and Linux systems, these will normally be in the lib directory.

To set this environment variable, you must specify the full file path for it. For example:

UNIX: setenv SDEHOME /unix1/arcgis/

Windows: Use Windows utilities to set a variable to something similar to:

```
Variable:   Value:
SDEHOME     C:\Program Files\ArcGIS\
```

The direct connect process will look for the appropriate driver in the bin or lib directories of the path specified.

You do not have to set the SDEHOME environment variable if the following are true:

- Your users are using ESRI client applications built with the ArcSDE 9 C API.
- Your users are not using UNIX.

UNIX or Linux systems

1. Include $SDEHOME/lib in the library environment variable for your platform.

   If your database is an Oracle database, include $ORACLE_HOME/lib as well.

   For example:

   ```
   setenv LD_LIBRARY_PATH $SDEHOME/
   lib:$ORACLE_HOME/lib:/usr/ openwin/lib:/usr/lib
   ```

2. Add the bin directory to the system path.

   An example follows for the SDEHOME variable.

   ```
   setenv PATH $JAVA_HOME/bin:$SDEHOME/
   bin:$AEJHOME/bin:/usr/sbin:/usr/bin:/usr/local/
   bin: /etc:/usr/ucb:/usr/dt/bin:/usr/bin/X11
   ```

3. If ArcIMS is your client application and Oracle is the database, append $ORACLE_HOME/lib to the LD_LIBRARY_PATH variable in the aimsappsrvr and aimsmonitor scripts, located in the $AIMSHOME/Xenv directory.

   For example, where your LD_LIBRARY_PATH variable now reads:

   ```
   LD_LIBRARY_PATH-$AIMSHOME/lib:$AIMSHOME/bin;
   export LD_LIBRARY_PATH
   ```

   It should now be:

   ```
   LD_LIBRARY_PATH-$AIMSHOME/lib:$AIMSHOME/
   bin:$ORACLE_HOME/lib; export LD_LIBRARY_PATH
   ```

The etc directory

If an etc directory exists for the client application, it must be located in the directory you specified for SDEHOME. If it isn’t located there, you must create it there. This etc directory is where the log file of error messages will be stored by default.

The dbinit.sde file

This file is located in the etc directory of your SDEHOME. This file can be used to set environment variables for direct connect use. It may be more convenient to set environment variables for direct connect here than via system tools.

See Chapter 3 in *Managing ArcSDE Application Servers* for more information on the dbinit.sde file.

Client/Database compatibility

Direct connect drivers are only compatible with a same vintage database configured for ArcSDE. For example, you cannot direct connect from ArcMap 9 to a database that is still at an 8.3 configuration. You would have to run the ArcSDE 9 setup configuration on that 8.3 database to be able to use direct connect from the ArcMap 9 client.

Registration and authorization

ArcSDE application servers and all direct connect configurations must be registered before use. The end result of the registration process is an authorization file that is used to enable the software for use. Please note that if you are upgrading an existing ArcSDE instance, you may need new keycodes. Please refer to the *ArcSDE Installation Guide* for details.

Setting up clients for Oracle direct connect

Setting up the database

You must set up and configure each database that users will be directly connecting to. Use standard Oracle tools and ArcSDE tools and documentation to:

- Install the application server.
- Perform postinstallation configuration.

When your database is configured for ArcSDE, you are ready to set up your client machines.

Setting up the client machines

When you set up the client machines, you perform the following steps in order on the client machine:

1. Use Oracle tools to install and configure Oracle Net to connect to an Oracle instance with SQLPLUS.

2. Set environment variables.

3. Test the connection.

Setting up Oracle Net

Oracle Net is required on each client machine you want to access your database directly.

1. On each client machine for which you want to set up a direct connection, install Oracle Net. See your Oracle documentation for instructions on Oracle Net installation.

2. Test that Oracle Net was installed successfully. On the client machine, type the following at a command prompt:

   ```
   sqlplus <user name>/<password>@<Oracle Net service name>
   ```

If you get the SQL> prompt, then the Net client is working and you are ready to go to the next step. If you do not get the SQL prompt, then consult the Oracle documentation's troubleshooting information.

Setting environment variables

You must set the SDEHOME and other environment variables. Set SDEHOME to point to the directory where the client application's dynamically linked library files are stored.

Connection syntax

There is a particular syntax to use when connecting with direct connect. For the Service (or instance) value:

```
sde:oracle
```

where

```
sde:oracle
```

is a required part of the syntax for Oracle8*i* connections

For Oracle 9i connections, use the syntax:

```
sde:oracle9i
```

in order to use the appropriate driver file.

Your password will have the Oracle Net Service name appended to it:

```
mypassword@<Oracle Net Service name>
```

You can also use the following syntax in place of adding the @<Oracle Net Sevice name> to your password:

```
sde:oracle:/;LOCAL=<sqlnetalias>
```

For UNIX this will be:

```
"sde:oracle:/;LOCAL=<sqlnetalias>"
```

(double quotes are required for UNIX).

Testing the connection from the client application

Test the connection from the client application you set up for direct connect.

APPENDIX H

Storing XML data

ArcIMS relies on the capabilities of ArcSDE and a relational database to store, search for, retrieve, and index metadata documents. Whenever ArcIMS receives a request to perform an operation, such as finding a metadata document or publishing a new document, ArcSDE translates the request into SQL and sends it to the database. The result of the operation is passed back from the database and out through ArcIMS. For example, the result might list the metadata documents that matched the search criteria or indicate the success or failure of publishing a document.

The database is a critical component of a Metadata Service. As the database administrator, you must ensure the database is configured properly to support text indexing before a Metadata Service can be started. You also have some control over how the Metadata Service's tables are stored in the database and how documents published to the Metadata Service are indexed. There are additional ArcSDE configuration and maintenance topics in this appendix to be aware of in connection with ArcIMS Metadata Services.

An ArcSDE XML column is used to store metadata documents that are published to a Metadata Service. ArcSDE XML columns are supported with Oracle, SQL Server, and DB2 databases and require text indexing functionality to be installed and configured in those databases. ArcSDE

XML columns, and therefore ArcIMS Metadata Services, are not supported with Informix databases.

Configuring ArcSDE for Oracle

ArcSDE XML columns require Oracle Text for Oracle10*g*™ and Oracle9*i*™ or InterMedia Text for Oracle8*i*™ to be installed. Once this is accomplished, the database and ArcSDE must be properly configured and maintained for optimal search performance.

XML documents are stored in the database as an Oracle Large Object (LOB). The LOB storage and caching parameters, the table space's block size, and the buffer cache configuration all affect the speed at which LOB data is retrieved, which in turn affects performance when searching. It is also important to manage the text indexes properly to maintain good performance.

Installing Oracle's text component

The database administrator must accomplish steps 1 and 2 below before an XML column, and therefore an ArcIMS Metadata Service, can be created.

Step 1: Install the text component.

1. Follow the instructions in the Oracle documentation to install the appropriate text component for your version of the database.

Oracle Text is installed by default with a typical installation of Oracle10*g* and Oracle9*i*. However, if you upgraded an earlier version of the database where the text component was not installed, Oracle Text may not be installed as part of the upgrade; if this is the case, it must be installed as an additional component after the upgrade is completed.

2. Follow the instructions in the Oracle documentation to verify the text component has been installed and is working properly.

Step 2: Grant privileges for the XML column.

After logging in to SQL*Plus as the ctxsys user, you must grant privileges to the user who will own the XML column by issuing the command

```
sql>grant execute on ctx_ddl to <mdUsername>;
```

where <mdUsername> is the user who will own the column. For ArcIMS Metadata Services, <mdUsername> is the user specified in the service's ArcXML configuration file.

Configuring LOB storage

XML documents are stored in BLOB format in the XML document table in the xml_doc and xml_doc_val columns. It is important to configure these columns accurately to achieve the best possible search performance.

Using inline or out of line storage

LOBs are stored inline if the LOB data is stored in the same block as the rest of the data in the row. However, inline storage is only possible if the LOB data is less than 4KB in size. With out of line storage, the data is stored in the LOB segment, and only the LOB locator is stored with the rest of the data in the row.

By default, Oracle stores LOB data inline. You can specify whether LOB data associated with an XML column is stored inline or out of line using the ArcSDE DBTUNE parameters XML_DOC_LOB_STORAGE and XML_DOC_VAL_LOB_STORAGE.

When LOB data is stored out of line for an XML column, by default ArcSDE places that data in the same table space as the XML document table. The LOB data may be moved to a different table space than the one containing the XML document table by using the XML DBTUNE parameters XML_DOC_STORAGE and the XML_DOC_VAL_STORAGE.

A typical XML document that contains metadata describing a GIS resource will be greater than 4KB in size. Tests show XML columns associated with ArcIMS Metadata Services perform best when the LOB data is stored out of line in a separate table space from the XML document table. Specify out of line storage for the XML column by setting the XML_DOC_LOB_STORAGE and XML_DOC_VAL_LOB_STORAGE parameters to DISABLE STORAGE IN ROW. And, after configuring another table space to store the LOB data, specify that table space's name in the XML_DOC_STORAGE and the XML_DOC_VAL_STORAGE parameters. For XML columns associated with ArcIMS Metadata Services, set these parameters with the DEFAULTS DBTUNE keyword.

A Metadata Service may contain gazetteer data instead of typical metadata XML documents. Gazetteer data is very small, typically less than 100 bytes in size. Metadata Services containing gazetteer data will perform best when the LOB data is stored inline. Default gazetteer data is provided with ArcIMS in sdeexport format and is loaded into ArcSDE using a batch script that calls the sdeimport command. This script references the DBTUNE keyword GAZETTEER, which does not specify “DISABLE STORAGE IN ROW” for the XML_DOC_LOB_STORAGE and XML_DOC_VAL_LOB_STORAGE parameters; Oracle’s default behavior will be used to store LO B data inline.

For more information about inline and out of line storage, see Oracle’s *LOB Performance Guidelines* white paper dated May 2004. It is available from Oracle’s Web site.

LOB caching and logging

LOB data may be read and written using the buffer cache. When the LOB data isn’t accessed using the buffer cache, it is read and written by directly accessing the data each time. Caching can improve search performance for XML columns where LOB data is stored out of line, particularly for frequently accessed LOBs. By default, Oracle will not cache LOB data that is stored out of line.

If logging is specified, changes to out of line LOB data will be added to the redo logs. Logging is enabled when LOB data is cached. This may decrease performance for publishing XML documents.

With ArcSDE XML columns, LOB data will initially be stored as NOCACHE NOLOGGING. However, XML columns that store LOB data out of line should be set to CACHE LOGGING to optimize search performance. For XML columns associated with an ArcIMS Metadata Service, both LOB columns in the XML document table will be set to CACHE LOGGING when the aimsmetaindx command is run.

The caching and logging properties will not affect LOB data that is stored inline such as with XML columns that contain gazetteer data.

For more information about caching and logging, see Oracle’s *LOB Performance Guidelines* white paper dated May 2004. It is available from Oracle’s Web site.

Table space block size and buffer cache configuration

When out of line LOB data is not present in the buffer cache, the block size can affect the amount of time required to read data from disk. You can improve search performance against an XML column if you know the data’s distribution and you set the table space and buffer cache to use the correct block size.

For example, if most of the LOB data is less than 8KB in size and the table space block size is 16KB, more time will be spent reading data from the disk than necessary. In this situation, you would improve performance if you used Oracle’s multi-block-size configuration, storing the out of line LOB data in a table space that was set to use 8KB blocks.

Setting the table space block size must be done in combination with allocating memory for the buffer cache. The size of the buffer cache depends on the block size of the table space used to store the out of line LOB data. If the table space uses an 8KB block size, memory must be allocated for an 8KB buffer cache. You must ensure that enough system memory has been allocated to support the cache.

Oracle uses a table space block size of 16KB by default. Initially, you should use a table space with the default block size and buffer cache settings to store out of line LOB data. Then, once XML documents have been published to the XML column, you can analyze the distribution of that data using the xml_lob_block_distribution script that is available in the %SDEHOME%\sdeexe\tools directory.

ESRI tests show that typical metadata XML documents published to an XML column will be less than 8KB in size. To optimize search performance for this type of XML column, you can take advantage of Oracle's multiblock-size configuration to store the out of line LOB data in a table space with an 8KB block size, while other table spaces can continue storing data in 16KB blocks. First, the Oracle instance must be configured to allow a block size of 8KB, then you can create a new table space that uses the 8KB block size and 8KB buffer cache to store the out of line LOB data. Modify the ArcSDE DBTUNE XML_DOC_STORAGE and XML_DOC_VAL_STORAGE parameters to use the new table space and create a new XML column using those parameters. Finally, republish your XML documents to the new XML column.

Ideally, all out of line LOB data blocks would be stored in the buffer cache to optimize search performance. You can use the xml_lob_cache_size script in the %SDEHOME%\sdeexe\tools directory to estimate the cache size required to store all the LOB blocks. Before running this script, change the block_size variable at the beginning of the script to the actual block size for the table space storing the LOB data. Depending on available physical memory, you could adjust the buffer cache to ensure that as many LOB data blocks will be stored in the buffer cache as possible.

See the Oracle documentation for more information about the table space db_block_size parameter and the buffer cache.

Set the optimizer parameters for Oracle9*i* and Oracle8*i*

For Oracle9*i* and Oracle8*i,* a few optimizer parameters should be set in the database for searches to perform well. Setting these parameters will not adversely affect other client software that accesses your ArcSDE database. You do not need to set these parameters for Oracle10*g*.

Oracle9i

For Oracle9*i*, the following parameter must be added to your Oracle server’s init.ora file:

```
OPTIMIZER_MODE=CHOOSE
```

Only the optimizer mode must be set; the other parameters used with Oracle8*i* aren’t required.

Oracle8i

For Oracle8*i*, the following parameters must be added to your Oracle server’s init.ora file:

```
OPTIMIZER_INDEX_CACHING = 90
OPTIMIZER_INDEX_COST_ADJ = 10
OPTIMIZER_MODE=CHOOSE
```

Refer to the Oracle8i documentation for a complete discussion of these parameters. In general, setting these parameters as specified directs the Oracle8*i* optimizer to favor index scans and nested loop execution plans over a full table scan or a sort merge.

Set the language of the lexer used to index text

If your XML column will store documents written in a language other than English, you may need to set the XML_IDX_INDEX_TEXT parameter in the SDE_dbtune table. With Oracle's text component, the lexer is used to split the original text into the words that will be indexed. If the content of the XML documents is written in English and you are using an English version of Oracle, you do not need to set this parameter for the text to be indexed correctly. Similarly, if you are using a localized version of Oracle such as a Japanese version of Oracle and the content of your documents is written in Japanese, the default lexer will be set to Japanese and you should not have to change the value of the XML_IDX_INDEX_TEXT parameter.

However, if you are using an international English version of Oracle and the language of your documents is not English, you will need to use the XML_IDX_INDEX_TEXT parameter to set Oracle’s text component to use the appropriate lexer for your language. First, the user who owns the XML column must create a lexer preference in Oracle that specifies which lexer to use. For example, if your XML documents are written in Traditional Chinese, you would use the lexer "Chinese_lexer". After logging in to SQL*Plus as the user who owns the XML column, you must create a lexer preference by issuing the create preference command. For example, for Traditional Chinese:

```
sql>exec ctx_ddl.create_preference(‘myChLex’,‘chinese_lexer’);
```

Then, in the SDE_dbtune table you would reference your lexer preferences by setting the XML_IDX_INDEX_TEXT parameter to "lexer myChLex". If this DBTUNE parameter is not set properly, the text in the XML documents will not be indexed properly and you will have trouble finding documents with a search.

See the Oracle documentation for more information about the lexers used by the text components.

Modifying the ArcSDE MAXBLOBSIZE parameter for ArcIMS Metadata Services

XML documents that contain metadata describing GIS resources may contain an image or a reference to an image that illustrates the contents of the resource. When these XML documents are published to an ArcIMS Metadata Service, a copy of the image is stored in a BLOB column in the feature class associated with the service.

For metadata XML documents created using ArcCatalog, a small thumbnail image may be embedded within the XML document. There should be no trouble publishing XML documents containing these thumbnail images to a Metadata Service with the default MAXBLOBSIZE setting for ArcSDE.

When a metadata XML document is published to a Metadata Service using ArcCatalog and a thumbnail is not embedded within the document, the document’s content is checked to see if there is a reference to an external image file. Specifically, the FGDC metadata element Browse Graphic File Name is checked. If a file path or HTTP address is found that references a JPEG image and ArcCatalog can access the image, the file will be loaded into the Metadata Service as the document’s thumbnail. ArcCatalog will only attempt to access the first JPEG image referenced in the metadata.

External image files referenced by a metadata document may be large. If the file is larger than approximately one megabyte in size, publishing the document to the Metadata Service may fail. You must adjust the default value of the MAXBLOBSIZE parameter in ArcSDE if you want to support publishing larger images as thumbnails.

The MAXBLOBSIZE parameter can be found in the giomgr.defs file located in the etc directory of your ArcSDE installation. For information on how to adjust this parameter, see the section on managing BLOB data in Chapter 3 of *Managing ArcSDE Application Servers*. This document can be found in your %SDEHOME% directory. You may also want to refer to Appendix D, “ArcSDE initialization parameters,” in the same guide.

Text index synchronization and optimization

An ArcSDE XML column can have two text indexes to support different types of searches: a text index associated with the XML document table, which supports full text searches against all content in all XML documents, or a text index associated with the XML index table that supports searches against specific XPaths in the XML documents. For example, you might search only the content of an XML element containing scale information to find GIS resources that are appropriate for your application.

Both text indexes associated with an ArcSDE XML column are CONTEXT indexes, which are designed for indexing articles, reports, and similar documents such as metadata describing GIS resources. It is important to manage these indexes appropriately as new documents are published to the XML column to maintain good search performance. This involves synchronizing and optimizing the text indexes.

Synchronization

Text indexes are nontransactional in nature. As new documents are published, rows are added to the XML document table and the XML index table as appropriate. Content in the new rows will not be added automatically to the text indexes. The text indexes must be explicitly synchronized to index the new content; this is accomplished using the ctx_ddl.sync_index function. Until synchronization occurs, no documents published since the last synchronization will be found by full text or XPath searches.

For XML columns associated with an ArcIMS Metadata Service, both text indexes associated with the ArcSDE XML column will be synchronized when the aimsmetaindx command is run after publishing new docs. To learn more about the aimsmetaindx command, see Chapter 3, “Creating Metadata Services” in *Creating and Using Metadata Services*, a document that is included with your installation of ArcIMS.

Index fragmentation

CONTEXT indexes are inverted, meaning each indexed word is associated with the list of documents containing that word. When CONTEXT text indexes are frequently synchronized, they can become fragmented, which eventually affects the performance of searches using indexes. For example, after publishing a number of XML documents and synchronizing the index for the first time, the word “river” might have an entry as follows:

```
RIVER DOC1 DOC3 DOC5
```

Each time the index is synchronized after adding new documents, rows are added for words that were indexed in the new documents. Suppose the next three times the index was synchronized, one document was found to contain the word “river” each time. The index would become fragmented, like this

```
RIVER DOC1 DOC3 DOC5
RIVER DOC9
RIVER DOC11
RIVER DOC15
```

Synchronizing less frequently will result in longer document lists, reducing the number of rows in the index and, hence, reducing fragmentation.

As an example, ArcIMS Metadata Services support a configuration option for which the text indexes will be synchronized each time an XML document is published; this is accomplished by setting the index_words option to “automatic” in the service’s ArcXML configuration file. This option will produce highly fragmented text indexes for the text indexes associated with the service’s XML column. It would be preferable to set the index_words option to “manual” and synchronize the index at a more appropriate interval such as every night. For example, this could be accomplished by scheduling a cron job to run the ArcIMS aimsmetaindx command. See the Oracle documentation for information about scheduling processes such as updating text indexes.

Optimization

Regardless of how often the text indexes are synchronized, some fragmentation of the text index is unavoidable because of how the index is updated. You should periodically optimize CONTEXT text indexes to minimize fragmentation and index size and therefore improve search performance.

For XML columns associated with large Metadata Services that provide a central repository of regional information, you might optimize the index each night after synchronizing the indexes to maintain the best possible search performance. For XML columns containing a smaller number of documents, you may want to optimize the index weekly or monthly depending on how often new documents are published and the text indexes are synchronized.

Note: Text indexes for all XML columns should be optimized when the number of XML documents published to the XML column has increased by approximately 20 percent since the last time optimization occurred.

An XML column’s text indexes can be optimized using the sdexml command with the optimize option. For example,

```
sdexml -o optimize -l aTable,XML -i 5151 -s oracleServer -u aUser -p
aPassword
```

For XML columns associated with an ArcIMS Metadata Service, text indexes associated with the ArcSDE XML column can be optimized by running the aimsmetaindx command with the indexAll option. To learn more about the aimsmetaindx command, see Chapter 3, “Creating Metadata Services,” of *Creating and Using Metadata Services*.

With either optimization method, you can continue to search the contents of the XML column while optimization takes place.

For more information about Oracle text indexes, refer to the *Oracle Text Application Developers Guide*.

Updating database statistics for a Metadata Service's tables

Over time, as people publish documents to, update documents in, and delete documents from a Metadata Service, the statistics in the database for the service's objects will become stale. When this happens, performance may decline for searches that use criteria that are evaluated against data in the Metadata Service's tables rather than the XML column's indexes. These criteria include the UPDATED, AREA, DOCUMENTINFO, and SUBSET ArcXML requests.

To keep your Metadata Service operating at peak performance, you should periodically update the statistics for the business table associated with the Metadata Service's feature class and the relationships table. You should do this whenever approximately 20 percent of the records in the Metadata Service have changed as a result of adding, updating, or deleting documents.

Update statistics using the ArcSDE sdetable command from the operating system's command line. For Oracle you would execute this command as follows:

```
sdetable -o update_dbms_stats -t <table_name> -m compute -i <service> -s
<server_name> -u <DB_User_name> -p <DB_User_password>
```

Run the above command twice, once for the feature class and once for the relationships table. For the feature class, set the <table_name> argument to the table name prefix defined in the Metadata Service's ArcXML configuration file such as "imsmetadata." For the relationships table, set the <table_name> argument to <prefix>R, such as "imsmetadatar" if the table name prefix is "imsmetadata." Modify the remaining arguments to provide the appropriate ArcSDE connection information.

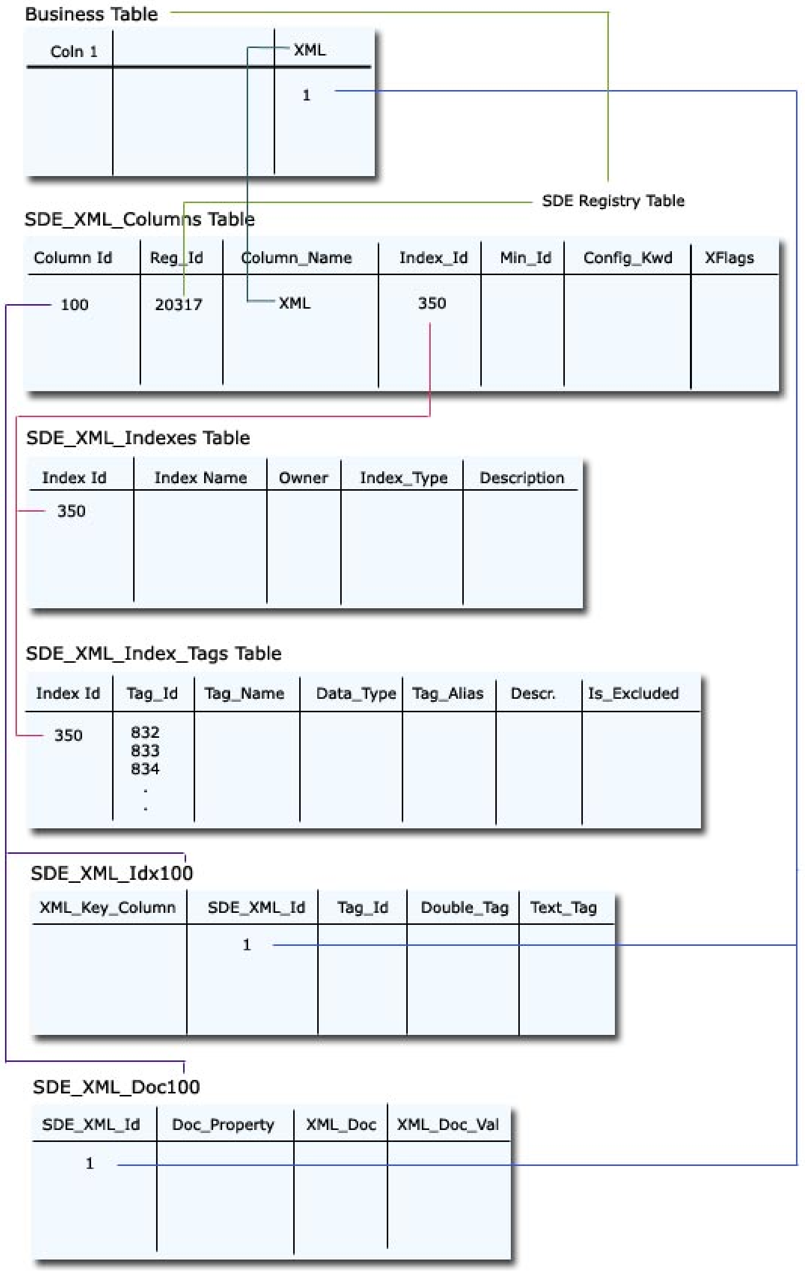
For more information about the sdetable command, see the Administrative Command References located in the documentation folder in %SDEHOME%, or

from *support.esri.com,* click Software > ArcSDE > ArcSDE Developer Help > Administrative Command References.

ArcSDE XML columns database schema

Three ArcSDE system tables are used to manage XML columns: XML columns, XML index tags, and XML indexes. These tables are owned by the SDE user. ArcSDE also creates two additional tables for each XML column that are used to store and index XML documents: the XML document and XML XPath index tables. These tables are owned by the user who owns the business table containing the XML column.

An example of the ArcSDE XML columns database schema is shown on the next page.



When an XML column is added to an object, the column is recorded in ArcSDE XML columns table. Information in this table is used to create the XML document and index tables and to record information about their text indexes in the ArcSDE XML indexes and index tags tables.

XML columns table (SDE_xml_columns)

When you add an XML column to a business table, a row is added to the XML columns table. This table occurs once in each ArcSDE database.

The table's columns are described below. The ArcSDE data type follows the column's name.

Column_id (SE_INT32)—The XML column's identifier and the table's primary key. This value is assigned by ArcSDE at the time the XML column is created.

Column_name (SE_STRING)—The name of the XML column in the business table.

Config_keyword (SE_STRING)—The DBTUNE configuration keyword containing parameters that determine how the XML document and the XML XPath index tables and text indexes created on those tables are defined in the database. For more information on DBTUNE keywords and their parameters, see the ArcSDE configuration and Metadata Service DBTUNE sections in Appendix B of *Creating and Using Metadata Services*. Also, review the chapter "Configuring DBTUNE storage parameters" in this document.

Index_id (SE_INT32)—The identifier of the XPath index associated with the XML column, if one exists. A foreign key to the XML indexes table.

Minimum_id (SE_INT32)—The value of the initial number used in the business table's XML column to identify individual XML documents.

Registration_id (SE_INT32)—The identifier of the business table containing the XML column. A foreign key to the ArcSDE table registry.

Xflags (SE_INT32)—A value indicating whether the original documents in the XML document table are stored compressed or uncompressed. By default, documents are compressed; compressed documents provide better performance.

XML indexes table (SDE_xml_indexes)

This table occurs once in each ArcSDE database. It contains one row for each XML column that has an XPath index.

Description (SE_STRING)—Text identifying the XPath index. If an index definition file was used to create the index, the index's description may be specified at the top of the file.

Index_id (SE_INT32)—The identifier of the XPath index. The table's primary key.

Index_name (SE_STRING)—The name of the XPath index. For XPath indexes associated with an ArcIMS Metadata Service, the name will be "ims_xml#" where # is the identifier of the XML column in the Metadata Service's business table.

Index_type (SE_INT32)—A value indicating the type of XPath index. With ArcSDE 9.1 the value will be two for the index type SE_XML_INDEX_DEFINITION and one for the index type SE_XML_INDEX_TEMPLATE. For XPath indexes associated with an ArcIMS Metadata Service, only the index type SE_XML_INDEX_DEFINITION is supported.

Owner (SE_STRING)—The database user who owns the XML column. For ArcIMS Metadata Services, this is the user specified in the service's ArcXML configuration file.

XML index tags table (SDE_xml_index_tags)

An XML column may optionally have an XPath index, which lets people search the content of a specific XML element or attribute in each document. The definition of which elements and attributes are included in or excluded from each XPath index is recorded in this table.

This table occurs once in each ArcSDE database. It contains one row for each XPath associated with an XML column's XPath index.

Data_type (SE_INT32)—A value indicating whether the XML element or attribute will be indexed as a string or a number. A one indicates the content of the tag will be indexed as text; a two indicates the content will be indexed as a number.

Description (SE_STRING)—Text identifying the content that should be contained in the XML element or attribute.

Index_id (SE_INT32)—The identifier of the XPath index associated with an XML column, if one exists. A foreign key to the XML indexes table.

Is_excluded (SE_INT32)—A value indicating whether the XML element is included in or excluded from the XPath index. A zero indicates the XPath is included; a one indicates the XPath is excluded.

Tag_alias (SE_INT32)—A number that may be used to identify an XPath. For example, the Z39.50 communication protocol uses numeric codes to refer to content that may be searched. This column is not used by the ArcIMS Z39.50 Connector.

Tag_id (SE_INT32)—The identifier of an XPath or tag.

Tag_name (SE_STRING)—An absolute XPath identifying an XML element or attribute that may occur in an XML document. For example, /metadata/mdDateSt identifies an XML element, and /metadata/dataIdInfo/tpCat/TopicCatCd/@value identifies an XML attribute. These XPaths must not contain asterisks (*) to refer to a group of XML elements or attributes—each element or attribute is matched exactly using the XPaths specified in this table.

XML document table (sde_xml_doc<column_id>)

The ArcSDE database contains one of these tables for each XML column. The number in the table name is the XML column's identifier. This table contains one row for each XML document stored in the XML column.

Doc_property (SE_INT32)—A value indicating whether or not any conflicts were found when adding the content of an XML document to the XPath index. A value of one indicates a conflict was found, for example, when an element is supposed to be indexed numerically but the document contains a string in that element instead. A NULL value indicates there were no problems indexing the document.

Sde_time_stamp* (SE_INT32)—For SQL Server only. This column contains a time stamp that is used to support incremental updates to the text index.

Sde_xml_id (SE_INT32)—The identifier for an XML document stored in the XML column. The primary key for the table.

Xml_doc (SE_BLOB)—The XML document.

Xml_doc_val (SE_BLOB)—The content of the entire XML document, with all XML tags removed. A text index is built on this column by default; this index is used to respond to full text queries. For ArcIMS Metadata Services, this index is used to respond to FULLTEXT requests.

XML XPath index table (sde_xml_idx<column_id>)

The ArcSDE database contains one of these tables for each XML column that has an XPath index. The number in the table name is the XML column's identifier. This table contains one row for each XML element or attribute in each document that is included in the XPath index.

Double_tag (SE_FLOAT64)—For XPaths that are indexed numerically, the number contained in the element or attribute. For XPaths that are indexed as text this column will contain a NULL value. For ArcIMS Metadata Services, this column is used to respond to TAGVALUE requests.

Sde_time_stamp* (SE_INT32)—For SQL Server only. This column contains a time stamp that is used to support incremental updates to the text index.

Sde_xml_id (SE_INT32)—The XML document's identifier and a foreign key to the XML document table.

Tag_id (SE_INT32)—The identifier for the XML element or attribute that is indexed. This number corresponds to the value in the tag_id column in the XML index tags table.

Text_tag (SE_CLOB)—For XML elements or attributes that are indexed as string, the text contained in the element or attribute. For XPaths that are indexed numerically this column will contain a NULL value. A text index may optionally be built on this column; this index is used to respond to XPath queries. For ArcIMS Metadata Services, this index is used to respond to TAGTEXT requests.

Xml_key_column (SE_INT32)—The primary key for the table.

Index