ArcGIS® 9

ArcSDE® Configuration and Tuning Guide for Informix®

Contents

CHAPTER 1

Getting started

Creating and populating a geodatabase is arguably a simple process, especially if you use ESRI® 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. This book provides instruction for configuring the physical storage parameters of your data in the database management system (DBMS). This book also provides some important guidelines for configuring and tuning the IBM Informix® instance itself.

Tuning and configuring the Informix instance

Building an efficient geodatabase involves properly tuning and configuring the Informix instance and proper arrangement and management of the database's tables and indexes. Chapter 2, 'Essential Informix 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 Informix database.
- Create the tablespaces that will store your tables and indexes.
- Tune the Informix 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 DBTUNE table to define physical data storage parameters of ArcSDE tables and indexes. The storage parameters are grouped into configuration keywords. You assign configuration keywords to your data objects (tables and indexes) when you create them from an ArcSDE client program.

Prior to ArcSDE 9, configuration keywords were stored in a dbtune.sde file maintained under the ArcSDE etc directory. The dbtune.sde file is still used by ArcSDE 9 as the initial source of storage parameters. When the ArcSDE 9 file sdesetupinfx command executes, the configuration parameters are read from the dbtune.sde file and written into the DBTUNE table.

It should also be noted that ArcSDE 9 has simplified the storage parameters. Rather than matching each Informix storage parameter with an ArcSDE storage parameter, the ArcSDE storage parameters have evolved into configuration strings and represent the entire storage configuration for a table or index. Prior to ArcSDE 9 storage parameters have been automatically converted to the new simpler ArcSDE 9 storage parameters. The ArcSDE storage parameter holds all the Informix storage parameters of an Informix CREATE TABLE or CREATE INDEX statement.

The sdedbtune command has been introduced at ArcSDE 9 to provide the ArcSDE administrator with an easy way to maintain the DBTUNE table. The sdedbtune command exports and imports the records of the DBTUNE table to a file in the ArcSDE etc directory.

The ArcSDE 9 installation creates the DBTUNE table. If the dbtune.sde file is absent or empty, sdesetupinfx creates the DBTUNE table and populates it with default configuration keywords representing the minimum ArcSDE configuration.

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

Creating spatial data in an Informix 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 Informix database. With these tools you can convert ESRI coverages and shapefiles into ArcSDE feature classes. You can also import an ArcSDE export file containing the data of a business table, feature class, or raster column.

Multiversioned ArcSDE data can be edited directly with either ArcCatalog or ArcMap.

An alternative approach to creating spatial data in an Informix 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 Informix database.

Note: ArcSDE XML columns are not supported with Informix databases. Therefore, ArcIMS Metadata Services are not supported with Informix databases.

Connecting to Informix

ArcSDE clients connect to the ArcSDE service. Under the ArcSDE three-tiered architecture, the ArcSDE client connects to the ArcSDE service, and the ArcSDE service spawns a dedicated *gsrvr* process that connects to the Informix instance. The gsrvr process brokers the spatial data between the ArcSDE client and the Informix instance. The ArcSDE service and the gsrvr processes typically reside on the Informix host machine, while ArcSDE clients are typically on remote machines.

National language support

If you intend to support a database that does not use the Informix default 7-bit United States ASCII English (US7ASCII) character set, you will have to take a few extra steps in creating the Informix 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 Informix 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 it. 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 Informix backup. In addition, suggested Informix reference materials are listed for further reading.

CHAPTER 2

Essential Informix configuring and tuning

The performance of an ArcSDE application depends to some extent on how well you configure and tune Informix. This chapter provides basic guidelines for tuning an Informix database for use with an ArcSDE application. It assumes that you have a basic understanding of the Informix data structures, such as dbspaces, sbspaces, tables, and indexes, and that you are proficient with Structured Query Language (SQL). We encourage you to refer to Informix's extensive documentation, in particular *Informix Performance Guide for Informix Dynamic Server 2000* and *Informix Administrator's Guide for Informix Dynamic Server 2000* for your appropriate Informix release.

How much time should you spend tuning?

The appreciable difference between a well-tuned database and one that is not 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 resources among users by configuring the components of a database to minimize contention and remove bottlenecks. The more people you have accessing your databases, the more effort is required to provide access to a finite resource.

A well-tuned Informix database makes optimum use of available central processing unit (CPU) 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, where it is impractical to continue tuning; instead, they continue to monitor the server and address performance issues as they arise.

Windows 2000 systems

Updating the onconfig file

Informix maintains its configuration parameters in the onconfig file located in the %INFORMIXDIR%\etc directory on Windows 2000. The parameters of this file control the server's memory use, the size and number of log files, temporary space, the location of the error logs, and much more. The onconfig file is read whenever the Informix server is started. So changes to the parameter require that you restart the server.

Naming the onconfig file

The standard onconfig file, onconfig.std, contains the default settings of the Informix parameters. Do not edit this file; instead, preserve it as a record of the default settings.

On Windows 2000, the Informix Dynamic Server (IDS) installation process automatically copies the onconfig.std file to the 'Onconfig' file.

For the remainder of this document, when the onconfig file is mentioned, we are referring to the %INFORMIXDIR%\etc\Onconfig on Windows 2000. On Windows 2000, the installation also sets the system variable ONCONFIG to Onconfig. The 'Onconfig' file is also defined in the Windows registry as the Informix onconfig file. If you intend to use an onconfig file with a different name, you need to change the ONCONFIG environment in the registry and the %INFORMIXDIR%\setenv.cmd file.

Some important onconfig parameters

The following is a list of some of the more important onconfig parameters whose default values you should change to improve the performance of your Informix server when using it with ArcSDE.

BUFFERS

The BUFFERS parameter file controls the size of the regular buffers, the area of memory in which Informix stores the most recently used pages of data. The first reader reads the pages from disk, while subsequent readers read the pages from the regular buffer until it is paged out of memory. A page will be paged out of the regular buffer if it is has been unused over a period of time and the memory is needed to hold other pages that are being used.

Increase the number of data buffers to 2,000 or 25 percent of your physical RAM, whichever is greater. BUFFERS is specified in pages. If your pages are 2 kilobytes (page size can be determined with the Informix command onstat -b) and your physical RAM is 256 MB, BUFFERS would be calculated as follows:

```
BUFFERS = <physical RAM converted to kilobytes> * 25% /
          <page size in kilobytes>
        = (256 * 1024) * 0.25 / 2
        = 32768

BUFFERS 32768
```

LOGSIZE

The LOGSIZE parameter controls the default size of the logical logs. The size of the logical logs can be specified when they are created with the INFORMIX onparams utility. However, if the size is not specified, LOGSIZE is used.

Set the logical log file size to 100,000 kilobytes. When the logical logs are moved out of the rootdbs, they will be created with this size.

```
LOGSIZE 100000
```

LOG_BACKUP_MODE

The LOG_BACKUP_MODE parameter specifies the mode in which logical logs are backed up. This mode can be either continuous or manual. Continuous mode will allow you to automatically do logical log backups when required.

```
LOG_BACKUP_MODE CONT
```

LOGSMAX

The LOGSMAX parameter specifies the maximum number of logical logs that may be created. Increase the LOGSMAX parameter so that you can create new logical logs in order to move them out of the rootdbs.

Set the maximum number of logical log files to 100.

```
LOGSMAX 100
```

CLEANERS

CLEANERS specifies the page cleaner threads started by the INFORMIX instance. Page cleaner threads periodically wake up and perform background writes of batches of dirty pages held in the regular buffers to disk.

Set the number of page cleaners to 6 or the number of disks that contain frequently accessed data, whichever is higher.

```
CLEANERS 6
```

STACKSIZE

STACKSIZE specifies the amount of stack allocated to the INFORMIX instance. Although for most applications Informix recommends that this parameter be left at its default value of 32 (kilobytes), for ArcSDE it is important to increase the size of this parameter to 64 (kilobytes) in support of the Informix Spatial DataBlade® user-defined datatypes (UDTs) accessed by ArcSDE.

Increase the initial stack size of each thread to 64 kilobytes. Set the STACKSIZE parameter to 64.

```
STACKSIZE 64
```

RA_PAGES

This read-ahead parameter sets the number of data and index pages that are cached in the regular buffers whenever a sequential scan of one or more tables occurs.

Set the read-ahead pages to 125.

```
RA_PAGES  125
```

RA_THRESHOLD

RA_THRESHOLD, the read-ahead threshold, specifies the number of remaining unread pages that triggers another call to read more pages from disk.

Set the number of unprocessed pages that trigger another read ahead to 85.

```
RA_THRESHOLD 85
```

DUMPDIR

The DUMPDIR parameter specifies the location of the dump directory where error log files are written in the event of an assertion failure.

Leave the dump directory set to tmp if you have adequate space there. However, you can create a tmp directory under the Informix installation directory and set DUMPDIR to that. Should an assert failure occur, the diagnostic files are one directory below the online.log file that references them.

```
DUMPDIR C:\informix\tmp
```

RESIDENT

The RESIDENT parameter specifies which portion of the INFORMIX instances shared memory can be swapped out of the operating system's shared memory. Allowing as many portions of the instance’s shared memory as possible to remain resident eliminates a large amount of I/O and context switching of the instance’s memory structures.

Setting the RESIDENT parameter to -1 keeps as many of the instance's memory structures as possible resident given the amount of physical memory and system resources available.

```
RESIDENT –1
```

MULTIPROCESSOR

The MULTIPROCESSOR parameter specifies whether the Informix Server machine has one or multiple processors to use.

Set to 0 if the Informix Server machine has only one processor and set to 1 if there are multiple processors.

System parameters that must be adjusted prior to initialization

TAPEDEV

The TAPEDEV parameter specifies the device used to back up the dbspaces. During the loading phase of your database, it is often a good idea to set this parameter to the NUL device. After the data is loaded set the parameter to the proper tape device. The rationale behind this is that the data is already backed up by the data source that you are loading it from. Therefore, if a dbspace is lost to a disk failure, the data can be restored from the original data source. Once the database is loaded, you can set it to your tape device.

```
TAPEDEV   NUL
```

LTAPEDEV

The LTAPEDEV parameter specifies the tape device to which the ONTAPE utility backs up the logical log files.

Set this to the NUL device. Once the server is up, you can set it to your tape device if you intend to archive the log files.

```
LTAPEDEV   NUL
```

NETTYPE

Set separate NETTYPE parameters to configure the poll threads for the shared memory and TCP/IP network protocols. The settings below allow 20 local connections and 200 remote connections. The configuration of the NETTYPE parameter is discussed in detail in the 'Network virtual processors' section later in this chapter. Set the NETTYPE parameters to the expected number of local and remote connections, as in the example for Windows 2000 below:

Windows/2000

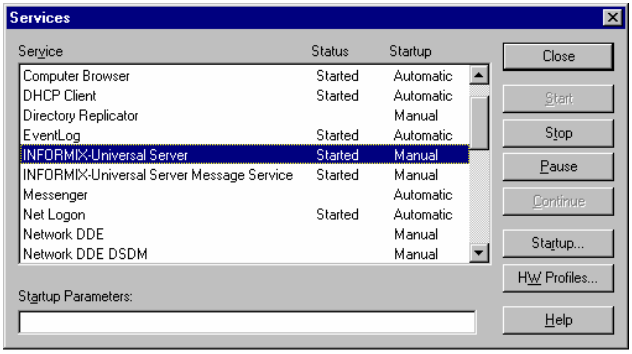
```
NETTYPE  olsoctcp,1,,NET
```

Restarting the Informix Dynamic Server

To apply the changes made to the onconfig file of the Informix kernel, you must restart the server.

Restarting the Informix service

The server is started and stopped from the Windows Service panel found on the control panel. Open the Control Panel and double-click the Services icon. Highlight the INFORMIX-Universal Server service and press the Stop button on the Services panel. Press Yes when prompted if you really want to do this and then press the Start button to restart the server. After a few seconds, the server should start up again; if it doesn't, check the %INFORMIXDIR%\online.log file to determine why the server won't start. Typographic errors in the onconfig file are the most common problems. Correct the problem and restart the server.



Tuning disk I/O contention

Disk I/O contention can prove to be one of the more difficult challenges for a Database Administrator (DBA) to overcome. Unlike memory and CPU issues that can be solved by acquiring more of these resources once all tuning procedures have been exhausted, the reduction of disk I/O contention must be solved through proper planning and administration of the file system.

Beyond the possibility of acquiring faster disk drives and controllers, the only real way to reduce disk I/O contention is to balance the I/O across the entire file system by distributing files that experience a high frequency of I/O with those that do not.

RAID systems

Redundant Arrays of Inexpensive (or Independent) Disks (RAID) boost performance by striping data into slices across multiple disks in a disk array. By spreading data across multiple disks, all disks share the burden of I/O operations, thus reducing the chance of a bottleneck occurring on one disk. RAID's performance increases as you add disks to the

array. The operating system and database will see only one volume, a logical representation of the entire disk array.

In a simple configuration, you could create a single disk array of four disks and configure one large data file within that RAID array. Your data would be striped across all four disks evenly, reducing contention. The database's transaction log should not occupy this same array. This solution proves scalable as well—additional performance benefits can be gained by adding disks to the array until performance increases begin to decline. More complex configurations would include separate disk arrays for indexes, data tables, and geometry data.

Creating the system dbspaces

In the section 'Arranging your data', which follows, you will learn how to create dbspaces to store your business tables and indexes. Before you start creating these dbspaces, however, create dbspaces to serve as temporary storage for the transitional functions of the Informix server. Logical log files, physical log file, and temporary space for sorting should occupy their own dbspace.

Depending on the available number of disks, try to spread the devices of the dbspaces across your file system. Try to keep the devices of the physical and logical logs separate. Either the physical log or the logical logs may share the same disk as the root device.

The temporary sorting devices (commonly referred to as temp devices) should be separated from all other devices, if possible. These temp devices are used heavily during the creation of the R-Tree index after data loading.

Therefore, it is a good rule of thumb to start with at least 200 MB of temporary storage spread across at least two sorting devices to handle the loading of large datasets and their associated R-Tree index building.

You may need to monitor the temporary space usage during the loading of large datasets to make sure Informix does not run out and produce an error. If this happened, it would typically leave the ArcSDE table in "load-only mode".

Device Files

To create a device file on a Windows 2000 platform, use Explore to locate the directory in which you want to create the device file and create a new text file. Rename the text file to your device filename.

Create the device for the physical log.

```
phydbs.000
```

Create three devices for the logical logs.

```
log1dbs.000
log2dbs.000
```

log3dbs.000

Create two devices for sorting.

temp1dbs.000
temp2dbs.000

For example, to create the physical log device file example above, right mouse click on My Computer and select Explore from the list. Locate the proper directory. If the directory does not exist, create it by selecting File>New>Folder from Explore. Rename the folder with a right mouse click. While the cursor is on the folder, select Rename from the list and rename the folder (or directory).

Once you have moved to the correct directory, select File>New>Text Document from the Explore menu. Rename the document with a right mouse click while the cursor is on the document by selecting Rename from the list. It is common for Informix device files to have a .000 initial extension to distinguish them from other types of Windows 2000 files.

Make sure that these device files have read permissions for all and full control for the Informix administrator account.

The Informix onspaces utility manages dbspaces. Use it to create the dbspaces and assign them to the devices that you have just set up. The onspaces syntax varies slightly depending on the kind of dbspace it is operating on. However, the basic syntax for creating the system dbspaces is:

```
onspaces -c -d <dbspace_name> -p <path to device> -o <offset> \
         -s <size in kilobytes>

onspaces -c -t -d <dbspace_name> -p <path to device> -o <offset> \
         -s <size in kilobytes>
```

The -t flag is included to indicate that the dbspace will be used for sorting and other temporary activities.

When creating a dbspace on the Windows 2000 platform, only the style of the pathname changes. Start the onspaces and other Informix utilities from the INFORMIX-Universal Server Command Line Utilities. To invoke this special MS–DOS® command line entry window, press Start>Programs>INFORMIX®-Universal Server>Command Line Utilities. The MS–DOS window runs the %INFORMIXDIR%\setenv.cmd Informix system environment file. If you try to execute the Informix utilities from a regular MS–DOS command window, you will receive errors unless you set the system environment variables listed in the %INFORMIXDIR%\setenv.cmd file.

```
# Create the first dbspace for logical logs
onspaces -c -d log1dbs -p D:\informix_data\log1dbs.000 -o 0 -s 125000

# Create the second dbspace for the logical logs
onspaces -c -d log2dbs -p E:\informix_data\log2dbs.000 -o 0 -s 125000

# Create the third dbspace for the logical logs
onspaces -c -d log3dbs -p E:\informix_data\log3dbs.000 -o 0 -s 125000
```

```
# Create the dbspace for the phydbs
onspaces -c -d phydbs -p C:\informix_data\phydbs.000 -o 0 -s 10000

# Create the first temporary dbspace
onspaces -c -t -d temp1dbs -p F:\informix_data\temp1dbs.000 -o 0 -s 150000

# Create the second temporary dbspace
onspaces -c -t -d temp2dbs -p G:\informix_data\temp2dbs.000 -o 0 -s 150000
```

Moving the physical log out of the root dbspace

Moving the physical log out of the root dbspace reduces the I/O contention. Simply change the PHYDBS parameter in your onconfig file to the dbspace you have just created for physical logging. In our example the PHYSDBS parameter would be set to phydbs.

```
PHYSDBS phydbs
```

Increase the size of the PHYSFILE to use the space allocated to the physical log's dbspace. In the example, the phydbs dbspace is 10,000 kilobytes, allowing you to increase the PHYSFILE to 9000. It cannot be increased to the size of the dbspace because Informix uses a certain amount of space for overhead.

```
PHYSFILE 9000
```

Shut down and restart the Informix server to use the phydbs dbspace for physical logging.

Windows 2000 users stop and start the server from the Services panel (see 'Restarting the Informix Dynamic Server' above).

Examine the end of the online.log file. An entry should exist stating that the physical logging has been changed to the dbspace you specified.

Moving the logical logs out of the root dbspace

For the same reason you moved the physical log from the root dbspace, you must do the same for the logical logs. First, make sure the LOGSMAX parameter in the onconfig file is set high enough.

By default, the installation creates ten logs in the root dbspace on the Windows 2000 platform. To add 3 log files on a Windows 2000 platform, set LOGSMAX to at least 100.

Make sure you create enough logical logs to handle your longest transaction. Typically, long transactions occur when you create or delete a large dataset or when you compress a geodatabase. You must checkpoint your logical logs by backing them up before you reach the long transaction high watermark percentage defined by the LTXHWM parameter in your Informix onconfig file. You should not change either the LTXHWM or LTXEHWM without the consent of an Informix technical support expert who is familiar with the behavior of the Informix Spatial DataBlade. If a transaction fails to complete and is rolled back because it reaches the long transaction high watermark, then you do not have enough logical logs.

To create the new logical logs, first set the server in quiescent mode by issuing the onmode command with -s flag Remember to execute all Informix utilities from the INFORMIX-Dynamic Server command line on the Windows 2000 platform, rather than the normal DOS command window.

```
C:\Informix> onmode -s
```

To add logical log files to each of the dbspaces created for them, use the Informix onparams utility. When you add the log files, make sure you alternate between at least two dbspaces. This ensures that while one log file is being flushed from one disk drive another can be written to another disk drive.

```
C:\Informix> onparams -a -d log1dbs
C:\Informix> onparams -a -d log2dbs
C:\Informix> onparams -a -d log3dbs
```

Activate the new logical logs by performing a zero-level archive with the Informix ontape utility.

```
C:\Informix> ontape -s
```

Now you can delete the original 10 logical logs that reside on the root dbspace. First, you must determine if one of the first 10 logical logs is the current one. Use the onstat -l command to generate a list of the logical logs.

```
C:\Informix> onstat -l
```

The logical log is current if its flags column contains a C. Find this logical log and note its number. If the number is between one and 10 on a Windows 2000 platform, you must advance the log with the onmode -l utility.

```
C:\Informix> onmode -l
```

Repeat the onstat -l followed by the onmode -l utility until a logical log with a number greater than 10 for Windows 2000, becomes current.

Then use the onparams -d utility to drop the logical logs in the root dbspace.

```
C:\Informix> onparams -d -l 1  -y
C:\Informix> onparams -d -l 2  -y
C:\Informix> onparams -d -l 3  -y
C:\Informix> onparams -d -l 4  -y
C:\Informix> onparams -d -l 5  -y
C:\Informix> onparams -d -l 6  -y
C:\Informix> onparams -d -l 7  -y
C:\Informix> onparams -d -l 8  -y
C:\Informix> onparams -d -l 9  -y
C:\Informix> onparams -d -l 10 -y
```

Use the ontape -s utility command to archive the change. The output of the onstat -l utility should list only those log files that were added to the log1dbs, log2dbs and log3dbs spaces. The output of the onstat -l output from a Windows 2000 platform should begin at 11.

Put the server back to online mode with the onmode -m utility.

```
C:\Informix> onmode -m
```

Setting up the temporary dbspace

By default, Informix uses the root dbspace when it needs temporary space for sorting. The creation of a large index, such as R-tree index can fill the root dbspace, resulting in a server crash. Or, in the case of loading data using the ArcCatalog product, the creation of the rtree index (the last step in loading data) will fail, and the layer loaded will remain in load-only mode.

It is better then to control the location of the temporary space by using separate dbspaces and adding chunks of space as necessary. Set the DBSPACETEMP parameter in the onconfig file to the temporary dbspaces created earlier. Remember to use at least 2 dbspaces totaling 300 MB as a starting point. Then add chunks accordingly, making sure that these chunks span different disks whenever possible.

```
DBSPACETEMP temp1dbs,temp2dbs
```

Restart the Informix server to set the temporary space in the server. On the Windows 2000 platform, the Informix server is restarted from the Services panel (see 'Restarting the Informix Dynamic Server' above).

Examine the online.log file to ensure that the temporary space is set. You should see an entry stating that the temporary files have been relocated from the root dbspace to the dbspaces you assigned to the DBSPACETEMP parameter.

Creating the default smart large object dbspace

The Spatial DataBlade module writes the compressed geometry to the smart large object whenever it is larger than 929 bytes. For this reason a default smart large object space or sbspace must exist.

On the Windows 2000 platform, the default sbspace and syssbspace (found in the onconfig file) are created during the installation of the server. You can add additional chunks of sbspace to this default sbspace or create additional sbspaces to access through the %SDEHOME%\etc\dbtune.sde file.

Allocating enough metadata within a smart large object sbspace

Make sure to also include enough space when creating both the sbspace and syssbspace for system metadata. Informix automatically creates the system metadata when you create a smart large object sbspace; however, it is usually a small percentage of the total sbspace space.

If the smart large object sbspace uses all the space allocated to the metadata, Informix returns an "out of smart large object dbspace" error after trying to store data even though plenty of smart large object sbspace exists.

Large datasets can require large amounts of smart large object metadata sbspace. Define the amount of smart large object sbspace to allocate to the metadata with the -Ms option of the following onspaces command when you create the sbspace.

```
C:\Informix> onspaces -c -S sblobdbs -g 1 -p
d:\ifmxdata\mailia\sblobdbs.000 -o 0 -s 350000 -Ms 50000
```

In this example, Informix allocates 50 MB of the total 350 MB of smart large object sbspace to metadata in the smart large object dbspace sblobdbs. You can also allocate space to the metadata when you add a chunk to the sbspace:

```
C:\Informix> onspaces -a sblobdbs -p d:\ifmxdata\mailia\sblob1dbs.000 -o 0
-s 350000 -Ms 50000
```

You can allocate all of the space to metadata when you add a chunk to smart large object sbspaces by specifying the same values for the -Ms and -s flags:

```
C:\Informix> onspaces -a sblobdbs -p d:\ifmxdata\mailia\sblob1dbs.000 -o 0
-s 350000 -Ms 350000
```

The metadata portion of a smart large object can be monitored with the "dbstat –d" command. Check the metadata available space for each smart large object sbspace.

Using smart large object sbspaces

In order to avoid the possibility of running out of default smart large object sbspace to hold both spatial column data or annotation column data, create separate smart large object sbspaces. Keep the default sbspace and syssbspace small for Informix system use only. You should make the sbspace and syssbspace separate smart large object sbspaces, but it is not necessary.

Smart large object sbspaces can be assigned to spatial columns of annotation columns using the DBTUNE table S_STORAGE parameter. For more information on the S_STORAGE parameter, see chapter 3, 'Configuring DBTUNE storage parameters'.

Smart large object sbpace at ArcSDE 9

At Spatial Database Engine™ (SDE®) 3.0.2.2 for Informix, the SDE large binary datatype SE_BLOB_TYPE was stored as the Informix datatype BYTE.

For ArcSDE 9 for Informix, the ArcSDE large binary datatype is stored as the Informix BLOB data type.

Since BLOB is stored in the smart large object BLOB spaces, you need to specify the default smart large object BLOB space in the ONCONFIG file and make sure it is large enough to hold your large binary data. Make sure you allocate enough space for the sbspace metadata.

The preferred alternative to using the default smart large object sbspace is to create separate smart large object sbspaces and control the placement of spatial columns and

annotation columns into these smart BLOBs using the storage parameters of the DBTUNE table. See Chapter 3, 'Configuring DBTUNE storage parameters', for details.

Arranging your data

Proper arrangement of tables and indexes on the file system will help to minimize disk I/O bottlenecks. Placement of these data objects requires that you estimate their size and create the dbspaces they will be stored in. You add the dbspaces names along with a list of other data object storage parameters to a DBTUNE table configuration keyword. To learn more about the storage parameters of the DBTUNE table, see chapter 3, 'Configuring DBTUNE storage parameters'. The ArcSDE server uses the parameters when it creates the data objects with the ArcSDE administration commands.

Creating the dbspaces and sbspaces

A dbspace is a logical unit of storage that Informix uses to store tables and indexes. An sbspace is a logical unit of storage designed specifically to store smart large objects. Both are created and maintained by the onspaces command. Both may have one or more physical units of storage assigned to them. The physical units of storage are called chunks. Chunks may be either raw devices or cooked files.

Raw devices are not recommended on some platforms because the advantage is negligible from an ease-of-use standpoint. Consult the *INFORMIX-Dynamic Server Administrator's Guide* for advice on when to use a raw device or a cooked file.

The size and placement of dbspaces and sbspaces depend on the tables and indexes stored within them. Here are some basic guidelines to help determine the size and placement of your tables, indexes, and smart large objects. Given the number and size of the disk drives available on your system, you may not be able to follow these guidelines to the letter, but follow them as best you can, keeping in mind that the goal is to minimize disk I/O contention.

Separate indexes onto a different disk drive from the tables they index

If the Informix optimizer determines that an index will speed up the execution of a query, it will read pages of the index into memory, search the pages for a match, and read matching table records from disk into memory. Storing the index and table on the same disk forces the disk head to unproductively travel back and forth between the index and the table. Arranging the tables and indexes on separate disks allows multiple disk heads to simultaneously read from the index and the table reducing disk head travel and seek time.

Separate smart large objects from their associated spatial tables

Spatial data too large to be stored inline with other table data is written to the designated smart large object. As with indexes, smart large object sbspaces should be stored on a disk separate from both the table and the indexes.

Place high-use tables in the middle disk drive partitions to minimize disk head movement

Placing high-use tables in the middle partitions of a disk drive reduces disk head travel. Based on the law of averages, arranging data so that the disk head spends most of its time in the middle partition reduces travel. Consult your operating system configuration manual for directions on partitioning your disk drives. Allocate a single chunk to the partition and assign the dbspace of the high-use tables to it.

Separate large high-use tables on to different disk drives

Balance disk I/O by spreading large high-use tables throughout the file system. Discuss the application model with the designers to determine which tables will be accessed most. Arrange these tables on separate disks to ensure equal employment of disk heads and controllers.

Group smaller tables together into dbspaces by usage

Creating a separate dbspace for each table in your database is unrealistic. Each dbspace has an associated overhead cost, and it's cumbersome to manage a large number of dbspaces. Group smaller tables together into a single dbspace. You should also group the related indexes into another dbspace so they may be placed on a separate disk drive.

Grouping the smaller tables by usage into separate dbspaces allows you to place the high-use smaller tables into the middle partitions.

Optimize extent sizes

Estimating the size of your tables and indexes allows you to allocate the initial extent to contain the entire data object. For data objects grouped together into the same dbspace, this prevents their extents from becoming interleaved. Interleaved extents can reduce performance if the disk head has to seek over the extents of other tables.

Assign individual dbspaces to large tables

Large tables should have their own dbspaces. This allows you to move these tables easily throughout the file system. Some tables may be so large that the dbspace assigned to them requires more than one chunk. If so, it's a good idea to separate the chunks onto different disk drives and separate controllers if possible. Doing so allows multiple access to data of the same table and reduces overall seek time.

Using onspaces to create dbspaces and sbspaces

The Informix onspaces command creates and maintains dbspaces and sbspaces. Run the onspaces command as the informix user.

Dbspaces

Dbspaces are created with the onspaces command using the following basic syntax:

```
onspaces -c -d <dbspace name> -p <pathname> -o <offset> \
         -s <size in kilobytes>
```

The dbspace name must be unique within the database. The pathname specifies the location of either a raw device or a cooked file. If it is a cooked file, the file must exist, and the informix user and group must have read and write permissions to the file. On Windows 2000 platforms, use Explore to create an empty text file in the appropriate directory.

In this example, a cooked file d:\Ifmxdata\mailia\roadsdbs1.000 was created and its permissions are set to read and write access for the informix user and group. The onspaces command creates the roadsdbs dbspace and allocates 50,000 kilobytes to the d:\Ifmxdata\mailia\roadsdbs1.000 for its initial chunk.

```
C:\Informix> onspaces -c -d roadsdbs -p d:\Ifmxdata\mailia\roadsdbs1.000 -
o 0 -s 50000
```

Additional chunks may be added to a dbspace with

```
C:\Informix> onspaces -a <dbspace name> -p <pathname> -o <offset> -s <size
in kilobytes>
```

In this example, the 50,000 KB chunk d:\Ifmxdata\mailia\roadsdbs2.000 is added to the dbspace roadsdbs2.

```
C:\Informix> onspaces -a roadsdbs -p d:\Ifmxdata\mailia\roadsdbs2.000 -o 0
-s 50000
```

Sbspaces

Smart large object spaces are created in the same fashion as dbspaces. The -S flag directs the onspaces command to create a smart large object space instead of a regular dbspace. The basic syntax for creating a smart large object space is:

```
C:\Informix> onspaces -c -S <sbspace name> -p <pathname> -o <offset> -s
<size in kilobytes>
```

In this example, the sbspace roadsblob is created with the 10,000 KB initial chunk d:\Ifmxdata\mailia\roadsblobdbs1.000.

```
C:\Informix> onspaces -c -S roadsblob  -g 1 -p
d:\Ifmxdata\mailia\roadsblobdbs1.000 -o 0 -s 10000
```

You can add additional chunks to the smart large object space with the following onspaces syntax. You will notice that the -S flag is not required when adding a chunk.

```
C:\Informix> onspaces -a <sbspace name> -p <pathname> -o <offset> -s <size
in kilobytes>
```

In this example, the d:\Ifmxdata\mailia\roadsblobs2.000 -o 0 -s 10000 chunk is added to the smart large object space roadsblob.

```
C:\Informix> onspaces -a roadsblob -p d:\Ifmxdata\mailia\roadsblobs2.000 -
o 0 -s 10000
```

Note: The logging of smart large objects is independent of the rest of the Informix database. By default, smart large objects are not logged.

If users are going to load data for read-only purposes, there is no need to have smart BLOB logging enabled. Simply take a level 0 archive after loading the data and recoverability is ensured.

However, for data that is manipulated additional steps must be taken to ensure recoverability. Database and smart BLOB logging must be enabled following the initial data load. After you enable database and smart BLOB logging, a level 0 archive of the instance must be taken to ensure recoverability.

To turn logging on, add -Df LOGGING=ON to the onspaces command that creates the sbspace.

C:\Informix> onspaces -a <sbspace name> -p <pathname> -o <offset> -s <size in kilobytes> -Df LOGGING=ON

The logging mode of an sbspace and the smart BLOBs it contains can be changed with onspaces:

C:\Informix> onspaces -ch <sbspace name> -Df "LOGGING=<logging mode>"

UNIX systems

Updating the onconfig file

Informix maintains its configuration parameters in the onconfig file located in the $INFORMIXDIR/etc on UNIX. The parameters of this file control the server's memory use, the size and number of log files, temporary space, the location of the error logs, and much more. The onconfig file is read whenever the Informix server is started. So changes to the parameter require that you restart the server.

Naming the onconfig file

The standard onconfig file, onconfig.std, contains the default settings of the Informix parameters. Do not edit this file; instead, preserve it as a record of the default settings.

On UNIX systems, manually copy the onconfig.std file to a new file name such as onconfig.sde.

For the remainder of this document, when the onconfig file is mentioned, the $INFORMIXDIR/etc/<your copied file> on UNIX is being referred to.

On UNIX systems, add the ONCONFIG system variable to the INFORMIX .cshrc or .profile file. For example, if you have named your onconfig file onconfig.sde, you would set the ONCONFIG variable to that.

```
setenv ONCONFIG onconfig.sde
```

Some important onconfig parameters

The following is a list of some of the more important onconfig parameters whose default values you should change to improve the performance of your Informix server when using it with ArcSDE.

BUFFERS

The BUFFERS parameter file controls the size of the regular buffers, the area of memory in which Informix stores the most recently used page of data. The first reader reads the page from disk, while subsequent readers read the page from the regular buffer until it is paged out of memory. A page will be paged out of the regular buffer if it is has been unused over a period of time and the memory is needed to hold other pages that are being used.

Increase the number of data buffers to 2,000 or 25 percent of your physical RAM, whichever is greater. BUFFERS is specified in pages. If your pages are 2 kilobytes (page size can be determined with the Informix command onstat -b) and your physical RAM is 256 MB, BUFFERS would be calculated as follows:

```
BUFFERS = <physical RAM converted to kilobytes> * 25% /
          <page size in kilobytes>
        = (256 * 1024) * 0.25 / 2
        = 32768

BUFFERS 32768
```

LOGSIZE

The LOGSIZE parameter controls the default size of the logical logs. The size of the logical logs can be specified when they are created with the INFORMIX onparams utility. However, if the size is not specified, LOGSIZE is used.

Set the logical log file size to 100,000 kilobytes. When the logical logs are moved out of the rootdbs, they will be created with this size.

```
LOGSIZE 100000
```

LOG_BACKUP_MODE

The LOG_BACKUP_MODE parameter specifies the mode in which logical logs are backed up. This mode can be either continuous or manual. Continuous mode will allow you to automatically do logical log backups when required.

LOGSMAX

The LOGSMAX parameter specifies the maximum number of logical logs that may be created. Increase the LOGSMAX parameter so that you can create new logical logs in order to move them out of the rootdbs.

Set the maximum number of logical log files to 100.

```
LOGSMAX 100
```

CLEANERS

CLEANERS specifies the page cleaner threads started by the INFORMIX instance. Page cleaner threads periodically wake up and perform background writes of batches of dirty pages held in the regular buffers to disk.

Set the number of page cleaners to 6 or the number of disks that contain frequently accessed data, whichever is higher.

```
CLEANERS 6
```

STACKSIZE

STACKSIZE specifies the amount of stack allocated to the INFORMIX instance. Although for most applications Informix recommends that this parameter be left at its default value of 32 (kilobytes), for ArcSDE it is important to increase the size of this parameter to 64 (kilobytes) in support of the Informix Spatial DataBlade user-defined datatypes accessed by ArcSDE.

Increase the initial stack size of each thread to 64 kilobytes. Set the STACKSIZE parameter to 64.

```
STACKSIZE 64
```

RA_PAGES

This read-ahead parameter sets the number of data and index pages that are cached in the regular buffers whenever a sequential scan of one or more tables occurs.

Set the read-ahead pages to 125.

```
RA_PAGES  125
```

RA_THRESHOLD

RA_THRESHOLD, the read-ahead threshold, specifies the number of remaining unread pages that triggers another call to read in more pages from disk.

Set the number of unprocessed pages that trigger another read ahead to 85.

```
RA_THRESHOLD 85
```

DUMPDIR

The DUMPDIR parameter specifies the location of the dump directory where error log files are written in the event of an assertion failure.

Leave the dump directory set to tmp if you have adequate space there. However, you can create a tmp directory under the Informix installation directory and set DUMPDIR to that. Should an assert failure occur, the diagnostic files are one directory below the online.log file that references them.

```
DUMPDIR /usr/informix/tmp       /* UNIX
```

RESIDENT

The RESIDENT parameter specifies which portion of the INFORMIX instances shared memory can be swapped out of the operating system's shared memory. Allowing as many portions of the instance's shared memory as possible to remain resident eliminates a large amount of I/O and context switching of the instance's memory structures.

Setting the RESIDENT parameter to -1 keeps as many of the instance's memory structures as possible resident, given the amount of physical memory and system resources available.

```
RESIDENT -1
```

MULTIPROCESSOR

The MULTIPROCESSOR parameter specifies whether the Informix Server machine has one or multiple processors which to use.

Set to 0 if the Informix Server machine has only one processor, and set to 1 if there are multiple processors.

System parameters that must be adjusted prior to initialization

ROOTPATH

The ROOTPATH parameter specifies the initial chunk of the root dbspace. The default setting /dev/online_root causes the initialization of the INFORMIX instance to fail unless you have actually created the device beforehand. Change the default setting from

/dev/online_root path to the device of rootdbs space you have created. For example, after creating the device with the UNIX *touch* command as the informix user and setting its permissions to 660 with the UNIX *chmod* command, set the ROOTPATH to the full pathname of the root dbspace chunk file. If you are using a raw device, set the ROOTPATH to the full pathname of the link to the raw device.

```
ROOTPATH  /disk1/informix_data/rootdbs
```

MSGPATH

The MSGPATH parameter specifies the full pathname to the message log file that the database server will write status and diagnostic messages to.

Update MSGPATH to reflect the location of your Informix installation.

```
MSGPATH  /disk1/informix/online.log
```

ALARMPROGRAM

The ALARMPROGRAM parameter specifies the full path of the script that will be executed when a log full event is issued. Set the parameter to log_full.sh to have the logical logs backed up automatically and to no_log.sh if you intend to back up the logs manually.

Update ALARMPROGRAM to reflect the location of your Informix installation.

```
ALARMPROGRAM  /disk1/informix/etc/log_full.sh
```

TAPEDEV

The TAPEDEV parameter specifies the device used to back up the dbspaces. During the loading phase of your database, it is often a good idea to set this parameter to the /dev/null device. After the data is loaded, set the parameter to the proper tape device. The rationale behind this is that the data is already backed up by the data source that you are loading it from. Therefore, if a dbspace is lost to a disk failure, the data can be restored from the original data source. Once the database is loaded, you can set it to your tape device.

```
TAPEDEV  /dev/null
```

LTAPEDEV

The LTAPEDEV parameter specifies the tape device to which the ONTAPE utility backs up the logical log files.

Set this to the /dev/null device. Once the server is up, you can set it to your tape device if you intend to archive the log files.

```
LTAPEDEV  /dev/null
```

DBSERVERNAME

The DBSERVERNAME parameter specifies the unique name of your database server. The *dbservername* is assigned a communications protocol in the *sqlhosts* file. Typically the dbservername is set to the database server name that is associated with the shared memory communications protocol. The DBSERVERALIASES parameter normally holds the database server name associated with the TCP/IP communications protocol.

Set this value to the lowercase name of your shared memory server.

```
DBSERVERNAME  gis
```

DBSERVERALIASES

Set this value to the lowercase name of your TCP/IP server.

```
DBSERVERALIASES  gis_net
```

NETTYPE

Set separate NETTYPE parameters to configure the poll threads for the shared memory and TCP/IP network protocols. The settings below allow 20 local connections and 200 remote connections. The configuration of the NETTYPE parameter is discussed in detail in the 'Network virtual processors' section later in this chapter. Set the NETTYPE parameters to the expected number of local and remote connections, as in the example for Solaris 2 below:

Solaris 2

```
NETTYPE  ipcshm,1,20,CPU
NETTYPE  tlitcp,2,100,NET
```

Following is the NETTYPE parameters for UNIX configurations:

HP

```
NETTYPE  ipcstr,1,20,CPU
NETTYPE  soctcp,2,100,NET
```

IBM

```
NETTYPE  ipcshm,1,20,CPU
NETTYPE  soctcp,2,100,NET
```

Restarting the INFORMIX server

To restart the INFORMIX server on the UNIX system first shut the server down by issuing the *onmode -ky* command at the UNIX prompt while logged in as the informix user.

```
informix> onmode -ky
```

Then restart the server with the *oninit* command.

```
informix> oninit
```

Tuning disk I/O contention

Disk I/O contention can prove to be one of the more difficult challenges for a DBA to overcome. Unlike memory and CPU issues that can be solved by acquiring more of these resources once all tuning procedures have been exhausted, the reduction of disk I/O contention must be solved through proper planning and administration of the file system.

Beyond the possibility of acquiring faster disk drives and controllers, the only real way to reduce disk I/O contention is to balance the I/O across the entire file system by distributing files that experience a high frequency of I/O with those that do not.

RAID systems

Redundant Arrays of Inexpensive (or Independent) Disks (RAID) boost performance by striping data into slices across multiple disks in a disk array. By spreading data across multiple disks, all disks share the burden of I/O operations, thus reducing the chance of a bottleneck occurring on one disk. RAID's performance increases as you add disks to the array. The operating system and database will see only one volume, a logical representation of the entire disk array.

In a simple configuration, you could create a single disk array of four disks and configure one large data file within that RAID array. Your data would be striped across all four disks evenly, reducing contention. The database's transaction log should not occupy this same array. This solution proves scalable as well—additional performance benefits can be gained by adding disks to the array until performance increases begin to decline. More complex configurations would include separate disk arrays for indexes, data tables, and geometry data.

Creating the system dbspaces

In the section 'Arranging your data', which follows, you will learn how to create dbspaces to store your business tables and indexes. Before you start creating these dbspaces, however, create dbspaces to serve as temporary storage for the transitional functions of the Informix server. Logical log files, physical log files, and temporary space for sorting should occupy their own dbspace.

Depending on the available number of disks, try to spread the devices of the dbspaces across your file system. Try to keep the devices of the physical and logical logs separate. Either the physical log or the logical log may share the same disk as the root device.

The temporary sorting devices (commonly referred to as temp devices) should be separated from all other devices, if possible. These temp devices are used heavily during the creation of the R-Tree index after data loading.

Therefore, it is a good rule of thumb to start with at least 300 MB of temporary storage spread across at least two sorting devices to handle the loading of large datasets and their associated R-Tree index building.

You may need to monitor the temporary space usage during the loading of large datasets to make sure Informix does not run out and produce an error. If this happened, it would typically leave the ArcSDE table in "load-only mode".

Here is a UNIX example of creating the physical log, logical logs, and temporary dbspace devices. When these devices are first created, they are empty and occupy zero space on the disk. After the dbspaces are assigned to them by the onspaces command, the devices immediately grow to the size allocated by the dbspace.

Note: These examples use cooked devices, which do not provide the best performance on a UNIX system. For best performance you should create all dbspaces on a UNIX raw device. Consult the *INFORMIX-Universal Server Administrator's Guide* for advice on creating dbspaces on raw devices.

Create the device for the physical log.

```
gis> touch /gis1/informix_data/phydbs
gis> chmod 660 /gis1/informix_data/phydbs
```

Create three devices for the logical logs.

```
gis> touch /gis2/informix_data/log1dbs
gis> chmod 660 /gis2/informix_data/log1dbs
gis> touch /gis3/informix_data/log2dbs
gis> chmod 660 /gis3/informix_data/log2dbs
gis> touch /gis3/informix_data/log3dbs
gis> chmod 660 /gis3/informix_data/log3dbs
```

Create two devices for sorting.

```
gis> touch /gis4/informix_data/temp1dbs
gis> chmod 660 /gis4/informix_data/temp1dbs
gis> touch /gis5/informix_data/temp2dbs
gis> chmod 660 /gis5/informix_data/temp2dbs
```

The Informix onspaces utility manages dbspaces. Use it to create the dbspaces and assign them to the devices that you have just set up. The onspaces syntax varies slightly depending on the kind of dbspace it is operating on. However, the basic syntax for creating the system dbspaces is:

```
onspaces -c -d <dbspace_name> -p <path to device> -o <offset> \
         -s <size in kilobytes>

onspaces -c -t -d <dbspace_name> -p <path to device> -o <offset> \
         -s <size in kilobytes>
```

The -t flag is included to indicate that the dbspace will be used for sorting and other temporary activities.

In the UNIX example below, dbspaces are created for the logical logs, physical log, and temporary space.

```
# Create the first dbspace for logical logs
onspaces -c -d log1dbs -p /gis2/informix_data/log1dbs -o 0 -s 125000

# Create the second dbspace for the logical logs
onspaces -c -d log2dbs -p /gis3/informix_data/log2dbs -o 0 -s 125000

# Create the third dbspace for the logical logs
onspaces -c -d log3dbs -p /gis3/informix_data/log3dbs -o 0 -s 125000

# Create the dbspace for the phydbs
onspaces -c -d phydbs -p /gis1/informix_data/phydbs -o 0 -s 10000

# Create the first temporary dbspace
onspaces -c -t -d temp1dbs -p /gis4/informix_data/temp1dbs -o 0 -s 150000

# Create the second temporary dbspace
onspaces -c -t -d temp2dbs -p /gis5/informix_data/temp2dbs -o 0 -s 150000
```

Moving the physical log out of the root dbspace

Moving the physical log out of the root dbspace reduces the I/O contention. Simply change the PHYDBS parameter in your onconfig file to the dbspace you have just created for physical logging. In our example the PHYSDBS parameter would be set to phydbs.

```
PHYSDBS phydbs
```

Increase the size of the PHYSFILE to use the space allocated to the physical log's dbspace. In the example, the phydbs dbspace is 10,000 kilobytes, allowing us to increase the PHYSFILE to 9000. It cannot be increased to the size of the dbspace because Informix uses a certain amount of space for overhead.

```
PHYSFILE 9000
```

Shut down and restart the Informix server to use the phydbs dbspace for physical logging.

UNIX users use the onmode command to shut down the server and the oninit command to start it.

```
gis> onmode -ky
```

gis> **oninit**

Moving the logical logs out of the root dbspace

For the same reason you moved the physical log from the root dbspace, you must do the same for the logical logs. First, make sure the LOGSMAX parameter in the onconfig file is set high enough.

By default, the installation creates six logs in the root dbspace on a UNIX platform. To add 20 log files to an Informix server on a UNIX platform, LOGSMAX must be set to at least 26.

Make sure you create enough logical logs to handle your longest transaction. Typically, long transactions occur when you create or delete a large dataset or when you compress a geodatabase. You must checkpoint your logical logs by backing them up before you reach the long transaction high watermark percentage defined by the LTXHWM parameter in your Informix onconfig file. You should not change either the LTXHWM or LTXEHWM without the consent of an Informix technical support expert who is familiar with the behavior of the Informix Spatial DataBlade. If a transaction fails to complete and is rolled back because it reaches the long transaction high watermark, then you do not have enough logical logs.

To create the new logical logs, first set the server in quiescent mode by issuing the onmode command with -s flag.

```
gis> onmode -s
```

To add logical log files to each of the dbspaces created for them, use the Informix onparams utility. When you add the log files, make sure you alternate between at least two dbspaces. This ensures that while one log file is being flushed from one disk drive, another can be written to on another disk drive.

```
gis> onparams -a -d log1dbs
gis> onparams -a -d log2dbs
gis> onparams -a -d log3dbs
```

Activate the new logical logs by performing a zero-level archive with the Informix ontape utility.

```
gis> ontape -s
```

Now you can delete the original six logical logs that reside on the root dbspace. First, you must determine if one of the first six logical logs is the current one. Use the onstat -l command to generate a list of the logical logs.

```
gis> onstat -l
```

The logical log is current if its flags column contains a C. Find this logical log and note its number. If the number is between one and six on a UNIX platform, you must advance the log with the onmode -l utility.

```
gis> onmode -l
```

Repeat the onstat -l followed by the onmode -l utility until a logical log with a number greater than six for a UNIX platform, becomes current.

Then use the onparams -d utility to drop the logical logs in the root dbspace.

```
gis> onparams -d -l 1 -y
gis> onparams -d -l 2 -y
gis> onparams -d -l 3 -y
gis> onparams -d -l 4 -y
gis> onparams -d -l 5 -y
```

```
gis> onparams -d -l 6 -y
```

Use the ontape -s utility command to archive the change. The output of the onstat -l utility should list only those log files that were added to the log1dbs, log2dbs, and log3dbs spaces. The output of the onstat -l on the UNIX platform should begin with log file number 7.

Put the server back to online mode with the onmode -m utility.

```
gis> onmode -m
```

Setting up the temporary dbspace

By default, Informix uses the root dbspace when it needs temporary space for sorting. The creation of a large index (such as R-tree index) can fill the root dbspace, resulting in a server crash. Or, in the case of loading data using the ArcCatalog product, the creation of the rtree index (the last step in loading data) will fail, and the layer loaded will remain in load-only mode.

It is better to control the location of the temporary space by using separate dbspaces and adding chunks of space as necessary. Set the DBSPACETEMP parameter in the onconfig file to the temporary dbspaces created earlier. Remember to use at least two dbspaces totaling 300 MB as a starting point. Then add chunks accordingly, making sure that these chunks span different disks whenever possible.

```
DBSPACETEMP temp1dbs,temp2dbs
```

Restart the Informix server to set the temporary space in the server. On UNIX platforms the onmode -ky command shuts down the server and the oninit command starts it again.

```
gis> onmode -ky
gis> oninit
```

Examine the online.log file to ensure that the temporary space is set. You should see an entry stating that the temporary files have been relocated from the root dbspace to the dbspaces you assigned to the DBSPACETEMP parameter.

Creating the default smart large object dbspace

The Spatial DataBlade module writes the compressed geometry to the smart large object whenever it is larger than 929 bytes. For this reason a default smart large object space, or sbspace, must exist.

On the UNIX platform, an sbspace must be created. Create the device for the sbspace. On a UNIX platform this is done with the touch and chmod commands.

```
gis> touch /gis6/informix_data/sblobdbs
gis> chmod 660 /gis6/informix_data/sblobdbs
```

Use the onspaces utility to create the sbspace. The -S flag directs the onspaces utility to create an sbspace to store a smart large object. Set the -g flag to 1.

```
onspaces -c -S sblobdbs -g 1 -p /gis6/informix_data/sblobdbs -o 0 -s
300000
```

Set the default smart large object space parameter SBSPACENAME in the onconfig file to the sbspace you created.

Restart the Informix server to set the default sbspace by invoking the onmode -ky utility to shut down the server followed by the oninit utility to start it again.

Check the online.log file; look for a message stating that the default smart large object space has been changed to the sbspace you specified in the onconfig file.

Allocating enough metadata within a smart large object sbspace

Make sure to also include enough space when creating both the sbspace and syssbspace for system metadata. Informix automatically creates the system metadata when you create a smart large object sbspace; however, it is usually a small percentage of the total sbspace space.

If the smart large object sbspace uses all the space allocated to the metadata, Informix returns an "out of smart large object dbspace" error after trying to store data even though plenty of smart large object sbspace exists.

Large datasets can require large amounts of smart large object metadata sbspace. Define the amount of smart large object sbspace to allocate to the metadata with the -Ms option of the following onspaces command when you create the sbspace.

```
gis> onspaces –c –S sblobdbs –g 1 –p /gis1/ifmxdata/mailia/sblobdbs –o 0 –
s 350000 –Ms 50000
```

In this example, Informix allocates 50 MB of the total 350 MB of smart large object sbspace to metadata in the smart large object dbspace sblobdbs. You can also allocate space to the metadata when you add a chunk to the sbspace:

```
gis> onspaces –a sblobdbs –p /gis1/ifmxdata/mailia/sblob1dbs –o 0 –s
350000 –Ms 50000
```

You can allocate all of the space to metadata when you add a chunk to smart large object sbspaces by specifying the same values for the -Ms and -s flags:

```
gis> onspaces –a sblobdbs –p /gis1/ifmxdata/mailia/sblob1dbs –o 0 –s
350000 –Ms 350000
```

The metadata portion of a smart large object can be monitored with the "dbstat –d" command. Check the metadata available space for each smart large object sbspace.

Using smart large object sbspaces

In order to avoid the possibility of running out of default smart large object sbspace to hold both spatial column data or annotation column data, create separate smart large object sbspaces. Keep the default sbspace and syssbspace small for Informix system use

only. You should make the sbspace and syssbspace separate smart large object sbspaces, but it is not necessary.

Smart large object sbspaces can be assigned to spatial columns or annotation columns using the DBTUNE table S_STORAGE parameter. For more information on the S_STORAGE parameter, see chapter 3, 'Configuring DBTUNE storage parameters'.

Smart large object space at ArcSDE 9

At Spatial Database Engine (SDE) 3.0.2.2 for Informix, the SDE large binary datatype SE_BLOB_TYPE was stored as the Informix datatype BYTE.

For ArcSDE 9 for Informix, the ArcSDE large binary datatype is stored as the Informix BLOB data type.

Since BLOB is stored in the smart large object BLOB spaces, you need to specify the default smart large object BLOB space in the ONCONFIG file and make sure it is large enough to hold your large binary data. Make sure you allocate enough space for the sbspace metadata.

The preferred alternative to using the default smart large object sbspace is to create separate smart large object sbspaces and control the placement of spatial columns and annotation columns into these smart BLOBs using the storage parameters of the DBTUNE table. See Chapter 3, 'Configuring DBTUNE storage parameters', for details.

Arranging your data

Proper arrangement of tables and indexes on the file system will help to minimize disk I/O bottlenecks. Placement of these data objects requires that you estimate their size and create the dbspaces they will be stored in. You add the dbspaces, names along with a list of other data object storage parameters to a DBTUNE table configuration keyword. To learn more about the storage parameters of the DBTUNE table, see chapter 3, 'Configuring DBTUNE storage parameters'. The ArcSDE server uses the parameters when it creates the data objects with the ArcSDE administration commands.

Creating the dbspaces and sbspaces

A dbspace is a logical unit of storage that Informix uses to store tables and indexes. An sbspace is a logical unit of storage designed specifically to store smart large objects. Both are created and maintained by the onspaces command. Both may have one or more physical units of storage assigned to them. The physical units of storage are called chunks. Chunks may be either raw devices or cooked files.

Informix recommends the use of raw devices on UNIX platforms because they provide faster access and higher reliability in the event of a system failure. On UNIX platforms

cooked files are adequate for demonstrations and storing tables that are infrequently updated.

The size and placement of dbspaces and sbspaces depend on the tables and indexes stored within them. Here are some basic guidelines to help determine the size and placement of your tables, indexes, and smart large objects. Given the number and size of the disk drives available on your system, you may not be able to follow these guidelines to the letter, but follow them as best you can, keeping in mind that the goal is to minimize disk I/O contention.

Separate indexes onto a different disk drive from the tables they index

If the Informix optimizer determines that an index will speed up the execution of a query, it will read pages of the index into memory, search the pages for a match, and read matching table records from disk into memory. Storing the index and table on the same disk forces the disk head to unproductively travel back and forth between the index and the table. Arranging the tables and indexes on separate disks allows multiple disk heads to simultaneously read from the index and the table, reducing disk head travel and seek time.

Separate smart large objects from their associated spatial tables

Spatial data too large to be stored inline with other table data is written to the designated smart large object. As with indexes, smart large object sbspaces should be stored on a disk separate from both the table and the indexes.

Place high-use tables in the middle disk drive partitions to minimize disk head movement

Placing high-use tables in the middle partitions of a disk drive reduces disk head travel. Based on the law of averages, arranging data so that the disk head spends most of its time in the middle partition reduces travel. Consult your operating system configuration manual for directions on partitioning your disk drives. Allocate a single chunk to the partition and assign the dbspace of the high-use tables to it.

Separate large high-use tables on to different disk drives

Balance disk I/O by spreading large high-use tables throughout the file system. Discuss the application model with the designers to determine which tables will be accessed most. Arrange these tables on separate disks to ensure equal employment of disk heads and controllers.

Group smaller tables together into dbspaces by usage

Creating a separate dbspace for each table in your database is unrealistic. Each dbspace has an associated overhead cost, and it’s cumbersome to manage a large number of

dbspaces. Group smaller tables together into a single dbspace. You should also group the related indexes into another dbspace so they may be placed on a separate disk drive.

Grouping the smaller tables by usage into separate dbspaces allows you to place the high-use smaller tables into the middle partitions.

Optimize extent sizes

Estimating the size of your tables and indexes allows you to allocate the initial extent to contain the entire data object. For data objects grouped together into the same dbspace this prevents their extents from becoming interleaved. Interleaved extents can reduce performance if the disk head has to seek over the extents of other tables.

Assign individual dbspaces to large tables

Large tables should have their own dbspaces. This allows you to move these tables easily throughout the file system. Some tables may be so large that the dbspace assigned to them requires more than one chunk. If so, it's a good idea to separate the chunks onto different disk drives and separate controllers if possible. Doing so allows multiple access to data of the same table and reduces overall seek time.

Using onspaces to create dbspaces and sbspaces

The Informix onspaces command creates and maintains dbspaces and sbspaces. Run the onspaces command as the informix user.

Dbspaces

Dbspaces are created with the onspaces command using the following basic syntax:

```
gis> onspaces -c -d <dbspace name> -p <pathname> -o <offset> \
         -s <size in kilobytes>
```

The dbspace name must be unique within the database. The pathname specifies the location of either a raw device or a cooked file. If it is a cooked file, the file must exist, and the informix user and group must have read and write permissions to the file. On UNIX platforms, before you invoke the onspaces command, use the touch and chmod commands to create the file and set the permissions.

In this example the UNIX touch command creates the cooked file /gis6/informix_ck/roadsdbs1 and the chmod command changes its permissions to read and write access for the informix user and group. The onspaces command creates the roadsdbs dbspace and allocates 50,000 kilobytes to the /gis6/informix_ck/roadsdbs1 for its initial chunk.

```
gis> touch /gis6/informix_ck/roadsdbs1
gis> chmod 660 /gis6/informix_ck/roadsdbs1
gis> onspaces -c -d roadsdbs -p /gis6/informix_ck/roadsdbs1 -o 0 -s 50000
```

Additional chunks may be added to a dbspace with

```
gis> onspaces -a <dbspace name> -p <pathname> -o <offset> -s <size in
kilobytes>
```

In this example the 50,000-KB chunk /gis7/informix_ck/roadsdbs2 is added to the dbspace roadsdbs2.

```
gis> onspaces -a roadsdbs -p /gis7/informix_ck/roadsdbs2 -o 0 -s 50000
```

Sbspaces

Smart large object spaces are created in the same fashion as dbspaces. The -S flag directs the onspaces command to create a smart large object space instead of a regular dbspace. The basic syntax for creating a smart large object space is:

```
gis> onspaces -c -S <sbspace name> -p <pathname> -o <offset> -s <size in
kilobytes>
```

In this example the sbspace roadsblob is created with the 10,000 KB initial chunk /gis8/informix_ck/roadsblobdbs1.

```
gis> onspaces -c -S roadsblob  -g 1 -p /gis8/informix_ck/roadsblobdbs1 -o
0 -s 10000
```

You can add additional chunks to the smart large object space with the following onspaces syntax. You will notice that the -S flag is not required when adding a chunk.

```
onspaces -a <sbspace name> -p <pathname> -o <offset> -s <size in
kilobytes>
```

In this example the /gis9/informix_ck/roadsblobs2 -o 0 -s 10000 chunk is added to the smart large object space roadsblob.

```
gis> onspaces -a roadsblob -p /gis9/informix_ck/roadsblobs2 -o 0 -s 10000
```

Note: The logging of smart large objects is independent of the rest of the Informix database. By default, smart large objects are not logged.

If users are going to load data for read-only purposes, there is no need to have smart BLOB logging enabled. Simply take a level 0 archive after loading the data and recoverability is ensured.

However, for data that is manipulated additional steps must be taken to ensure recoverability. Database and smart BLOB logging must be enabled following the initial data load. After you enable database and smart BLOB logging, a level 0 archive of the instance must be taken to ensure recoverability.

To turn logging on add -Df LOGGING=ON to the onspaces command that creates the sbspace.

gis> onspaces -a <sbspace name> -p <pathname> -o <offset> -s <size in kilobytes> -Df LOGGING=ON

The logging mode of an sbspace and the smart BLOBs it contains can be changed with onspaces:

gis> onspaces -ch <sbspace name> -Df "LOGGING=<logging mode>"

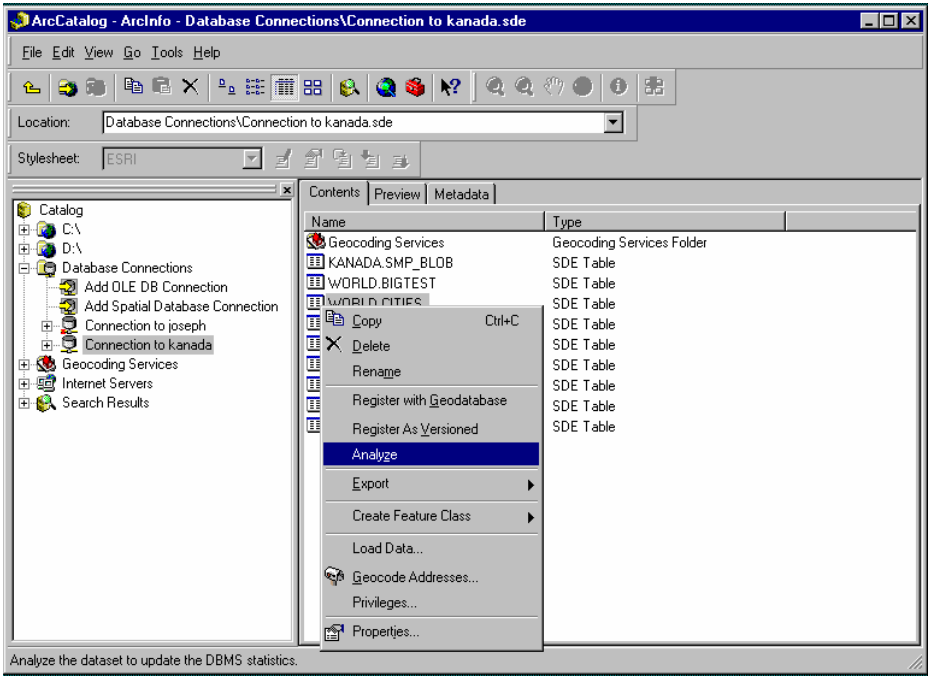
You must also log your database. It is recommended that you use buffered logging. To change the state of an ArcSDE database from no logging to logging, use the following command:

gis> ontape -B <arcgis_database_name>

Updating Informix statistics

For optimal performance of feature classes created with ArcSDE, keep the statistics of the business table up-to-date by frequently updating statistics.

To update the statistics of all of the tables and indexes within a feature dataset in ArcCatalog, right-click on the feature dataset and click Analyze. To update the tables and indexes within a feature class, right-click the feature and click Analyze as shown below.



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 Informix SQL UPDATE STATISTICS 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 statistics for all the required tables, do not specify the -K (schema object) option.

```
sdetable -o update_dbms_stats -t roads -m high -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.

You must periodically update your dynamic data objects' statistics so that the Informix optimizer will continue to choose an optimum execution plan. To save time, you can analyze all of the data objects within a feature dataset in ArcCatalog.

If you decide to update the statistics of all or some of the feature class tables with the Informix UPDATE STATISTICS statement, you should never compute statistics on a spatial index table. For more information on the Informix SQL UPDATE STATISTICS statement, refer to the *Informix SQL Reference Manual*.

The statistics of a table's indexes are automatically computed when the statistics on the table are created, so there is no need to separately generate statistics for the indexes. However, if you need to do so you can use the sdetable UPDATE_DBMS_STATS operation with the -n option and the index name.

The example below illustrates how the statistics for the roads_idx index of the roads business table can be updated.

```
sdetable -o update_dbms_stats -t roads -K B -n roads_idx -u av -p mo
```

For more information on updating the statistics on 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 ArcSDE Developer Help.

Tuning CPU

Many server-class machines are multiprocessors—computers that contain more than one CPU and parallel process several instructions at a time. The initial configuration of Informix Dynamic Server defaults to single CPU mode. Doing so ensures that Informix Dynamic Server starts and runs correctly on single CPU servers. If yours is a multiple CPU server, configure Informix to take advantage of its parallel processing capabilities.

If you have a single CPU machine, set SINGLE_CPU_VP to 1, set MULTIPROCESSOR to 0, and keep them that way. Setting SINGLE_CPU_VP to 1

bypasses superfluous mutex calls required only when running multiple virtual processors. Setting MULTIPROCESSOR to 0 specifies that locking will be optimized for a single CPU processor.

For multiple processor machines set the SINGLE_CPU_VP to 0 and set MULTIPROCESSOR to 1. Setting these variables will allow your Informix server to take advantage of a machine's parallel processing capabilities.

During its initialization the Informix Dynamic Server creates virtual processors to run the various threads that service the client applications and other background tasks. Virtual processors are similar to operating system processes, but they are controlled and manipulated by the Informix Dynamic Server. Some of the virtual processors are divided into classes, some of which are tunable. Of particular interest are the CPU, AIO, and network virtual processor classes.

You may configure the CPU and AIO virtual processors by setting a list of onconfig parameters that include NUMCPUVPS, AFF_SPROC, AFF_NPROCS, NOAGE, and NUMAIOVPS. However, Informix recommends that you set the VPCLASS parameter for each virtual processor class. If you use the VPCLASS method, then you cannot use the other variables. If Informix detects the presence of both types of parameters in the onconfig file, it returns an error and will not start the server.

Because Informix recommends the use of the VPCLASS parameter, the VPCLASS parameter will be discussed in this document. If you would rather use the other parameters, consult the *INFORMIX-Universal Server Administrator's Guide*.

The basic syntax of the VPCLASS parameter is

VPCLASS *classname,{num=num_VPS,max=max_vps,aff=affinity,noage,noyield}*

The classname is the only required field. It is possible to name and create your own user-defined virtual processor classes. However, ESRI Spatial DataBlade uses the predefined CPU VP class. Unless you are using a DataBlade product or custom-built application that requires the presence of a user-defined virtual processor class, do not create one.

CPU and AIO virtual processor classes

In addition to other possible uses, the AIO virtual processor maintains the Informix server ancillary files, such as the message log, so you should always define at least one.

UNIX systems use the AIO virtual processor for nonlogged I/O if kernel-asynchronous I/O (KAIO) is not implemented or if the I/O performed is to a cooked file.

On UNIX systems, the AIO virtual processor is not used for nonlogged I/O if you have implemented KAIO and the I/O is to a raw device. Instead, the CPU virtual processor

performs the I/O. Having the CPU class perform the I/O avoids expensive context switching between the CPU virtual processor and the AIO virtual processor.

To implement kernel-asynchronous I/O on a UNIX system, consult the IDS_9.2 release notes file under the $INFORMIXDIR/release directory. The file contains instructions that tell you whether or not KAIO is enabled by default, how to enable or disable KAIO, and which operating system patches are required to enable it.

Informix recommends the use of raw devices on UNIX systems whenever possible. A raw device is a UNIX block device—in this case a hard disk—that's been configured with a 'character-special' interface allowing the application, rather than the operating system, to perform the I/O buffering. The counterpart, a cooked device, is managed by the operating system. All I/O to a cooked device is buffered by the operating system.

Using a raw device on UNIX operating systems guarantees that committed data has been written to disk. Also, the performance is better because data is transferred directly to shared memory and not copied first to the operating system's kernel buffer pool and then to shared memory.

Logged I/O occurs whenever a change is made to a table record. Logged I/O is written to both the physical log file and the logical log file. On Windows 2000 systems, a CPU virtual processor always runs a KAIO thread to perform logged I/O. However, on UNIX systems, the LIO virtual processor performs the I/O to the logical log I/O, and the PIO virtual processor performs the physical log I/O if the log files are stored on a cooked file system or KAIO is not implemented. If both KAIO and raw devices are used, the CPU virtual processor performs the logged I/O, eliminating the expensive context switching between the CPU virtual processor and the LIO and PIO virtual processors.

The number of CPU virtual processors configured should not exceed the number of processors in the system. Set the VPCLASS num option to 2 and the max option to the number of processors on the server.

For example, if the server contains eight processors, the VPCLASS parameter would be set as follows:

```
VPCLASS cpu, num=2, max=8
```

Informix recommends that you configure at least one AIO virtual processor to handle the ancillary I/O. The recommendation always applies to Informix servers installed on Windows systems and on UNIX systems if KAIO has been implemented and all nonlogged I/O is to a raw device.

For UNIX systems that have implemented KAIO but are using some cooked file space, Informix recommends you allocate two AIO virtual processors per active dbspace composed of cooked file space. If four dbspaces use cooked file space and KAIO was implemented, configure the AIO virtual processors as:

```
VPCLASS aio, num = 8
```

If KAIO is not implemented, Informix recommends that you allocate two AIO virtual processors for each disk the Informix server accesses frequently. For example, if KAIO is not implemented and eight disks are accessed frequently, implement the AIO virtual processor as:

```
VPCLASS aio, num = 16
```

Network virtual processors

Network virtual processors are implicitly defined by the NETTYPE parameter. The NETTYPE parameter defines the number of poll threads allocated for each database connection type. The NETTYPE parameter defines the connection protocol, number of poll threads, concurrent connections per poll thread, and type of virtual processor that will run the connection's poll thread. A poll thread can be run by a CPU virtual processor or by a network virtual processor. The network virtual processors include SHM, SOC, STR, and TLI. All of the network virtual processors are defined under the general virtual processor class NET. When the server is initialized it starts one network virtual processor or each poll thread of the protocol defined by the NETTYPE parameter.

If the onconfig file contained the following NETTYPE parameter, as in the example below for IBM AIX® and HP-UX® platforms:

```
NETTYPE  soctcp,2,100,NET
```

two SOC virtual processors would be started to run the poll threads for the TCP/IP sockets protocol. The above example will accept 200 concurrent TCP/IP connections.

Poll threads can be run inline by a CPU virtual processor. Set the NETTYPE virtual processor to CPU rather than NET if you wish to have the poll thread for a particular connection type run by the CPU virtual processor. CPU virtual processors could run the TCP/IP socket poll threads. The NETTYPE parameter would be set as follows for the Windows 2000 platform:

```
NETTYPE  soctcp,2,100,CPU
```

Poll threads tend to run more efficiently on CPU virtual processors, particularly on single processor computers. If you have a lot of CPU virtual processors, it is possible to run all the poll threads on them. However, as user activity increases and the CPU virtual processors become congested, it is a good idea to off-load the work of the poll threads to network virtual processors. Poll threads are processed faster when run by network virtual processors, but they must still wait for a CPU virtual processor to run the sqlexec thread that processes the client's request.

As a rule of thumb, when you have a large number of remote clients, use network processors to run the poll threads for network connection protocols (soctcp or tlitcp, depending on the platform). Use the CPU processors to run the few local client poll threads (ipcshm or ipcstr, depending on the platform).

This is a typical NETTYPE configuration, an example for the Sun™ Solaris™ 2 platform:

```
NETTYPE     ipcshm,1,20,CPU
NETTYPE     tlitcp,2,100,NET
```

Priority aging

You should disable priority aging on the UNIX systems supported by the ArcSDE for Informix software. By default, priority aging is enabled for these systems. Over time, priority aging decrements the priority of long-running processes. Informix virtual processors run continuously and will run with a lower priority the longer the server is up, unless priority aging has been disabled. To disable priority aging, include the noage option on the VPCLASS parameter of each virtual processor class.

For example, if you wish to disable priority aging for the CPU virtual processor class, you would add the noage option.

```
VPCLASS CPU, num=2, max=8, noage
```

Processor affinity

Processor affinity allows you to assign the CPU virtual processors to specific CPU processors. Assigning CPU virtual processors directly to machine processors reduces context switching and improves performance. The virtual processors are assigned using the aff option of the VPCLASS parameter. The first processor starts at 0, and the last processor is the number of CPU processors on the machine minus 1. To assign eight CPU virtual processors to eight CPUs, the VPCLASS parameter would have the following syntax:

```
VPCLASS CPU, num=2, max=8, aff=0-7, noage
```

Not all platforms supported by ArcSDE for Informix software support processor affinity. See the table on the following page for a list of platforms that do.

	KAIO	Processor Affinity	No Aging
Sun Solaris	Yes	No	Yes
IBM	Yes	No	Yes
HP-UX	Yes	Yes	Yes
Windows 2000 NT	Default	Yes	Default

Kernel asynchronous I/O is supported on all platforms supported by the ArcSDE for Informix software as is priority aging. However, the operating system versions of the Sun and IBM platforms supported by the ArcSDE for Informix software do not support processor affinity.

No yield option

You cannot set the noyield option for either the CPU or AIO virtual processor class. This option can only be set for user-defined virtual processor classes. If you have created user-defined virtual processor classes and wish to know more about this option, consult the *INFORMIX-Universal Server Administrator's Guide*.

Tuning memory

Memory tuning involves the allocation of the resource to the various components of the Informix Dynamic Server. Each process running on a computer requires a certain amount of memory to temporarily store its machine code and data. Database management system servers also employ shared memory to store data used by many client applications. For a complete examination of this subject, you should consult the *INFORMIX-Universal Server Administrator's Guide* and the *INFORMIX-Universal Server Performance Guide*.

Buffers

Buffers are specified in system pages. The Informix server checks the regular buffers to find the pages it needs. If it doesn't find them, it reads them into the regular buffers before using them. Doing so avoids reading the pages from disk for each user, improving performance. Pages accessed from memory are much faster than from disk. The pages are maintained in the buffer as long as the instance remains up or until more recently accessed pages require the space. Informix recommends that, initially, the buffers occupy 20–25 percent of physical memory. For example, if the page on your system is 2 KB and you have 512 MB of physical memory, set BUFFERS to 65536 to occupy 25% of the physical memory.

Ideally, you want to keep the ratio of disk reads to buffer reads as low as possible. You can monitor the ratio by periodically issuing onstat -p and examining the %cached reads after the system has been up for a while. If this ratio falls below 90 percent for a decision support system, you should consider increasing the size of the regular buffers.

LRU queues and page cleaners

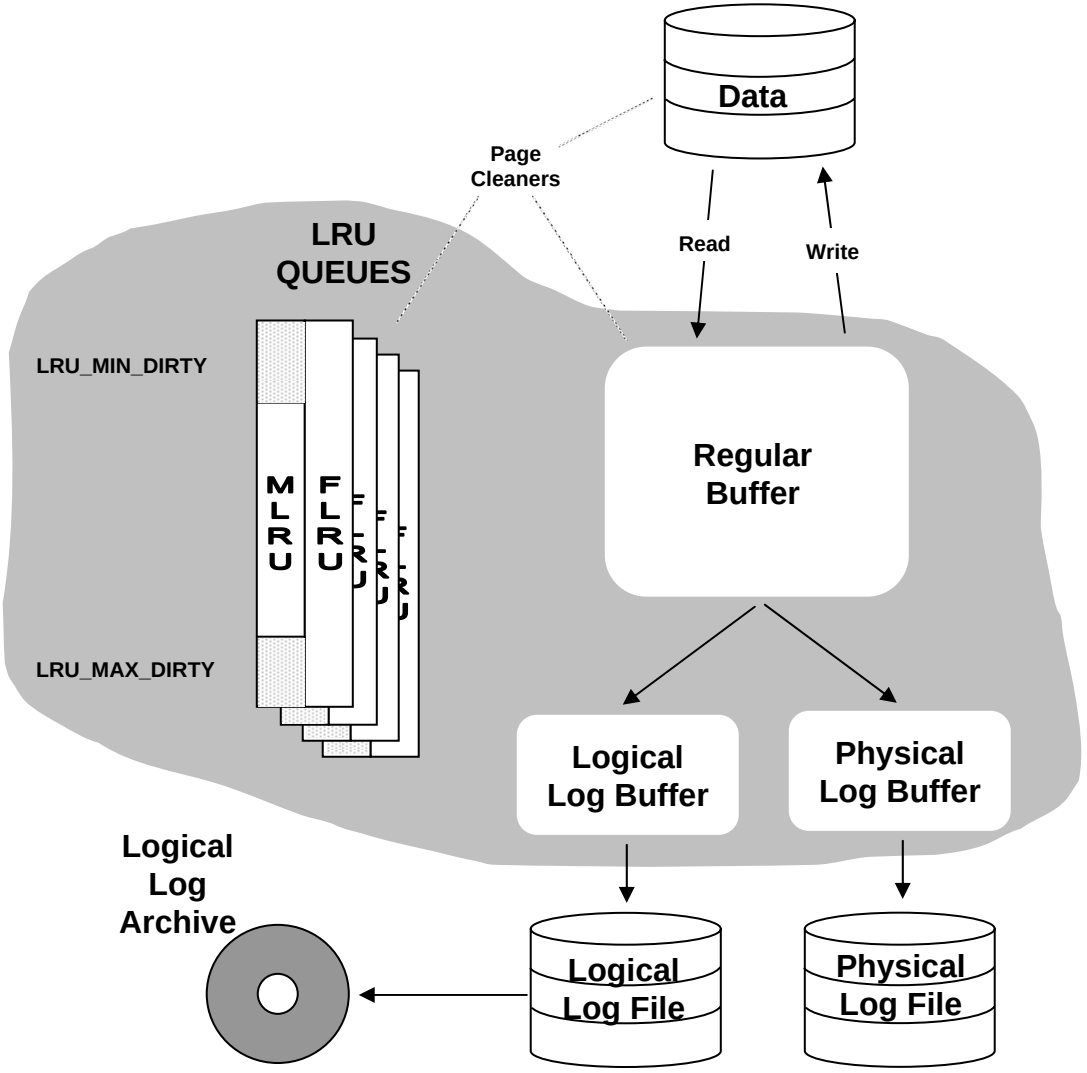
Each page read into the regular buffers is added to a least recently used (LRU) queue that tracks the frequency by which each page is accessed. If the regular buffers become full, the user sqlexec process selects an LRU queue at random to find pages that can be overwritten. The sqlexec process examines the free least recently used (FLRU) list of the LRU queue.

Pages are free if they have not been changed. Changed, or dirty pages move from the FLRU list to the modified least recently used (MLRU) list. If no free pages are listed in

FLRU, the sqlexec is forced to do a foreground write to clear pages from the MLRU queue. Foreground writes are performed one at a time and are resource intensive.

Page cleaners normally write dirty pages to disk. They perform inexpensive background writes. However, if the page cleaners are unable to keep up, foreground writes become necessary.

The onstat -F command can determine how often foreground writes have occurred. Avoid foreground writes by making sure enough LRU queues and page cleaners are available. The number of LRU queues is set by the LRU variable in the onconfig file. Informix recommends you set LRU to a minimum of 4. For multiprocessor machines, set LRU to four times the number of CPUs.



The figure above is a conceptual illustration of the Informix Dynamic Server's memory management. At the beginning of a transaction, pages are read into the data buffers from disk. An LRU queue is selected at random, and the address of the data buffer pages is added to the most recently used end of the free least recently used queue FLRU. Changes to the data stored in the data buffers move the buffers from the FLRU queue to the modified least recently used queue. A change also causes the before image of the data buffer page to be written to the physical log buffer, and the modifying SQL statement is written to the logical log buffer. When these buffers become full they are written to a corresponding file. In addition, the logical log files must be archived in order to roll the transactions forward in the event of a system failure. The page cleaner processes periodically wake up and write the accumulated changed data to disk. The page cleaners are activated whenever the user's process detects that the percent of dirty pages listed in the MLRU exceeds LRU_MAX_DIRTY. The page cleaners continue to write the dirty pages to disk until the percent of dirty pages is less than LRU_MIN_DIRTY.

The onconfig parameter CLEANERS controls the number of page cleaner threads running. Set the CLEANERS to the number of disks receiving frequent updates.

The LRU_MIN_DIRTY and LRU_MAX_DIRTY parameters determine when the page cleaners wake up and go to sleep. Page cleaners wake up whenever the percent of dirty pages in the MLRU reaches the LRU_MAX_DIRTY threshold, and they clean the dirty pages, starting with the least recently used, until the percent of dirty pages is less than LRU_MIN_DIRTY.

The default setting of the LRU_MIN_DIRTY is 50, and the default setting of the LRU_MAX_DIRTY is 60. If the onstat -F command shows a significant number of foreground writes, and increasing the number of LRU queues and page cleaners does not have any effect, try lowering the LRU_MIN_DIRTY and LRU_MAX_DIRTY thresholds to shorten the queues.

Logical log buffers

The Informix server uses three log buffers to temporarily store changes. As one buffer flushes to a log file, a user thread can write to another one. The log buffers generally default to 32 kilobytes or pages. Set the LOGBUFF to an even increment of the page size.

Determine the optimal size of the logical log buffer with the onstat -l command after the system has been running for a while in update mode. Onstat -l reports the current statistics of the physical log buffer and logical log buffer.

Under the Logical Logging section of the onstat -l output, if the value under pages/io is less than 75 percent of the value under bufsize, the logical log buffer is too large; shared memory is being wasted. Reduce the onconfig LOGBUFF parameter.

If the value under pages/io is greater than 95 percent of the value under bufsize, the logical log buffer may be too small. Increase the LOGBUFF parameter.

Changes to the onconfig file do not take effect until you restart the Informix server.

Physical log buffers

The Informix server uses two physical log buffers to temporarily store the pages that are about to be changed, commonly referred to as the before image. A physical buffer flushes to the physical log file once it becomes full. As one buffer flushes, the other buffer becomes current and the user thread writes to it.

The PHYSBUFF should always be an even increment of the page size.

Determine the optimal size of the physical log buffer with the onstat -l command after the system has been running for a while in update mode. Onstat -l reports the current statistics of the physical log buffer and logical log buffer.

Under the Physical Logging section of the onstat -l output, if the value under pages/io is less than 75 percent of the value under bufsize, the physical log buffer is too large; shared memory is being wasted. Reduce the onconfig PHYSBUFF parameter.

If the value under pages/io is greater than 95 percent of the value under bufsize, the physical log buffer may be too small. Increase the PHYSBUFF parameter.

Changes to the onconfig file do not take effect until the next time you start the Informix server.

Residency

When the operating system switches between processes, it normally pages portions of process memory to disk. Process memory designated as resident is not swapped to disk. Part of the Informix server's shared memory is resident, but the operating system will not treat it as such unless the onconfig RESIDENT parameter is set to 1. Set RESIDENT to 1.

CHAPTER 3

Configuring DBTUNE storage parameters

DBTUNE storage parameters allow you to control how ArcSDE clients create objects within an Informix database. They determine such things as which dbspace a table or index is created in. The storage parameters define the extent size of the data object they store as well as other Informix specific storage attributes. Additional parameters exist that allow you to further configure the Informix Spatial environment.

The DBTUNE table

The DBTUNE storage parameters are maintained 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 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 keywords. Within each configuration keyword, there are a number of storage parameters, and the names of these are stored in the parameter_name field. Each storage parameter has a configuration string stored in the config_string field.

After creating the DBTUNE table, the setup phase (sdesetupinfx command) of the ArcSDE 9 installation populates the table with the contents of the dbtune.sde file, which it expects to find under the etc directory of the SDEHOME directory.

If the DBTUNE table already exists, the ArcSDE setup phase will not alter its contents should you decide to run it again.

Editing the DBTUNE table

Although you are free to edit the contents of the DBTUNE table using a SQL interface such as DBACCESS, the sdedbtune administration tool has been provided to enable you to export the contents of the table to a file. The file can then be edited with a UNIX file-based editor such as vi or a Windows 2000 file-based editor such as notepad. After updating the file, you can repopulate the DBTUNE table using the import operation of the sdedbtune command.

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

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


ArcSDE   9                Wed Oct  4 22:32:44 PDT 2000
Attribute        Administration Utility
-----------------------------------------------------

        Successfully exported to file SDEHOME\etc\dbtune.out

$ vi dbtune.out

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


ArcSDE   9                Wed Oct  4 22:32:44 PDT 2000
Attribute        Administration Utility
-----------------------------------------------------

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

The sdedbtune administration tool always exports the file in the etc directory of the ArcSDE home directory. You cannot relocate the file to another directory with a qualifying pathname. By not allowing the relocation of the file, the sdedbtune command ensures that the file is located in the ArcSDE software's etc directory, under the ownership of the ArcSDE administrator.

The dbtune.proto file, located under the SDEHOME tools directory, provides an example of a DBTUNE file that can be edited and imported into the DBTUNE table.

Unlike the dbtune.sde file located under the SDEHOME etc directory, the dbtune.proto file contains suggested nondefault values that you may or may not want to use.

Adding keywords to the DBTUNE table

You may add parameter groups to the DBTUNE table for any special purpose. For instance, you may wish 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 are appended to a CREATE TABLE or CREATE INDEX statement in SQL. The configuration strings specify storage parameters that must be considered valid by the Informix server.

Selecting the configuration string

The choice of configuration strings by an ArcSDE application depends upon the operation being performed and the type of object it is being performed on 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 string whose configuration keyword matches the given configuration keyword and whose parameter_name matches the chosen name.

When running the **sdetable** and certain other ArcSDE administration commands, you may specify your own configuration keyword. When running ArcSDE applications, the configuration keyword is specified to the ArcSDE server automatically.

If the application cannot find the requested configuration string within the specified parameter group, it searches the DEFAULT parameter group. If the requested configuration string cannot be located within the DEFAULT parameter group, the

ArcSDE use no configuration string, and the CREATE TABLE or CREATE INDEX statement picks up the defaults according to the rules of the Informix server.

Table parameters

Table parameters define the storage configuration of an Informix table. The table parameter is appended to an Informix CREATE TABLE statement during its creation by ArcSDE. Valid entries into an ArcSDE table parameter include all Informix CREATE TABLE statement parameters to the right of the statement's columns list.

For example, a business table created with the following Informix CREATE TABLE statement:

```
database cntry94;

--Create table cntry94
create table cntry94
(
area            decimal(15,3),
name            varchar(40,0),
abbrevname      varchar(12,0),
fips_code       varchar(2,0),
wb_cntry        varchar(3,0),
feature         st_multipolygon,
se_row_id       integer,
primary key (se_row_id) constraint sde.sp_ref_pk
put feature in (small_business_table),
extend size 16 next size 16
lock mode row
);
```

would be entered into a dbtune file B_STORAGE table parameter with the following configuration string:

```
B_STORAGE " IN SMALL_BUSINESS_TABLE EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW "
```

An example using Table Fragmentation:

```
create table cntry94
(
area            decimal(15,3),
name            varchar(40,0),
abbrevname      varchar(12,0),
```

```
fips_code       varchar(2,0),
wb_cntry        varchar(3,0),
feature         st_multipolygon,
objectid      integer,
primary key (se_row_id) constraint sde.sp_ref_pk

)
fragment by expression objectid <= 100 IN dbspace1, objectid > 100 and objectid <200
IN dbspace2, objectid >= 200 IN dbspace3 extent size 16 next size 16
lock mode row;
```

would be entered into a dbtune file B_STORAGE table parameter with the following configuration string:

```
B_STORAGE " FRAGMENT BY EXPRESSION objected<=100 IN dbspace1, objected
>100 and objected<200 IN dbspace2, objected >= 200 in dbspace3 EXTENT SIZE
16 NEXT SIZE 16 LOCK MODE ROW "
```

Index parameters

Index parameters define the storage configuration of an Informix index. The index parameter is appended to an Informix CREATE INDEX statement during its creation by ArcSDE. Valid entries into an ArcSDE index parameter include all Informix CREATE INDEX statement parameters to the right of the statement's column list.

For example, an index created with the following Informix CREATE INDEX statement:

```
CREATE INDEX cntry94_idx on cntry94
fillfactor 90,
in business_index;
```

would be entered into a dbtune B_INDEX_USER storage parameter with the following configuration string:

```
B_INDEX_USER "FILLFACTOR 90 IN BUSINESS_INDEX"
```

Defining the storage parameters

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

Meta parameters

Meta parameters define such things as the way certain types of data will be stored, the environment of a keyword, or a comment that describes something about the keyword such as what it should be used for.

The business table storage parameter

A business table is any Informix 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 parameters exist to support the creation of business table indexes.

The B_INDEX_USER 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 parameter holds the storage configuration of the index ArcSDE creates on the register table's object ID column, commonly referred to as the ROWID. A registered table can be created with the alter_reg operation of the sdetable command or from the ArcCatalog interface.

The B_RTEE parameter holds the storage configuration of the spatial column index 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 ArcInfo™ when it creates a feature class and by the add operation of the sdelayer command.

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

Multiversioned table storage parameters

Registering a business table or feature class as multiversioned allows multiple users to maintain and edit their copy of the object. At appropriate intervals, each user merges the changes they have made to their copy with the changes made by other users and reconciles any conflicts that arise when the same features are modified.

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

The A_STORAGE storage parameter maintains the storage configuration of the adds table. Four other 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 multiversioned 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 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_RTREE 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 its index are duplicated in the adds 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 storage parameter holds the storage configuration of the deletes table. Two other 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 parameters of the configuration keyword that the business table was created with.

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 storage parameter holds the Informix CREATE TABLE storage configuration of the RAS table.

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

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

The BND_INDEX_COMPOSITE storage parameter holds the Informix CREATE INDEX storage configuration of the BND table's composite column index.

The BND_INDEX_ID storage parameter holds the Informix CREATE INDEX storage configuration of the BND table's rid column index.

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

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

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

The BLK_INDEX_COMPOSITE storage parameter holds the Informix 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

- TOPOLOGY
- IMS_METADATARELATIONSHIPS
- IMS_METADATA
- IMS_METADATATAGS
- IMS_METADATATHUMBNAILS
- IMS_METADATAUSERS
- IMS_METADATAVALUES
- IMS_METADATAWORDINDEX
- IMS_METADATAWORD

DEFAULTS keyword

Each dbtune table has a fully populated DEFAULTS keyword. The DEFAULTS keyword can be selected whenever you create a table, index, feature class, or raster column. If you do not select a keyword for one of these objects, the DEFAULTS keyword is used. If you do not include a parameter in a keyword you have defined, ArcSDE substitutes the parameter from the DEFAULTS keyword.

The DEFAULTS keyword relieves you of the need to define all the parameters for each of the keywords you define. The parameters of the DEFAULTS 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 keyword parameter in the dbtune.sde file, it automatically adds the parameter. If you import a dbtune file with the sdedbtune command, it will automatically add any parameters that are missing. ArcSDE will detect the presence of the following list of parameters and insert the parameter and the default configuration string.

```
##DEFAULTS
UI_TEXT                    "DEFAULTS"
B_STORAGE                  "EXTENT SIZE 16 NEXT SIZE 16 LOCK M
B_INDEX_ROWID              "FILLFACTOR 90"
B_INDEX_USER               "FILLFACTOR 90"
B_RTREE                    ""
A_STORAGE                  "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
A_INDEX_ROWID              "FILLFACTOR 90"
A_INDEX_STATEID            "FILLFACTOR 90"
A_INDEX_USER               "FILLFACTOR 90"
A_RTREE                    ""
D_STORAGE                  "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
D_INDEX_DELETED_AT         "FILLFACTOR"
D_INDEX_STATE_ROWID        "FILLFACTOR"
S_STORAGE                  ""
RAS_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
```

```
RAS_INDEX_ID                "FILLFACTOR 90"
BND_STORAGE                 "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
BND_INDEX_COMPOSITE         "FILLFACTOR 90"
BND_INDEX_ID                "FILLFACTOR 90"
AUX_STORAGE                 "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
AUX_INDEX_COMPOSITE         "FILLFACTOR 90"
BLK_STORAGE                 "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
BLK_INDEX_COMPOSITE         "FILLFACTOR 90"
USE_EXCLUSIVE_LOCKING       "TRUE"

END
```

NOTE: S_STORAGE was added to ArcSDE 9 to represent the "smart blob sbspace" and must be set in order for B_RTREE to work successfully. S_STORAGE is used to store the spatial feature shape data. The S_STORAGE is the equivalent of the "put feature in <smart large object sbspace>" from the SDE 3.x dbtune.sde file. If S_STORAGE is not set, then spatial feature data will be put in the default sbspace.

During Compression of large databases, Informix Database frequently runs out of the available locks. In order to prevent this, tables are locked in exclusive mode during compress. With ArcSDE 9 Compress could be run with existing user connections. In order to be able to compress with existing user connections, the exclusive locking during compress needs to be disabled. The USE_EXCLUSIVE_LOCKING parameter can be set to false to allow compress with concurrent user connections.

The SDE 3.x A_SBLOB_DBS parameter is not automatically converted to the S_STORAGE parameter. You need to manually convert the A_SBLOB_DBS parameter into the new S_STORAGE parameter; otherwise, the default smart large object sbspace is used. All other SDE 3.x parameters convert automatically to ArcSDE 9 parameters using the "`sdedbtune -o import`" command.

Setting the system table DATA_DICTIONARY keyword

During the execution of the install operation of the sdesetupinfx administration tool, the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY keyword. You may customize the keyword in the dbtune.sde file (found in the $SDEHOME/etc on UNIX systems or %SDEHOME%\etc on Windows 2000 systems) prior to running the sdesetupinfx tool. In this way you can change default storage parameters of the DATA_DICTIONARY keyword.

Edits to all of 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—directly 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 highly active. Separating these objects into their own tablespaces allows you to position their data files with data files that experience low I/O activity and thus avoid as much disk I/O contention as possible.

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

```
##DATA_DICTIONARY
UI_TEXT                     "DATA_DICTIONARY"
B_STORAGE                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"
STATES_TABLE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
STATES_INDEX                "FILLFACTOR 90"
STATE_LINEAGES_TABLE        "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
VERSIONS_TABLE              "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
VERSIONS_INDEX              "FILLFACTOR 90"
MVTABLES_MODIFIED_TABLE     "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
MVTABLES_MODIFIED_INDEX     "FILLFACTOR 90"
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 get 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 Informix, the default values for TOPOLOGY and TOPOLOGY::DIRTYAREAS are

```
##TOPOLOGY_DEFAULTS
```

```
B_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID   "FILLFACTOR 90"
B_INDEX_USER    "FILLFACTOR 90"
A_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
UI_TOPOLOGY_TEXT        "The topology default configuration"
A_INDEX_ROWID   "FILLFACTOR 90"
D_INDEX_DELETED_AT      "FILLFACTOR 90"
A_INDEX_STATEID "FILLFACTOR 90"
A_INDEX_USER    "FILLFACTOR 90"
D_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
D_INDEX_STATE_ROWID     "FILLFACTOR 90"
A_RTREE ""
B_RTREE ""
END

##TOPOLOGY_DEFAULTS::DIRTYAREAS
B_RTREE ""
A_RTREE ""
B_INDEX_ROWID   "FILLFACTOR 90"
B_INDEX_USER    "FILLFACTOR 90"
A_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
A_INDEX_ROWID   "FILLFACTOR 90"
D_INDEX_DELETED_AT      "FILLFACTOR 90"
A_INDEX_STATEID "FILLFACTOR 90"
A_INDEX_USER    "FILLFACTOR 90"
D_STORAGE       "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
D_INDEX_STATE_ROWID     "FILLFACTOR 90"
END
```

The IMS METADATA keywords

The IMS METADATA keywords control the storage of the IMS Metadata tables. These keywords are a standard part of the dbtune table. If the keywords are not present in the dbtune file when it is imported into the DBTUNE table, ArcSDE applies software defaults. The software defaults have the same settings as the keyword parameters listed in the dbtune.sde table that is shipped with ArcSDE. The Informix parameter settings such as the initial and next should be sufficient. However, you will need to edit the storage parameters tablespace names. As always, try to separate the tables and indexes into different tablespaces.

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

The IMS metadata keywords are as follows:

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

```
##IMS_METADATA

B_STORAGE                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"
B_RTREE                     ""
S_STORAGE                   ""
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 following 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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_USER                "FILLFACTOR 90"

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                   "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90"
B_INDEX_USER                "FILLFACTOR 90"

END
```

Note: These keywords are obsolete. ArcSDE XML columns are not supported with Informix databases. Therefore, ArcIMS Metadata Services are no longer supported with Informix databases. These keywords will not be used by ArcSDE or ArcIMS Metadata Services.

Changing the appearance of DBTUNE keywords in the ArcInfo user interface

ArcSDE 9 introduces two new parameters that will support the ArcInfo user interface UI_TEXT and UI_NETWORK_TEXT. ArcSDE administrators can add one of these parameters to each keyword to communicate to the ArcInfo schema builders the intended use of the keyword. The configuration string of these parameters will appear in ArcInfo interface dbtune keyword scrolling lists.

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

The UI_NETWORK_TEXT parameter should be added to parent network keywords.

Adding a comment to a configuration keyword

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

LOGFILE keywords

Log files are used by ArcSDE to maintain temporary and persistent sets of selected records. Whenever a user connects to ArcSDE for the first time, the SDE_LOGFILES and SDE_LOGFILE_DATA tables and indexes are created.

You may create a keyword for each user that begins with the LOGFILE_<username>. For example, if the user's name is STANLEY, ArcSDE will search the dbtune table for the LOGFILE_STANLEY keyword. If this keyword is not found, ArcSDE will use the parameters of the LOGFILE_DEFAULTS keyword to create the SDE_LOGFILES and SDE_LOGFILE_DATA tables.

ArcSDE always creates the DBTUNE table with a LOGFILE_DEFAULTS keyword. If you do not specify this keyword in the dbtune file that you import with the sdedbtune command, ArcSDE will populate the dbtune table with default LOGFILE_DEFAULTS parameters. Further, if the dbtune file contains some of the LOGFILE_DEFAULTS keyword parameters, ArcSDE will supply the rest. Therefore, the LOGFILE_DEFAULTS keyword is always fully populated.

If a user-specific keyword exists, but some of the parameters are not present, the parameters of the LOGFILE_DEFAULTS keyword are used. If some of the parameters are not set in either a user-specific keyword or the LOGFILE_DEFAULTS keyword, the Informix defaults are used.

Creating a log file keyword for each user allows you to separate the log files onto different 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 parameters supplied with the installed dbtune.sde file. However, for applications that make heavy use of 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 the imported dbtune file does not contain a LOGFILE_DEFAULTS keyword, or if any of the logfile parameters are missing, ArcSDE will insert the following records:

```
##LOGFILE_DEFAULTS

UI_TEXT                 "LOGFILE_DEFAULTS"
LF_STORAGE              "EXTENT SIZE 16 NEXT SIZE 16 LOC"
LF_INDEXES              "FILLFACTOR 90"
LD_STORAGE              "EXTENT SIZE 16 NEXT SIZE 16 LOC"
LD_INDEX_DATA_ID        "FILLFACTOR 90"
LD_INDEX_ROWID          "FILLFACTOR 90"

END
```

The LD_STORAGE and LF_STORAGE parameters that control the storage of the SDE_LOGFILE_DATA and SDE_LOGFILES tables by default are generated with Informix logging turned on, the Informix 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.

Network class composite configuration keywords

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

To accommodate the vast difference in size of the network tables, the network composite keyword is subdivided into elements. A network composite keyword has three elements: the parent element defines the general characteristic of the 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 keyword looks like any other keyword. The description element is demarcated by the addition of the ::DESC suffix to

the parent element's keyword, and the network element is demarcated by the addition of the ::NETWORK suffix to the parent element's keyword.

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

```
##ELECTRIC
A_RTREE     ""
B_STORAGE                   "IN BUSINESS EXTENT SIZE 16 NEXT SIZE 16
LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90 IN BUSINESS_INDEX"
B_INDEX_USER                "FILLFACTOR 90 IN BUSINESS_INDEX"
B_RTREE                     "IN BUSINESS_INDEX"
D_STORAGE                   "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
A_STORAGE                   "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
D_INDEX_STATE_ROWID         "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_ROWID               "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_STATEID             "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_USER                "FILLFACTOR 90 IN DELTA_INDEX"
UI_NETWORK_TEXT             "The electrical geometrical network class keyword"
D_INDEX_DELETED_AT          "FILLFACTOR 90 IN DELTA_INDEX"
COMMENT                     "This keyword is dedicated to the electrical geometric
network class "
END

##ELECTRIC::DESC
B_STORAGE                   "IN BUSINESS EXTENT SIZE 16 NEXT SIZE 16
LOCK MODE ROW"
A_STORAGE                   "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
B_INDEX_ROWID               "FILLFACTOR 90 IN BUSINESS_INDEX"
A_INDEX_ROWID               "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_STATEID             "FILLFACTOR 90 INDELTA_INDEX"
A_INDEX_USER                "FILLFACTOR 90 IN DELTA_INDEX"
D_STORAGE                   "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
D_INDEX_STATE_ROWID         "FILLFACTOR 90 IN DELTA_INDEX"
D_INDEX_DELETED_AT          "FILLFACTOR 90 IN DELTA_INDEX"
B_INDEX_USER                "FILLFACTOR 90 IN BUSINESS_INDEX"
END

##ELECTRIC::NETWORK
```

```
A_INDEX_ROWID            "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_STATEID          "FILLFACTOR 90 IN DELTA_INDEX"
A_INDEX_USER             "FILLFACTOR 90 IN DELTA_INDEX"
D_STORAGE                "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
D_INDEX_STATE_ROWID      "FILLFACTOR 90 IN DELTA_INDEX"
B_STORAGE                "IN BUSINESS EXTENT SIZE 16 NEXT SIZE 16
LOCK MODE ROW"
D_INDEX_DELETED_AT       "FILLFACTOR 90 IN DELTA_INDEX"
B_INDEX_ROWID            "FILLFACTOR 90 IN BUSINESS_INDEX"
B_INDEX_USER             "FILLFACTOR 90 IN BUSINESS INDEX"
A_STORAGE                "IN DELTA_TABLE EXTENT SIZE 16 NEXT SIZE
16 LOCK MODE ROW"
END
```

Following the import of the dbtune file, these records would be inserted into the DBTUNE table. You can use DBACCESS to see the following information:

```
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_USER
ELECTRIC            A_STORAGE
ELECTRIC            A_INDEX_ROWID
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 keyword parameters, the network descriptions table is created with the parameters of the ELECTRIC::DESC 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 parameters for the ArcGIS network class. If the user does not select a network class composite keyword from the ArcCatalog interface, the ArcGIS network is created with the parameters within the NETWORK_DEFAULTS configuration keyword.

Whenever a network class composite keyword is selected, its parameters are used to create the feature class, table, and indexes of the network class. If a network composite keyword is missing any parameters, ArcGIS substitutes the parameters of the DEFAULTS keyword rather than the NETWORK_DEFAULTS keyword. So the parameters of the NETWORK_DEFAULTS keyword are only used in the event that no network composite keyword is selected.

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

```
##NETWORK_DEFAULTS
A_RTREE                 ""
B_STORAGE               "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
B_INDEX_ROWID           "FILLFACTOR 90"
B_INDEX_USER            "FILLFACTOR 90"
B_RTREE                 "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
A_STORAGE               "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
D_INDEX_STATE_ROWID     "FILLFACTOR 90"
A_INDEX_ROWID           "FILLFACTOR 90"
A_INDEX_STATEID         "FILLFACTOR 90"
A_INDEX_USER            "FILLFACTOR 90"
UI_NETWORK_TEXT         "The network default configuration"
D_INDEX_DELETED_AT      "FILLFACTOR 90"
COMMENT                 "The base system initialization parameters for
NETWORK_DEFAULTS"
END
```

```
##NETWORK_DEFAULTS::DESC
B_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
A_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
B_INDEX_ROWID            "FILLFACTOR 90"
A_INDEX_ROWID            "FILLFACTOR 90"
A_INDEX_STATEID          "FILLFACTOR 90"
A_INDEX_USER             "FILLFACTOR 90"
D_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
D_INDEX_STATE_ROWID      "FILLFACTOR 90"
D_INDEX_DELETED_AT       "FILLFACTOR 90"
B_INDEX_USER             "FILLFACTOR 90"
END

##NETWORK_DEFAULTS::NETWORK

A_INDEX_ROWID            "FILLFACTOR 90"
A_INDEX_STATEID          "FILLFACTOR 90"
A_INDEX_USER             "FILLFACTOR 90"
D_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
D_INDEX_STATE_ROWID      "FILLFACTOR 90"
B_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
D_INDEX_DELETED_AT       "FILLFACTOR 90"
B_INDEX_ROWID            "FILLFACTOR 90"
B_INDEX_USER             "FILLFACTOR 90"
A_STORAGE                "EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE
ROW"
END
```

Informix default parameters

By default, Informix stores tables and indexes in the ArcSDE database dbspace using the tablespace's default storage parameters. Determine the ArcSDE database dbspace by querying:

```
C:\Informix> onstat –d
```

Editing the storage parameters

To edit the storage parameters, the sdedbtune administration command allows you to export the DBTUNE table to a file located in the $SDEHOME/etc directory on UNIX servers and in the %SDEHOME%\etc folder on Windows servers. It is an ArcSDE configuration file that contains the Informix table and index creation parameters. These parameters allow the ArcSDE service to communicate to the Informix server such things as:

- Which dbspace a table or index will be created in
- The size of its initial and next extent
- Other parameters that can be set on either the CREATE TABLE or CREATE INDEX statement

Converting SDE 3.x storage parameters to ArcSDE 9 storage parameters

In SDE 3.x, the dbtune storage parameters were maintained in the dbtune.sde file. The storage parameters of these previous versions mapped directly to Informix storage parameters.

The ArcSDE 9 storage parameters hold entire configuration strings of the table or index they represent.

The conversion of SDE 3.x storage parameters to ArcSDE 9 occurs automatically when the ArcSDE 9 sdesetupinfx utility reads the parameters from the SDE 3.x dbtune.sde file. The import operation of the sdedbtune command also converts an SDE 3.x dbtune file into ArcSDE 9 storage parameters before it writes them to the dbtune table. To see the results you can either use DBACCESS to list the parameters of the dbtune table or write the parameters of the dbtune table to another file using the export operation of the sdedbtune command.

The following table lists the conversion of SDE 3.x storage parameters to ArcSDE 9 storage parameters.

The SDE 3.x business table and index parameter prefix is "A_". The ArcSDE 8.1.2 business table and index parameter prefix is "B_". The SDE 3.x business table storage parameters are converted to the single ArcSDE 8.1.2 B_STORAGE storage parameter. The B_STORAGE parameter holds the entire business table's configuration string.

ArcSDE 3.x storage parameters	ArcSDE 8.1.2 storage parameters
A_TBLSP roads A_INIT 16 A_NEXT 6 A_LOCK_ROW 1	B_STORAGE "In roads EXTENT SIZE 16 NEXT SIZE 16 LOCK MODE ROW"

The SDE 3.x business table index storage parameters are converted to ArcSDE 9.x storage parameter configuration strings. The example below illustrates how the SDE 3.x storage parameters are converted into the ArcSDE 9.x spatial column index storage parameter B_RTREE. The other ArcSDE 9.x business table index storage parameters, B_INDEX_ROWID and B_INDEX_USER, are also constructed this way.

ArcSDE 3.x storage parameters	ArcSDE 9.x storage parameters
INDEX_TABLESPACE roads_index	B_RTREE
A_IX_FILL 90 A_IX_TBLSP roads_index	"FILLFACTOR 90 IN roads_index"
A_SBLOB_DBS roads_sblob	S_STORAGE "roads_sblob"

The complete list of ArcSDE 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

Parameter Name	Value	Parameter Description	Default Value
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
B_STORAGE	<string>	Business table	Informix defaults
B_INDEX_ROWID	<string>	Business table object ID column index	Informix defaults
B_INDEX_USER	<string>	Business table user index(s)	Informix defaults
A_STORAGE	<string>	Adds table	Informix defaults
A_INDEX_ROWID	<string>	Adds table object ID column index	Informix defaults
A_INDEX_STATEID	<string>	Adds table sde_state_id column index	Informix defaults
A_INDEX_USER	<string>	Adds table index	Informix defaults
D_STORAGE	<string>	Deletes table	Informix defaults
D_INDEX_ STATE_ROWID	<string>	Deletes table sde_states_id and sde_deletes_row_id column index	Informix defaults
S_STORAGE	<string>	Represents a “smart blob sbspace”	Informix defaults
D_INDEX_DELETED_AT	<string>	Deletes table sde_deleted_at column index	Informix defaults
LF_STORAGE	<string>	Sde_logfiles table	Informix defaults
LF_INDEXES	<string>	Sde_logfile table column indexes	Informix defaults

Parameter Name	Value	Parameter Description	Default Value
LD_STORAGE	<string>	Sde_logfile_data table	Informix defaults
LD_INDEX_DATA_ID	<string>	Sde_logfile_data table	Informix defaults
LD_INDEX_ROWID	<string>	Sde_logfile_data table sde_row_id column index	Informix defaults
RAS_STORAGE	<string>	Raster RAS table	Informix defaults
RAS_INDEX_ID	<string>	Raster RAS table RID index	Informix defaults
BND_STORAGE	<string>	Raster BND table	Informix defaults
BND_INDEX_COMPOSITE	<string>	Raster BND table composite column index	Informix defaults
BND_INDEX_ID	<string>	Raster BND table RID column index	Informix defaults
AUX_STORAGE	<string>	Raster AUX table	Informix defaults
AUX_INDEX_COMPOSITE	<string>	Raster AUX table composite column index	Informix defaults
BLK_STORAGE	<string>	Raster BLK table	Informix defaults
USE_EXCLUSIVE_LOCKING	<string>	Use Exclusive Locking during ArcSDE Compress	TRUE
BLK_INDEX_COMPOSITE	<string>	Raster BLK table composite column index	Informix defaults

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.

ArcSDE to Informix Data Type Mapping

ArcSDE uses 12 general data types. These types are mapped to Informix types in the following matrix. Precision refers to the number of digits in a value, while scale refers to the number of digits to the right of the decimal separator.

ArcSDE Data Type	Informix Data type
SE_STRING_TYPE	Char, varchar
SE_NSTRING_TYPE	Nchar, nvarchar
SE_NCLOB_TYPE	Clob
SE_INT16_TYPE (SE_SMALLINT_TYPE)	Smallint
SE_INT32_TYPE (SE_INTEGER_TYPE)	Integer
SE_INT64_TYPE	Int8

ArcSDE Data Type	Informix Data type
SE_FLOAT32_TYPE (SE_FLOAT_TYPE)	Decimal (precision < 7, scale > 0)
SE_FLOAT64_TYPE (SE_DOUBLE_TYPE)	Float
SE_DATE_TYPE	Datetime
SE_UUID_TYPE	Char
SE_BLOB_TYPE	Blob

Data creation

There are numerous applications that can create and load data within an ArcSDE Informix 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 rasters 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 Version 2.1c 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 graphical 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*
- *Using ArcToolbox*
- *Building a Geodatabase*

Creating and populating a feature class

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

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 (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 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 contain 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 the user beetle.

The same table could be created with a SQL CREATE TABLE statement using the Informix dbaccess 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
```

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 I/O 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. However, the grid size can be altered with the -o alter operation. The alter operation will apply an exclusive lock on the feature class, preventing all modifications by ArcInfo until the operation is complete.

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. A more "from scratch" method would be to add the data with an editor such as ArcMap.

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 I/O 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
```

Now your feature class is ready for use by ArcSDE client applications.

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. ArcMap requires data to be multiversioned to edit it. 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.

```
sdelayer -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 called beetle gives a user called spider SELECT privileges on a feature class called states.

```
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 data.

DELETE. The user may delete the selected object data.

UPDATE. The user may modify the selected object data.

INSERT. The user may add new data to the selected object 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 "from scratch" method of creating a schema and then 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 process is 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

The shp2sde command 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 wish to adopt for the data storage by including a "–k" option that references to a configuration keyword containing storage parameters for the business table and indexes of the feature class.

```
shp2sde -o create -f rdshp -l roads,shape -k GEOMETRY_TYPE -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 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 1+ -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 an -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 Informix 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 Informix 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 Informix, 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 Informix 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 client applications that export data from ArcSDE as well. With ArcSDE, the following command line tools exist:

- sdeexport—Creates an ArcSDE export file to easily move feature class data between Informix 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 either shared, session-based or stand-alone. 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 stand-alone log files and all available log files of this type is exhausted, ArcSDE will attempt to create a session-based log file if they are allowed; otherwise a shared log file is created. If the shared log file cannot be created, ArcSDE returns an error.

Shared ArcSDE logfiles

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 remain in 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 sessions 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.

Stand-alone ArcSDE logfiles

Stand-alone log files are created by a session for each log file the application needs to store. When an application deletes the log file, the stand-alone log file is truncated. The stand-alone log files are dropped when the session disconnects. To configure your server to use stand-alone log files, set the server configuration parameter MAXSTANDALONELOGS to the number of stand alone 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 6 stand-alone 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 stand-alone 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 logfiles

The sde user can create a pool of log files that can be checked out and used as either session-based or stand-alone log files by other users. Using a pool of sde owned log files avoids having to grant users CREATE TABLE privileges. Shared log files cannot be checked out from an sde owned 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 stand-alone log files if allowed. To calculate the total number of log files that could be checked out of the pool, use the following formulae:

If session log files are allowed, but not stand-alone log files:

```
LOGPOOLSIZE = total sessions expected
```

If stand-alone log files are allowed, but not session log files:

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

If both stand-alone 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 users 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 in the case of a session log file or whenever the log file occupying a stand alone 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, ArcCatalog, and ArcToolbox 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, we provide a glimpse of how to use ArcToolbox and ArcCatalog to load data. Please refer to the ArcInfo documentation on ArcCatalog, ArcToolbox, and the geodatabase for a full discussion of these tools.

Loading data

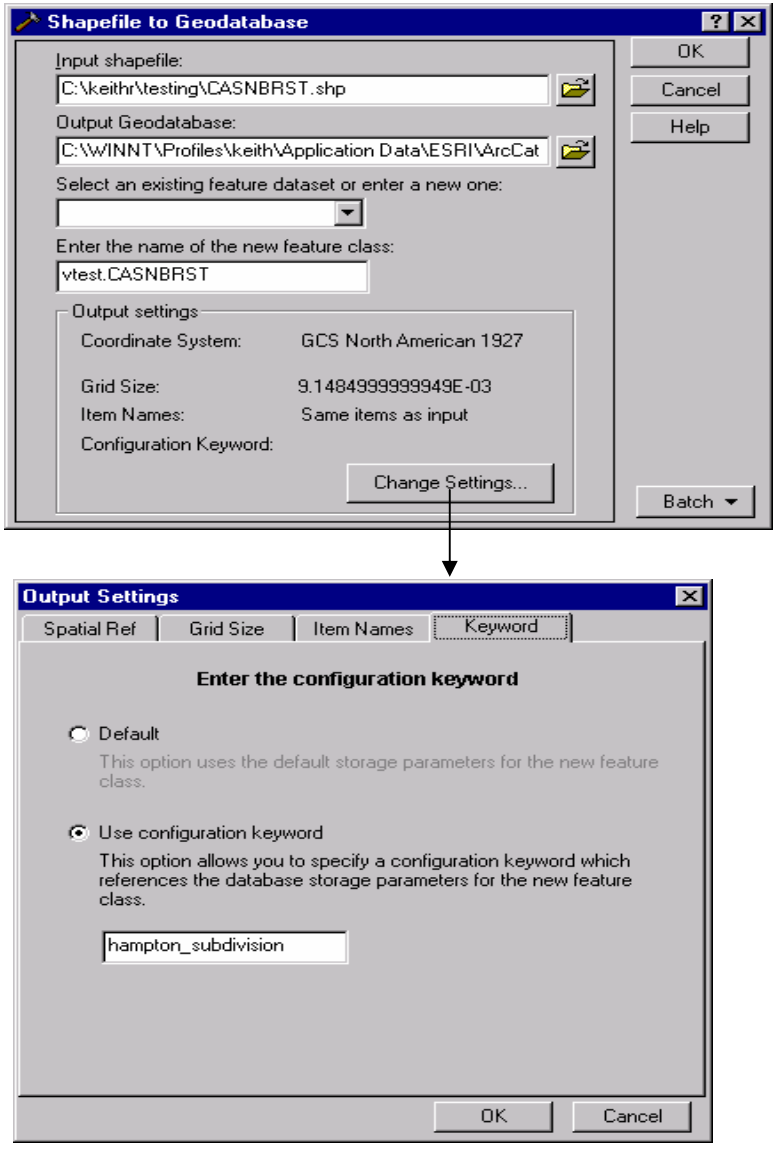
You can convert ESRI shapefiles, coverages, Map LIBRARIAN layers, and ArcStorm layers into geodatabase feature classes with the ArcToolbox and ArcCatalog applications. ArcToolbox provides a number of tools that enable you to convert data from one format to another.

ArcToolbox operations, such as the ArcSDE administration commands shp2sde, cov2sde, and sdeimport, accept configuration keywords.

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.

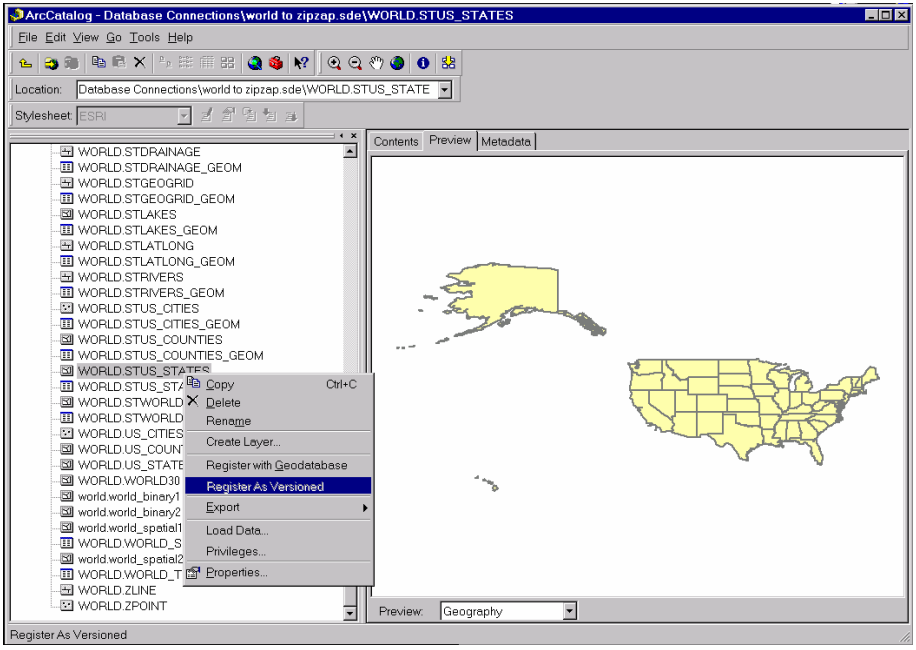
The configuration keyword contains storage parameters that list the Informix storage parameters that ArcSDE creates the feature class business table and indexes with.

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



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 select the Register As Versioned context menu item.



A feature class is registered as multiversioned from within ArcCatalog.

Compressing the geodatabase

When the multiversioned tables of a geodatabase have been edited over an extended period of time, and the number of states and rows in the delta tables has grown significantly, performance can be improved by running the compress database command. It is good to compress your database as often as possible.

The compress command removes the states that are no longer referenced by a version and moves the rows in the delta tables, which are common to all versions, to the business table. To achieve the maximum benefit when you run the compress command, first reconcile, post, and delete each version with the DEFAULT version. Sometimes this may not be a reasonable option based on your organization's work flow. At a minimum, to improve performance, simply reconcile each version with the DEFAULT version and save, then perform the compress. This will ensure that all the edits in the DEFAULT version will be compressed from the delta tables to the business table. The compress

command can be executed without first reconciling, posting, and deleting each version, but the performance benefits may not be as noticeable.

You can perform a compress of the database by using either the ArcCatalog or ArcSDE command line.

To perform a compress of the database, you must start ArcCatalog and add the compress database tool. To add the tool, right-click the gray area of the toolbar and select Customize.

From the Customize menu, choose the Command tab and select Geodatabase tools. Select Compress Database Command and drag it to the toolbar.

To use the Compress Database tool, connect to your ArcSDE service as the sde user. Click the sde connection, click the compress tool, and answer yes to the popup window asking if you are sure you want to compress the database.

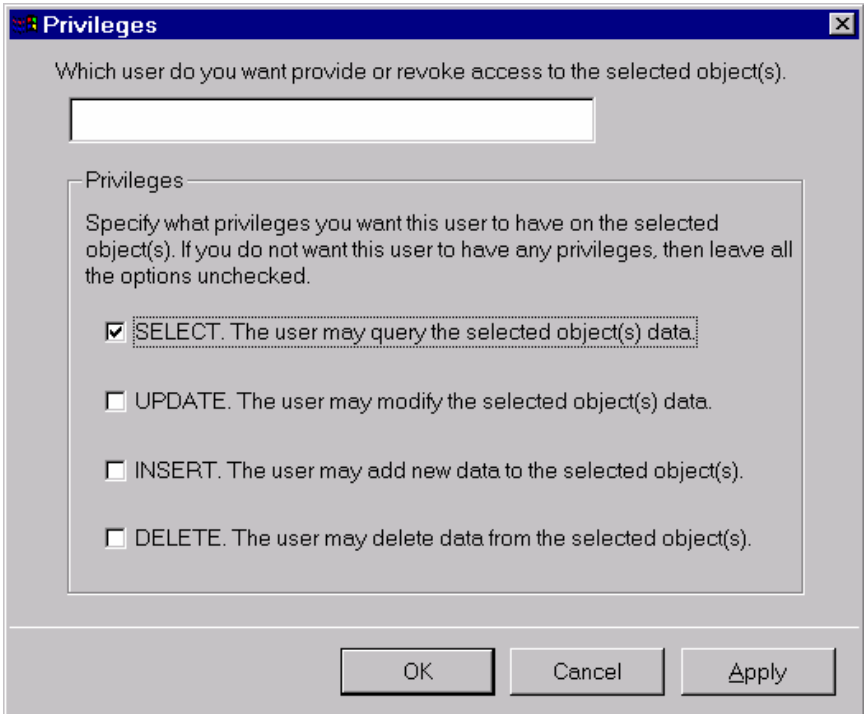
To compress the database from the command line, use sdeversion from either the DOS or UNIX prompt.

```
sdeversion -o compress      [-N]
                            -u <DB_user_name> [-p <DB_User_password>] [-q]
                            [-i <service>] [-s <server_name>] [-D <database>]
```

For more information on the sdeversion command, refer to ArcSDE Developer Help.

Granting privileges

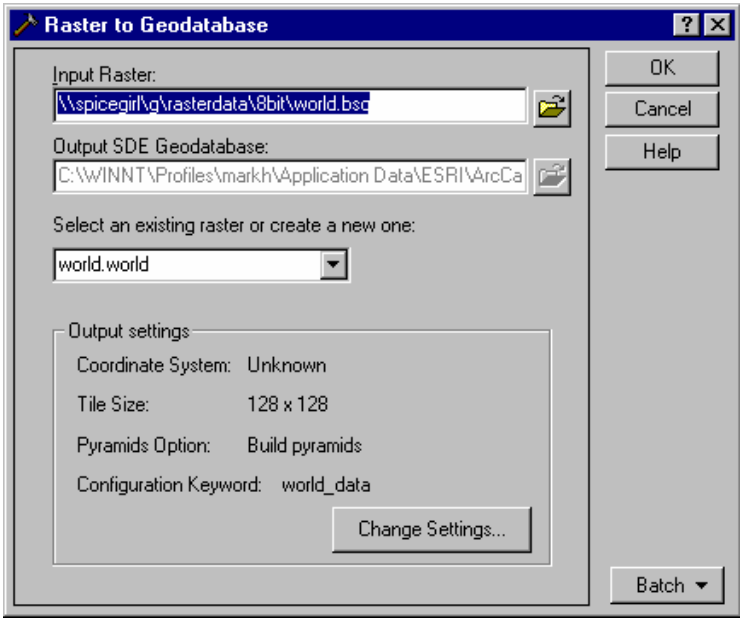
Using ArcCatalog, right-click the data object class and click the Privileges context menu. From the Privileges context menu assign privileges specifying the username and the privilege you wish to grant to or revoke from a particular user.



The ArcCatalog Privileges menu 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 Informix database.



Registering a business table

When ArcSDE registers a business table, it performs a number of tasks depending on the type of registration that was 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.

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.

Registration of these two registration types happens rather quickly.

Registering a table as SDE maintained

Tables registered as type SDE must have a row ID column that uniquely identifies the rows of the table.

NOTE: Tables registered by the geodatabase must be registered to ArcSDE as SDE maintained. If the geodatabase determined that the table has already been registered by ArcSDE as SDE maintained, the geodatabase uses the SDE maintained row ID. In this case 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 SDE maintained. 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 called 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.

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 is not, the registration fails.

Next, ArcSDE confirms that the column has a unique index. If the column was defined with a non-unique 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 then 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 non-unique values is an expensive process, especially if the table contains more than a 100,000 records.

Therefore if at all possible you should not rely on ArcSDE to perform this operation. You 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 insure 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. When tables are registered as multiversioned the associated adds and deletes table are created and analyzed. These tables as their name implies, store the records of the business that are added and deleted. They are named A<registration_id> and D<registration_id>.

Multiversioned views are available for SQL access to the multiversioned database. See the *ArcSDE Developers Guide* for more information.

How does ArcSDE use existing Informix tables?

Tables with spatial columns can be created by other applications. ArcSDE has been designed to use tables containing spatial 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

Users can use the ArcSDE administration command *sdelayer –o register* to manually register a table as a layer. Registering tables as layers manually gives users more control over how a table is registered. Here is an example of registering a table called TBL containing point geometries (-e p) in a spatial column called SHAPE. The table has an integer column called PID that will be used as a user-maintained unique feature identifier column (-C pid,USER).

sdelayer -o register -l tbl,shape -e p -C pid,USER -u <user> -p <pw>

Autoregistration

Autoregistration is controlled by the system configuration parameter DISABLEAUTOREG, which is set to TRUE by default. To turn on autoregistration, use the following administration command:

```
sdeconfig -o alter -v DISABLEAUTOREG=FALSE -u sde -p <sde_password>
```

Whenever an ArcSDE client lists the feature classes stored in the database, ArcSDE automatically searches the Informix system tables for new tables with spatial columns. When a new table is discovered, it is registered with ArcSDE and made available to applications.

ArcSDE uses the first record in a newly discovered table to establish the ArcSDE geometry type. If the table contains multiple geometry types, the sdelayer administration utility can be used to alter the geometry type definition.

ArcSDE searches for a column in the table to use as an OBJECTID column. To qualify, the column must be defined as INTEGER, NOT NULL, and UNIQUE constraints. If such a column is found, it is recorded in the ArcSDE table registry along with the table. If an OBJECTID column is not found, the table is registered, but operations requiring an OBJECTID are unavailable.

CHAPTER 5

National language support

Configuring ArcSDE for Informix to use a specific locale begins with the Informix database. The database must be created with the correct locale before data can be stored in it. The next step is to configure the Open Database Connectivity (ODBC) data source names (DSN) to transfer the data in the proper locale and make the correct codepage conversions that may be required when the operating systems of the client and server are different. This chapter provides guidelines for configuring ArcSDE with the correct locales. For more information refer to the Informix document *Informix Guide to GLS functionality.*

Creating an Informix database with a specific language locale

An Informix database must be created with a specific locale. If the locale is not set when the database is created, the database locale defaults to US English: en_us.8859-1 for UNIX and en_us:1252 for Windows 2000.

For example, to create a database that will store French characters on a UNIX server, set the DB_LOCALE variable to fr_fr.8859-1 before starting the Informix SQL utility DBACCESS. All database created during this DBACCESS session will be created with the French locale fr_fr.8859-1. To determine what the proper locale is for your database, consult the Informix document *Informix Guide to GLS functionality.*

Setting the NLS_LANG variable on the client

Once the ArcSDE Informix ArcSDE database has been created with the proper character set, data can be loaded into it using a variety of applications such as ArcGIS Desktop and

the ArcSDE administration tools shp2sde and cov2sde. To properly convert and preserve the characters, you must set the Informix NLS_LANG variable in the client applications system environment.

For instance, using ArcToolbox installed on Windows 2000 to convert a coverage containing German attribute data into an Informix database on a UNIX server created with the Western European character set WE8ISO8859P1, you would set the NLS_LANG to GERMAN_GERMANY.WE8ISO8859P1. To set the NLS_LANG variable for ArcToolbox, click Start, Settings, and Control Panel. Double-click on the System icon and select the Environment tab after the System menu appears. Click on the System Variables scrolling list and enter NLS_LANG in the Variable: input line and GERMAN_GERMANY.WE8ISO8859P1 in the Value: input line. Click Set and then OK.

Setting the NLS_LANG variable for Windows 2000 clients

Be careful setting the NLS_LANG on the Windows 2000 platform because there are actually two different codepage environments on this platform. Windows applications such as ArcGIS Desktop run in the Windows American National Standards Institute (ANSI) codepage 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) codepage environment. Some languages require two different NLS_LANG settings for the language character set component for each of these codepage environments.

For instance, in the above example the NLS_LANG variable would be set to GERMAN_GERMANY.WE8PC850 if the data was loaded from the MS–DOS Command Prompt using the ArcSDE administration tool shp2sde. If you use both Windows applications and MS–DOS applications together, then you should set the NLS_LANG variable for MS–DOS applications when you open the MS–DOS Command Prompt using the MS–DOS SET command.

SET NLS_LANG = GERMAN_GERMANY.WE8PC850

To determine if your language requires a separate language character set, consult the *Informix Guide to GLS functionality*.

Configuring the Informix server locale

The server locale identifies the locale that the database server uses for its server-specific files on the server computer. Set the server locale in the environment of the Informix DBA before starting the Informix server.

After the server locale has been set, all error messages and output to the online.log file will be reported in the locale set by the SERVER_LOCALE variable. Set the locale in the

DBA's system environment before starting the Informix server. The SERVER_LOCALE variable does not perform a code page conversion during a SQL session.

Configuring the Informix locale for ArcSDE

The Informix CLI does not obtain locale information from the system environment. The INFORMIX CLI driver obtains the client and database locale information from the ODBC DSN. Add the following parameters to the .odbc.ini for UNIX or the ODBC registry for Windows.

```
CLIENT_LOCALE=
DB_LOCALE=
TRANSLATIONDLL=$(INFORMIXDIR)/lib/esql/igo4a304.so
```

For example:

```
CLIENT_LOCALE=fr_fr.PC-Latin-1
DB_LOCALE=fr_fr.8859-1
TRANSLATIONDLL=/export/home/prods/csdk202/lib/esql/igo4a304.so
```

Setting the locale for ArcSDE

Add the locale information to the sde DSN on the computer that the ArcSDE service is started on. For example, if the sde database is a French Language database on UNIX, add the following locale information to the sde DSN. For the sde DSN, the CLIENT_LOCALE and the DB_LOCALE are always set to the locale the sde database was created with.

```
[sde]
. . .
CLIENT_LOCALE=fr_fr.8859-1
DB_LOCALE=fr_fr.8859-1
TRANSLATIONDLL= $(INFORMIXDIR)/lib/esql/igo4a304.so
```

The locale settings of the other DSNs will depend on the locale of the client. The DB_LOCALE of these DSNs is always set to the locale that the database was created with. The CLIENT_LOCALE, however, is set to the locale that the client will use. For instance, if the INFORMIX server is running on a UNIX host and the database was created with a French locale, the DB_LOCALE would be set to fr_fr.8859-1—the French locale for a database created on a UNIX server. The CLIENT_LOCALE is set to the locale of the client environment. If the client environment is on Windows, the CLIENT_LOCALE would be set to fr_fr.PC-Latin-1.

```
[bladetest]
. . .
CLIENT_LOCALE=fr_fr.PC-Latin-1
DB_LOCALE=fr_fr.8859-1
TRANSLATIONDLL= $(INFORMIXDIR)/lib/esql/igo4a304.so
```

Although the ArcSDE client and INFORMIX database can have a different locale, they must always be of the same basic language. For instance, you cannot create a French database, set the DB_LOCALE to fr_fr.8859-1, then set the CLIENT_LOCALE to a non-French locale.

CHAPTER 6

Backup and recovery

This chapter provides you with some basic backup and recovery guidelines. You should refer to the backup and recovery guidelines presented in the *Archive and Backup for Informix Dynamic Server.2000* and *Informix Backup and Restore Guide.*

Data recovery system

The central concepts of a data recovery system for Informix Dynamic Server 2000 can be explained by answering the following questions:

- What is a Dynamic Server recovery system?
- What is an archive?
- What is a logical log backup?
- What is a restore?

Informix provides two recovery systems for Dynamic Server: the ON-Archive system and the ontape utility. You can also use the ON-Bar utility to perform backup and restore operations. Please see the recommended Informix documentation (*Archive and Backup for Informix Dynamic Server.2000* and *Informix Backup and Restore Guide)* for differences between these backup and recovery systems and their usage.

What is a Dynamic Server recovery system?

A Dynamic Server *recovery system* enables you to back up your Dynamic Server data and subsequently restore it in the event that your current data becomes corrupt or inaccessible. The causes of data corruption or loss can range from a program error to

disk failure to a disaster that damages the entire facility. A recovery system enables you to recover data that you already lost due to such mishaps.

What is an archive?

An archive is a copy of either all or some portion of the data that Dynamic Server manages. More precisely, an archive is a copy of one or more Dynamic Server dbspaces (database spaces) and any supporting data that you might need to restore them.

You create an archive of Dynamic Server data on tape or disk that, ideally, you store in a safe location that is separate from your computer facility.

What is a logical-log backup?

A logical log backup is a copy on tape or disk of logical log files that you have made full and eligible for backup. The logical logs files store a record of Dynamic Server activity that occurs between archives.

What is a Dynamic server restore?

A Dynamic Server restore re-creates Dynamic Server data, particularly Dynamic Server dbspaces, from an archive and backed-up logical log files.

Physical and logical restores

You must restore Dynamic Server data in two operations. The first operation is a physical restore and the second, which must follow the first, is a logical restore.

Backing up the database

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.

The following command can be used to back up an ArcSDE database that contains spatial data:

```
dbexport sde -c
```

where sde is the name of the ArcSDE database.

For more information on different Informix database backup and recovery strategies, refer to the *Archive and Backup Guide for Informix Dynamic Server 2000*.

Regardless of which backup and restore mode you are using, you should make regular full backups of your Informix databases. A full backup should include the Informix database and the giomgr.defs, dbinit.sde, and services.sde files. You should also back up any dbtune files you have created and imported into the DBTUNE table.

Recovering the database

For the recovery of an Informix database refer to the *Archive and Backup for Informix Dynamic Server 2000* and *Informix Backup and Restore Guide*. Once the Informix database has been recovered, if necessary, restore the ArcSDE installation from the ArcSDE media and the dbtune files, giomgr.defs, dbinit.sde, and services.sde files from your backup tapes.

The following command can be used to recover your ArcSDE data that contains spatial data:

```
dbimport sde -c -d sdedbs -l buffered
```

where sde is the name of the ArcSDE database and sdedbs is the name of the sbspace being used.

You should test your backup before you need it. If you have just loaded your database, you should do a full backup and then recover the database from tape to make sure 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 Informix tables and indexes created by ArcSDE.

Estimating the size of your spatial tables

The *INFORMIX-Online Dynamic Server Performance Guide* provides precise guidelines for estimating table size. If you require such precision you should consult the 'Table Placement, Layout and Fragmentation' chapter. However, the rough estimates in this document should be within 100 MB for large tables.

This estimation method requires five steps.

1. Estimate the size of the spatial column.
2. Estimate the actual row size of the spatial table.
3. Estimate the metadata space requirements.
4. Estimate the storage space for the spatial table.
5. Estimate the smart large object storage space.

Estimating the size of the spatial column

Estimate the size of the spatial column with:

spatial column size = (average points per feature * coordinate factor) + annotation size

The average points per feature is the sum of all coordinate points required to render the features of a spatial table divided by the number of rows in the table. If you are about to

convert a large number of shapefiles, coverages, or other geographic data into an ArcSDE spatial table, it is unlikely that you can obtain the sum of all coordinates, let alone the number of rows required to perform this calculation. The table below lists reasonable approximations of the average number of points per feature that each data type often has.

The values listed in the table follow this simple logic. Point data type is always defined by a single coordinate.

Linestrings and polygons rendering a dense urban condition tend to have fewer coordinates than a sparse rural condition.

Most linestrings in an urban center have two coordinates. However, curves that define round features such as cul-de-sacs require several more coordinates, so urban linestrings tend to average about five coordinates.

In a rural setting, linestrings tend to be longer as features, such as roads, tend to extend for greater length, interrupted only by streams, rivers, and other rural networks.

The collection data types (multipoints, multilinestrings, and multipolygons) are difficult to estimate. The numbers shown below are based on the datasets that these data types are often applied to (broadcast patterns for multipoints, stream networks for multilinestrings, and island topology for multipolygons).

Data Type	Average Points per Feature
point	1
linestring (urban)	5
linestring (rural)	50
polygons (urban)	7
polygons (rural)	150
multipoint	50
multilinestring	250
multipolygon	1,000

Select the coordinate factor from the table below.

Coordinate Type	Coordinate Factor
xy	4.8
xyz	7.2
xym	7.2
xyzm	9.6

The coordinate factor is based on the type of coordinates stored. If the spatial column stores only x- and y-values, the coordinate factor is 4.8. If in addition to the x- and y-values the spatial column stores z-values (for three-dimensional applications) or measures (used by network analysis packages), the coordinate factor is 7.2. The coordinate factor is 9.6 if both z-values and measures are stored.

If your layer includes annotation, set the annotation size to 300 bytes. The annotation size includes the space required to store the text, the placement geometry, the lead line geometry (if one exists), and various metadata attributes describing the annotation's size and font. Three hundred is the average number of bytes required to store most annotation. The number will vary depending on factors such as the size of the text string and the complexity of the placement geometry. However, the variation becomes insignificant for large tables.

Estimating the actual row size of the spatial table

To determine the row size of the remaining columns of a spatial table, create the table without the spatial column and query the row size column of the systables table. In this example, a lots table is created with three columns.

```
create table lots (lot_id         integer,
                   owner_name     varchar(128),
                   owner_address  varchar(128))
```

Selecting the row size for the lots table from systables returns a value of 262 bytes.

```
select rowsize from systables where tabname = 'lots';
262
```

Tables containing variable length columns of type VARCHAR or NVARCHAR require the row size to be reduced to reflect the actual length of the data stored.

In the sample lots table the lot_id integer column is a fixed length and always occupies four bytes. However, the owner_name and owner_address are variable length varchar columns and may occupy up to 129 bytes each (an extra byte is required for the null

terminator). Upon closer examination it is determined that the average size of the owner name is 68 bytes and the average size of the address is 102. The row size should be reduced.

actual row size = systables row size - ((size of owner_name - average owner name) + (size of owner_address - average owner address))

actual row size = 262 - ((129 - 68) + (129 - 102))

actual row size = 262 - (61 + 27)

actual row size = 174

Estimating the metadata space requirement

Informix requires a certain amount of space to store metadata about the table and the rows. A spatial table with more than 10,000 rows will require about 200 bytes of metadata per row. Tables with fewer than 10,000 rows but more than 1,000 require 300 bytes. Tables with fewer than 1,000 rows add 400 bytes per row.

Number of Rows	Metadata Bytes
>10,000	200
1,000–10,000	300
<1,000	400

Estimating the storage space for the spatial table

To determine the storage space required for the spatial table, obtain a rough estimate of the number of rows in the table. Once you have that, add the spatial column size, the attribute column size, and the metadata size together and multiply the sum by the estimated number of rows. The result is a rough estimate of the size of the spatial table.

Estimating the smart large object storage space

A geometry value is stored inline whenever its size is less than or equal to 930 bytes. Geometry values greater than or equal to 930 bytes are stored offline in a designated smart large object. When geometries are written to a smart large object, an inline pointer of 64 bytes references the geometry.

Determine the amount of smart large object space required with the following formula:

smart large object ratio = (spatial column size / 1920)

The smart large object ratio cannot be greater than 1. So if the smart large object ratio is greater than one, set it to 1.

smart large object space = ((smart large object ratio) * number of rows)

Determine the amount of inline space required with the following formula:

inline space = (size of spatial table) - (smart large object space)

Estimating the size of your ArcSDE indexes

The ArcSDE server creates and maintains two indexes whenever you add a spatial column to one of your tables. The server creates an rtree index on the spatial column and a btree index on the SE_ROW_ID integer column. The spatial column rtree index is named a <N>_ix1, and the SE_ROW_ID btree index is named a <N>_ix2. The <N> in the index names represents the spatial column's unique layer number assigned by the ArcSDE server.

The indexes are 3 percent greater than the metadata and spatial column size of the table. Calculate the index space requirements by combining the spatial column size and the metadata size and multiplying this sum by the expected number of rows in the table. Increase the product by 3 percent (multiply by 1.03).

index space = ((metadata size + spatial column size) * expected number of rows) * 1.03

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.

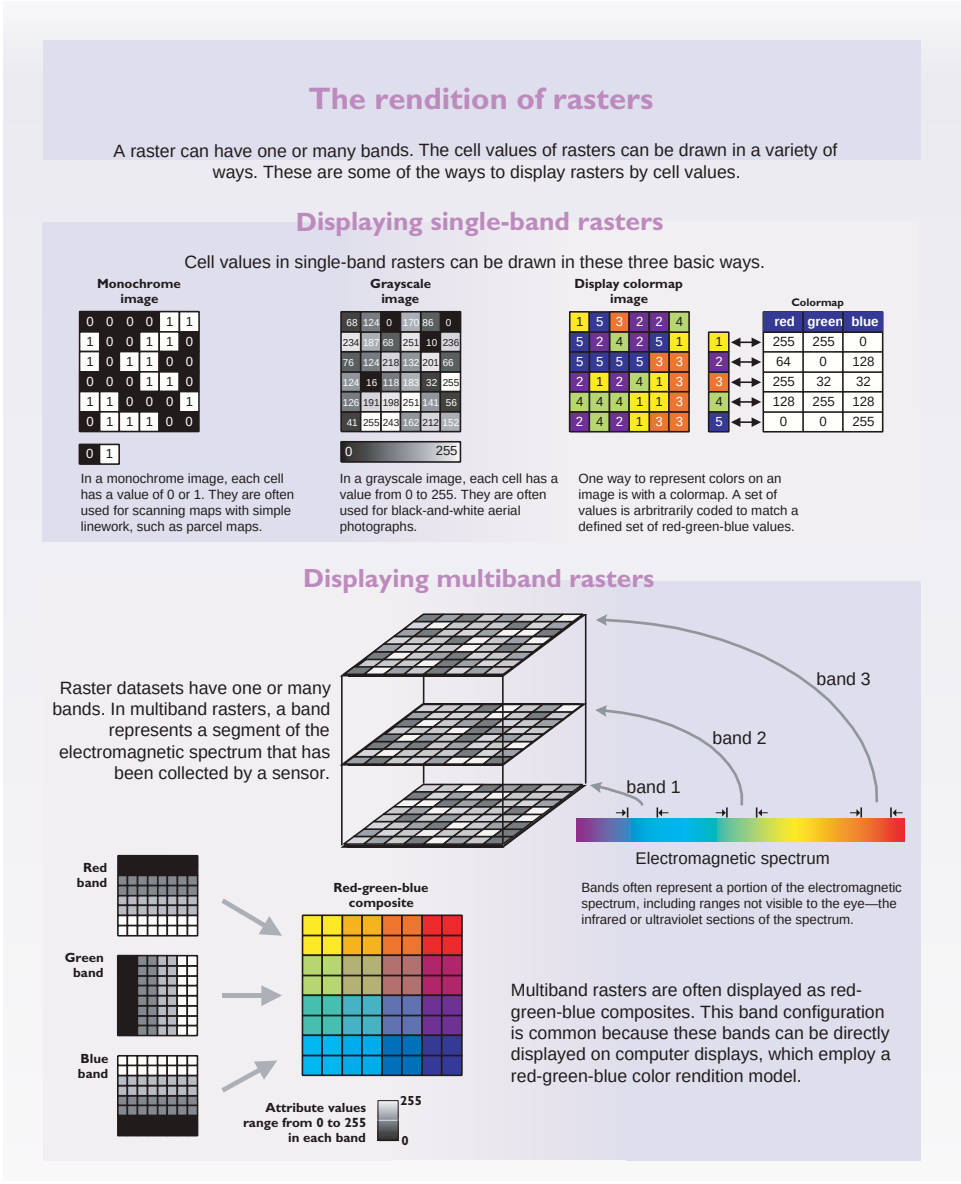
For an expanded discussion of the type of raster data supported by ESRI products, review Chapter 9, 'Cell-based modeling with rasters', in *Modeling Our World*.

ArcSDE stores raster datasets similar to the way it stores compressed binary feature classes (see Appendix C.) 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 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 select 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.

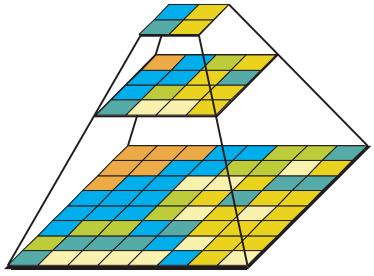


Figure B.1 When you build a pyramid, more rasters are created by progressively downsampling the previous level by a factor of two until the apex is reached. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The 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 Figure B.2 for an illustration of these tables and the manner in which they are associated with one another.

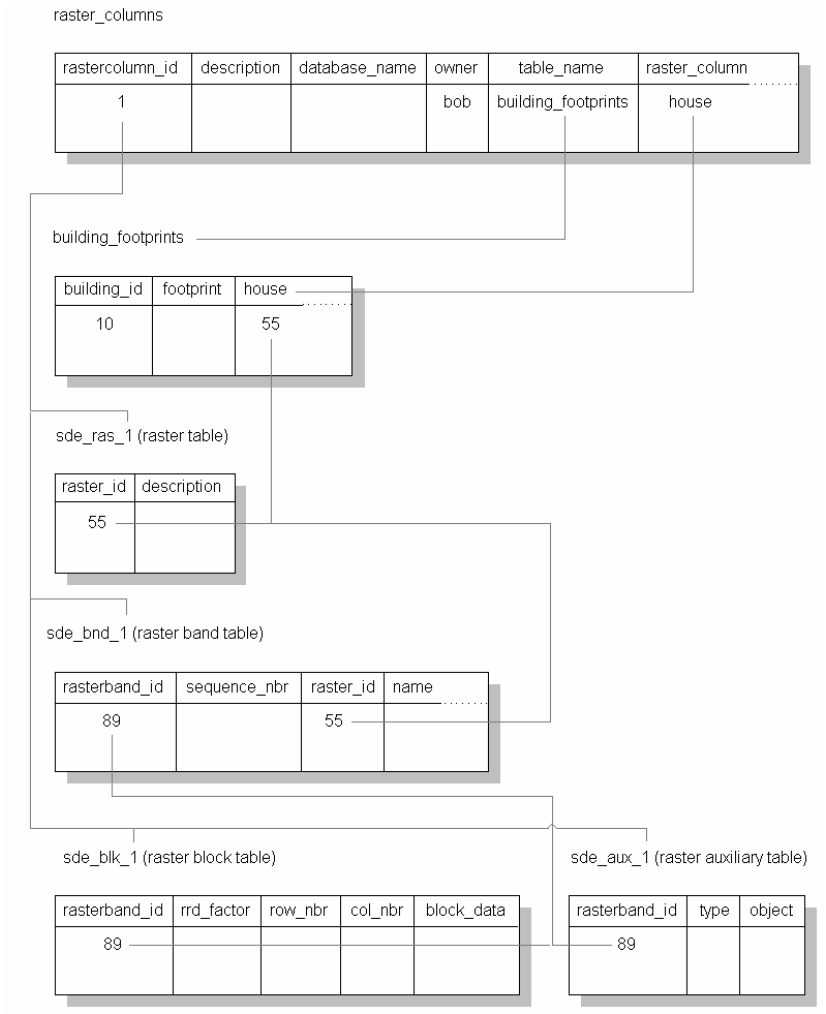


Figure B.2 When ArcSDE adds a raster column to a table, it records that column in the SDE user's raster_columns table. The rastercolumn_id table is used in the creation of the table names of the raster, raster band, raster auxiliary, and raster blocks table.

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 tables 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 bitmap 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 bitmap 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 bitmap 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

Informix Spatial DataBlade geometry types

ArcSDE for Informix stores its spatial data in the Informix Spatial DataBlade® data types. Therefore, before ArcSDE can store spatial data in an Informix database, the Spatial DataBlade must be registered. This document describes the ArcSDE/Informix Spatial DataBlade interface and provides a brief overview of the spatial data types and functions available following the registration of the Informix Spatial DataBlade. For more information on the Informix Spatial DataBlade, consult the *Informix Spatial DataBlade Module User's Guide.*

The Informix Spatial DataBlade embeds a GIS into your Informix® Dynamic Server (IDS) kernel. The Informix Spatial DataBlade module implements the Open GIS Consortium, Inc. (OpenGIS®, or OGC) SQL 3 specification of UDTs, columns capable of storing spatial data such as the location of a landmark, a street, or a parcel of land.

The GIS of the past was spatially centric and focused on gathering spatial data and attaching nonspatial 'attribute' data to it. The Spatial DataBlade module integrates spatial and nonspatial data providing a seamless point of access through the Informix SQL interface.

In addition to new data types, the Informix Spatial DataBlade provides new capabilities such as spatial joins. Application programmers typically join tables by comparing two or more columns to determine whether their values are equal, not equal, greater than, and so on. The Informix Spatial DataBlade includes functions capable of comparing the values of spatial columns to determine if they intersect, overlap, and so forth. These two-dimensional functions can join tables based on their spatial relationship and answer questions such as, "Is this school within five miles of a hazardous waste site?" Internally,

the Informix Spatial DataBlade ST_Overlaps function evaluates this question as, "Does this polygon (the building footprint of a school) overlap this circular polygon (the five-mile radius of a hazardous waste site)?" An application programmer can join a table storing sensitive sites, such as schools, playgrounds, and hospitals, to another table containing the locations of hazardous sites and return a list of sensitive areas at risk.

How the Informix Spatial DataBlade works

Once the Informix Spatial DataBlade is installed, you can create spatially enabled tables that include spatial columns. Geographic features can be inserted into the spatial columns. The Informix Spatial DataBlade converts spatial data into its storage format from one of three external formats:

- Well-known text (WKT) representation
- Well-known binary (WKB) representation
- ESRI shape representation

ArcSDE uses the ESRI shape representation.

Accessing the spatially enabled tables through the ArcSDE server allows you to write applications using the existing tools offered by the GIS software or create applications using the SDE C API. An experienced ODBC programmer can also make calls to the Informix Spatial DataBlade spatial functions. The majority of this document is devoted to discussing and applying these spatial functions.

After integrating spatial data into your database, you can include Spatial DataBlade functions in your SQL statements, comparing the values of spatial columns, transforming the values into other spatial data, and describing the properties of the data.

Adding records to the spatial reference table

The spatial reference system identifies the coordinate transformation matrix for each geometry. Geometry is the term adopted by the OpenGIS Consortium to refer to two-dimensional spatial data. All spatial reference systems known to the database are stored in the SPATIAL_REFERENCES table.

NAME	DATA TYPE	NULL?
srid	integer	NOT NULL
description	varchar(64)	NULL
auth_name	varchar(255)	NULL
auth_srid	integer	NULL

NAME	DATA TYPE	NULL?
falsex	float	NOT NULL
falsey	float	NOT NULL
xyunits	float	NOT NULL
falsez	float	NULL
zunits	float	NULL
falsem	float	NULL
munits	float	NULL
srtext	char(2048)	NOT NULL

Spatial references table schema

The SPATIAL REFERENCES table stores a record for each spatial reference in the database.

The datatype for each column is defined below.

- srid (SE_INTEGER_TYPE)—Contains the unique ID that identifies each SRID in the database.
- description (SE_STRING_TYPE)—An optional short description of the spatial reference system. ArcSDE leaves this field NULL when it creates the spatial reference system automatically.
- auth_name (SE_STRING_TYPE)—The name of the standard body cited for the spatial references system. ArcSDE leaves this field NULL when it creates the spatial reference system automatically.
- auth_srid (SE_STRING_TYPE)—The ID of the spatial reference system as defined by the authority cited in auth_name. ArcSDE leaves this field NULL when it creates the spatial reference system automatically.
- falsex (SE_FLOAT_TYPE)—The x-value offset or the minimum allowable X-ordinate value.
- falsey (SE_FLOAT_TYPE)—The y-value offset or the minimum allowable Y-ordinate value.
- xyunits (SE_FLOAT_TYPE)—The XY coordinate system units or spatial reference system's XY coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.
- falsez (SE_FLOAT_TYPE)—The z-value offset or the minimum allowable Z-ordinate value.

- zunits (SE_FLOAT_TYPE)—The z-coordinate system units or spatial reference system's z-coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.
- falsem (SE_FLOAT_TYPE)—The m-value offset or the minimum allowable M-ordinate value.
- munits (SE_FLOAT_TYPE)—The m-coordinate system units or spatial reference system's m-coordinate precision. Coordinates whose precision exceeds this value are truncated when they are stored.
- srtext (SE_STRING_TYPE)—The srtext column contains the well-known text representation of the spatial reference system. For information on this subject, see Appendix B, 'OGC Well Known Text Representation of Spatial Systems', in the *Informix Spatial DataBlade Module Users Guide*.

Internal functions use the parameters of a spatial reference system to translate and scale each floating point coordinate of the geometry into 32-bit positive integers prior to storage. Upon retrieval, the coordinates are restored to their external floating point format.

The floating point coordinates are converted to integers by subtracting the falsex and falsey values, which translates to the false origin, scales by multiplying by the xyunits, adds a half unit, and truncates the remainder.

The optional z-coordinates and measures are dealt with similarly, except that they are translated with falsez and falsem and scaled with zunits and munits, respectively.

SRID, the spatial_references primary key, contains a unique number for each spatial reference system.

The spatial reference system is assigned to a geometry during its construction. The spatial reference system must exist in the spatial reference table. All geometries in a column must have the same spatial reference system.

Whenever ArcSDE creates a feature class, it searches the SPATIAL_REFERENCES table in an attempt to locate a matching spatial reference system. If one is found the SRID is assigned to the feature class; otherwise, ArcSDE adds a new spatial reference system to the SPATIAL_REFERENCES table and assigns it to the feature class.

The ArcSDE administration tools, shp2sde columns and cov2sde columns, provide an option for you to enter a predefined SRID when you use them to create a new feature class. In this example, the roads coverage is converted to the roads feature class with a SRID of 10. The coordinates of the coverage feature must fit within the extent, of the

spatial reference system. Each feature found to lie outside the spatial reference system's extent is rejected.

```
cov2sde -o create -l roads,feature -f roads -R 10 -g 100,0,0 -u world -p
world
```

Creating feature classes in an Informix database

An Informix spatial table can include one or more spatial columns, although ArcSDE restricts a feature class to a single spatial column. Spatial columns are defined with one of the Informix Spatial DataBlade's UDTs. A spatial column can only accept data of the type required by the spatial column. For example, an ST_Polygon column rejects integers, characters, and even other types of nonpolygon geometry.

When ArcSDE creates an Informix table with a spatial column, it also creates an SE_ROW_ID integer column. The SE_ROW_ID column is required by ArcSDE client applications to keep track of selection sets; more specifically it is used in ArcSDE log files.

ArcSDE adds a record to the GEOMETRY_COLUMNS table whenever it creates a feature class in an Informix database. Applications using the Informix Spatial DataBlade are responsible for inserting a record into the GEOMETRY_COLUMNS table each time they add a spatial column to the database.

NAME	DATA TYPE	NULL?
f_table_catalog	varchar(32)	NOT NULL
f_table_schema	varchar(32)	NOT NULL
f_table_name	varchar(128)	NOT NULL
f_geometry_column	varchar(128)	NOT NULL
storage_type	integer	NULL
geometry_type	integer	NOT NULL
coord_dimension	integer	NULL
srid	integer	NOT NULL

Geometry_columns table schema

The GEOMETRY_COLUMNS table stores a record for each geometry column in the database.

The datatype for each column is defined below.

- f_table_catalog (SE_STRING_TYPE)—The database in which the geometry column's table is stored.

- f_table_schema (SE_STRING_TYPE)—The owner of the geometry column's table.
- f_table_name (SE_STRING_TYPE)—The geometry column's table name.
- f_geometry_column (SE_STRING_TYPE)—The name of the geometry column.
- storage_type (SE_INTEGER_TYPE)—This is an OGC required field that is not used by ArcSDE.
- geometry_type (SE_INTEGER_TYPE)—The geometry type code. ArcSDE inserts the following values into this field:

Geometry Type Code	Geometry Type
0	ST_Geometry
1	ST_Point
3	ST_LineString
5	ST_Polygon
7	ST_MultiPoint
9	ST_MultiLineString
11	ST_MultiPolygon

- coord_dimension (SE_INTEGER_TYPE)—This is an OGC required field that is not used by ArcSDE.
- srid (SE_INTEGER_TYPE)—The geometry column's spatial reference system. This is a foreign key to the SRID column of the SPATIAL_REFERENCES table.

Creating a spatial index

Spatial columns contain two-dimensional geographic data, and applications querying those columns require an index strategy that will quickly identify all geometries that lie within a given extent. For this reason the Informix Spatial DataBlade provides support for building a spatial index called an R-tree spatial index.

The R-tree index differs from the traditional hierarchical btree index provided by the Informix Dynamic Server software.

The btree index cannot be applied to a spatial column because the two-dimensional characteristic of the spatial column requires an R-tree index. For the same reason, you can't apply R-tree indexes to a nonspatial column or a composite column.

The R-tree index's 'create index' syntax includes the additional 'using rtree' clause to create an R-tree index rather than a btree index. The full syntax is

```
create index <index> on <table> (<spatial column> ST_Geometry_Ops) using
rtree (<parameters>) <index options>;
```

The ST_Geometry_Ops is the Informix Spatial DataBlade operator class. ST_Geometry_Ops manages the R-tree index.

ArcSDE creates a spatial index when a feature class is first created and when it is switched from load_only_io mode to normal_io mode. The spatial index is created with default parameter bottom_up_build = 'yes' and no index options.

You do not need to ever tune the spatial index for performance since this is all handled through the R-TREE index. Therefore, you will never have to experiment with the spatial index by trying different cell sizes and different grid level configurations. ArcSDE for Informix does not require specifying a spatial index or defining spatial grid sizes. You can completely ignore the "-g" Spatial Index flag in all ArcSDE client executables, i.e.,

shp2sde -o create -l <table,column> [-V <version_name>] -f <shape_file> [-I] [Spatial_Index] [{-R <SRID> | [Spatial_Ref_Opts]}] [-S <layer_description_str>] [-v] [-e <entity_mask>] [-k <config_keyword>] [-M <minimum_ID>] [-a {none | all | file=<file_name>}] [-r <reject_shpfile>] [-c <commit_interval>] [-i <service>] [-s <server_name>] [-D <database>] -u <DB_User_name> [-p <DB_User_password>]

Where [Spatial_Index] := [-g {<grid_sz0>[,<grid_sz1>[,<grid_sz2>]] |GRID,<grid_sz0>[,<grid_sz1>[,<grid_sz2>]] |DBTUNE |NONE |RTREE |FIXED,<sdo_level> |HYBRID,<sdo_level>,<sdo_num_tiles>}]

Setting the "-g" flag will not hurt anything but the "-g" flag is ignored for ArcSDE for Informix and therefore will not be used. Nor is it ever necessary.

Updating statistics

The Informix optimizer will not use the R-tree index unless the statistics on the table are up-to-date. If the R-tree index is created after the data has been loaded, the statistics are up-to-date and the optimizer will use the index. However, if the index is created, and data is loaded afterwards, the optimizer will not use the R-tree index because the statistics will be out of date. To update the statistics use the update statistics Informix SQL statement.

```
update statistics for table <table_name>
```

Spatial DataBlade data types

The *Oxford American Dictionary* defines the noun 'geometry' as "the branch of mathematics dealing with the properties of and relations of lines, angles, surfaces, and solids." On August 11, 1997, the OGC, in its publication of *OpenGIS Features for ODBC (SQL) Implementation Specification*, coined another definition for the noun geometry. The word was selected to define the geometric features that, for the past millennium or more, cartographers have used to map the world. Typically, points represent an object at a single location, linestrings represent a linear characteristic, and polygons represent a spatial extent. A very abstract definition of the Open GIS noun geometry might be "a point or aggregate of points symbolizing a feature on the ground". This definition, however, fails to describe the rich set of properties and functionality associated with geometry.

To understand geometry in this context it is easier to describe it as it has been implemented within the Informix Spatial DataBlade as a UDT, and like all UDTs in an object relational system, it has a unique set of properties and methods.

ST_Geometry columns as a data type allow you to define columns that store spatial data. The ST_Geometry data type itself is an abstract noninstantiable superclass, the subclasses of which are instantiable. An instantiated data type is one that can be defined as a table column and have values of its type inserted into it. A column can be defined as type ST_Geometry, but ST_Geometry values cannot be inserted into it since they cannot be instantiated. Only the subclass values can be inserted into this column because only they can be instantiated. Therefore, the ST_Geometry data type can accept and store any of its subclasses, while its subclass data types can only accept their own values.

Throughout the remainder of this document the term geometry or geometries collectively refers to the superclass ST_Geometry data type and all of its subclass data types. Whenever it is necessary to specify the geometry superclass directly, it will be referred to as the ST_Geometry superclass or the ST_Geometry data type.

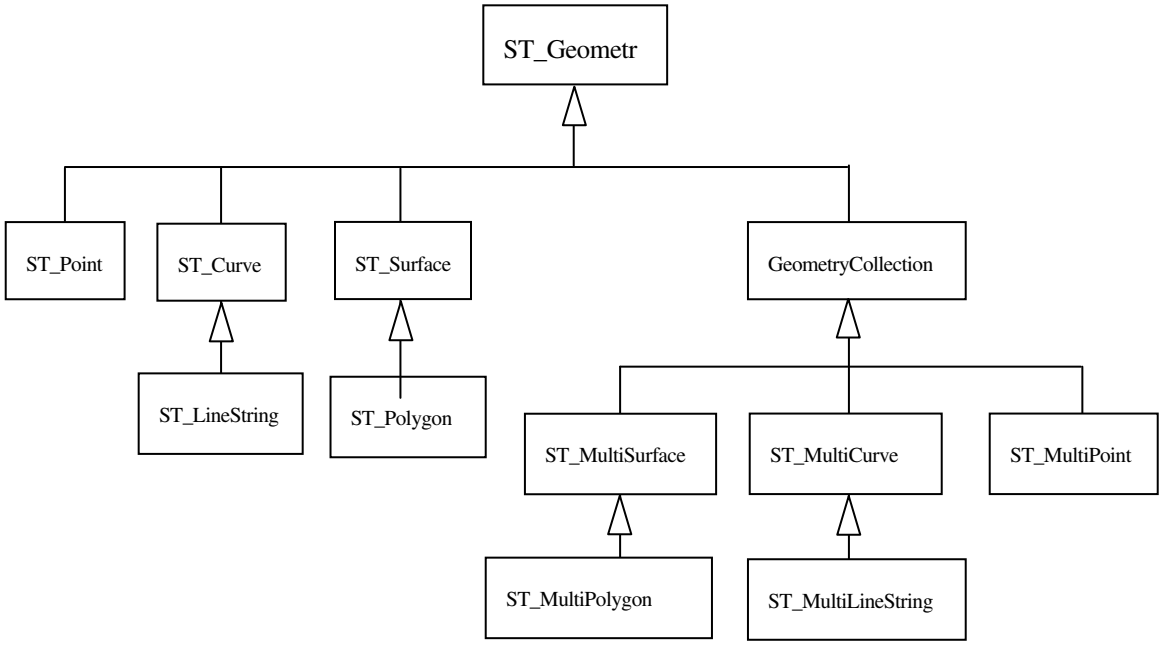


Figure C.1 The hierarchy of the ST_Geometry datatype is divided into the subtypes ST_Point, ST_Curve, and ST_Surface simple types and the geometry collections ST_MultiSurface, ST_MultiCurve, and ST_MultiPoint. ST_LineString is the subtype of ST_Curve. ST_Polygon is the subtype of ST_Surface. ST_MultiPolygon is the subtype of ST_MultiSurface. ST_MultiLineString is the subtype of ST_MultiCurve.

Geometry properties

Each subclass inherits the properties of the ST_Geometry superclass but also has properties of its own. Functions that operate on the ST_Geometry data type will accept any of the subclass data types. However, some functions have been defined at the subclass level and will only accept certain subclasses' data types.

Interior, boundary, exterior

All geometries occupy a position in space defined by its interior, boundary, and exterior. The exterior of a geometry is all space not occupied by the geometry. The boundary of a geometry serves as the interface between its interior and exterior. The interior is the space occupied by the geometry. The subclass inherits the interior and exterior properties directly; however, the boundary property differs for each.

The ST_Boundary Spatial DataBlade function takes an ST_Geometry and returns an ST_Geometry that represents the source ST_Geometry's boundary.

Simple or nonsimple

Some subclasses of ST_Geometry (ST_LineStrings, ST_MultiPoints, and ST_MultiLineStrings) are either simple or nonsimple. They are simple if they obey all topological rules imposed on the subclass and nonsimple if they "bend" a few. An ST_LineString is simple if it does not intersect its interior. An ST_MultiPoint is simple if none of its elements occupy the same coordinate space. An ST_MultiLineString is simple if none of its element's interiors are intersected by its own interior.

The Spatial DataBlade ST_IsSimple predicate function takes an ST_Geometry and returns t (TRUE) if the ST_Geometry is simple and f (FALSE) otherwise.

Empty or not empty

A geometry is empty if it does not have any points. An empty geometry has a NULL envelope, boundary, interior, and exterior. An empty geometry is always simple and can have z-coordinates or measures. Empty linestrings and multilinestrings have a 0 length. Empty polygons and multipolygons have a 0 area.

The Spatial DataBlade ST_IsEmpty predicate function takes an ST_Geometry and returns t (TRUE) if the ST_Geometry is empty and f (FALSE) otherwise.

Number of points

A geometry can have zero or more points. A geometry is considered empty if it has zero points. The point subclass is the only geometry that is restricted to zero or one point; all other subclasses can have zero or more.

Envelope

The envelope of a geometry is the bounding geometry formed by the minimum and maximum (x,y) coordinates. The envelopes of most geometries form a boundary rectangle; however, the envelope of a point is the point since its minimum and maximum coordinates are the same, and the envelope of a horizontal or vertical linestring is a linestring represented by the boundary (the endpoints) of the source linestring.

The Spatial DataBlade ST_Envelope function takes an ST_Geometry and returns an ST_Geometry that represents the source ST_Geometry's envelope.

Dimension

A geometry can have a dimension of 0, 1, or 2.

The dimensions are

0—has neither length nor area

1—has a length

2—contains area

The point and multipoint subclasses have a dimension of 0. Points represent zero-dimensional features that can be modeled with a single coordinate, while multipoints represent data that must be modeled with a cluster of unconnected coordinates.

The subclasses linestring and multilinestring have a dimension of 1. They store road segments, branching river systems, and any other features that are linear in nature.

Polygon and multipolygon subclasses have a dimension of 2. Forest stands, parcels, water bodies, and other features whose perimeter encloses a definable area can be rendered by either the polygon or multipolygon data type.

Dimension is important not only as a property of the subclass but also in playing a part in determining the spatial relationship of two features. The dimension of the resulting feature or features determines whether or not the operation was successful. The dimension of the features is examined to determine how they should be compared.

The Spatial DataBlade ST_Dimension function takes an ST_Geometry and returns its dimension as an integer.

Z-coordinates

Some geometries have an associated altitude or depth. Each of the points that form the geometry of a feature can include an optional z-coordinate that represents an altitude or depth normal to the earth's surface.

The Spatial DataBlade SE_Is3D predicate function takes an ST_Geometry and returns t (TRUE) if the function has z-coordinates and f (FALSE) otherwise.

Measures

Measures are values assigned to each coordinate. The value represents anything that can be stored as a double-precision number.

The Spatial DataBlade SE_IsMeasured predicate function takes a geometry and returns t (TRUE) if it contains measures and f (FALSE) otherwise.

Spatial reference system

The spatial reference system identifies the coordinate transformation matrix for each geometry.

The Spatial DataBlade ST_SRID function takes an ST_Geometry and returns its spatial reference identifier as an integer.

Instantiable subclasses

The ST_Geometry data type is not instantiable but instead must store its instantiable subclasses. The subclasses are divided into two categories: the base geometry subclasses and the homogeneous collection subclasses. The base geometries include ST_Point, ST_LineString, and ST_Polygon, while the homogeneous collections include ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon. As the names imply, the homogeneous collections are collections of base geometries. In addition to sharing base geometry properties, homogeneous collections have some of their own properties as well.

The Spatial DataBlade ST_GeometryType function takes an ST_Geometry and returns the instantiable subclass in the form of a character string. The Spatial DataBlade ST_NumGeometries function takes a homogeneous collection and returns the number of base geometry elements it contains. The Spatial DataBlade ST_GeometryN function takes a homogeneous collection and an index and returns the nth base geometry.

ST_Point

An ST_Point is a zero-dimensional geometry that occupies a single location in coordinate space. An ST_Point has a single x,y coordinate value. An ST_Point is always simple; has a NULL boundary; and is used to define features such as oil wells, landmarks, and elevations.

Spatial DataBlade functions that operate solely on the ST_Point data type include ST_X, ST_Y, SE_Z, and SE_M.

The ST_X function returns a point data type's x-coordinate value as a double-precision number.

The ST_Y function returns a point data type's y-coordinate value as a double-precision number.

The SE_Z function returns a point data type's z-coordinate value as a double-precision number.

The SE_M function returns a point data type's m-coordinate value as a double-precision number.

ST_LineString

An ST_LineString is a one-dimensional object stored as a sequence of points defining a linear interpolated path. The ST_LineString is simple if it does not intersect its interior. The endpoints (the boundary) of a closed ST_LineString occupy the same point in space. An ST_LineString is a ring if it is both closed and simple. As well as the other properties inherited from the superclass ST_Geometry, ST_LineStrings have length. ST_LineStrings are often used to define linear features such as roads, rivers, and power lines.

The endpoints normally form the boundary of an ST_LineString unless the ST_LineString is closed, in which case the boundary is NULL. The interior of an ST_LineString is the connected path that lies between the endpoints, unless it is closed, in which case the interior is continuous.

Spatial DataBlade functions that operate on ST_LineStrings include ST_StartPoint, ST_EndPoint, ST_PointN, ST_Length, ST_NumPoints, ST_IsRing, and ST_IsClosed.

The ST_StartPoint function takes an ST_LineString and returns its first point.

The ST_EndPoint function takes an ST_LineString and returns its last point.

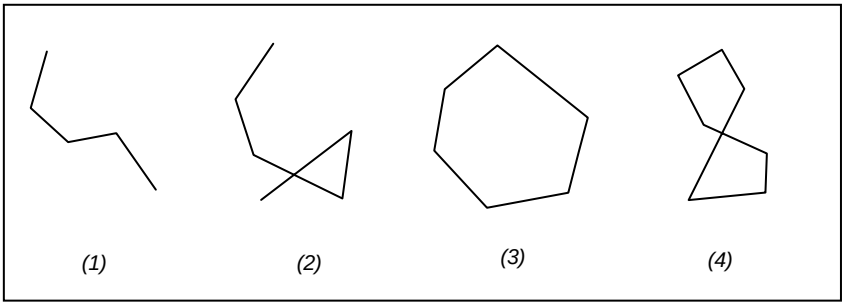
The ST_PointN function takes an ST_LineString and an index to an nth point and returns that point.

The ST_Length function takes an ST_LineString and returns its length as a double-precision number.

The ST_NumPoints function takes an ST_LineString and returns the number of points in its sequence as an integer.

The ST_IsRing predicate function takes an ST_LineString and returns t (TRUE) if the ST_LineString is a ring and f (FALSE) otherwise.

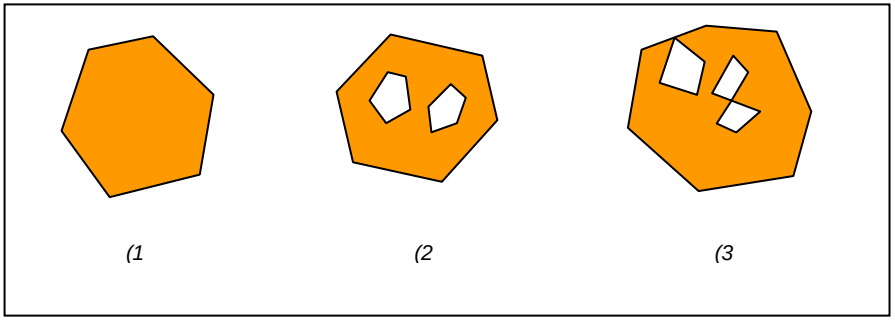
The ST_IsClosed predicate function takes an ST_LineString and returns t (TRUE) if the ST_LineString is closed and f (FALSE) otherwise.



Examples of ST_LineString objects: (1) a simple nonclosed ST_LineString, (2) a nonsimple nonclosed ST_LineString, (3) a closed simple ST_LineString and is therefore a ring, (4) a closed nonsimple ST_LineString and is not a ring.

ST_Polygon

An ST_Polygon is a two-dimensional surface stored as a sequence of points defining its exterior bounding ring and 0 or more interior rings. ST_Polygon, by definition, is always simple. Most often ST_Polygon defines parcels of land, water bodies, and other features having spatial extent.



Examples of ST_Polygon objects: (1) an ST_Polygon whose boundary is defined by an exterior ring, (2) an ST_Polygon whose boundary is defined by an exterior ring and two interior rings and the area inside the interior rings is part of the ST_Polygon's exterior, and (3) a legal ST_Polygon because the rings intersect at a single tangent point.

The exterior and any interior rings define the boundary of an ST_Polygon, and the space enclosed between the rings defines the ST_Polygon's interior. The rings of an

ST_Polygon can intersect at a tangent point but never cross. In addition to the other properties inherited from the superclass ST_Geometry, ST_Polygon has area.

Spatial DataBlade functions that operate on ST_Polygon include ST_Area, ST_ExteriorRing, ST_NumInteriorRing, ST_InteriorRingN, ST_Centroid, and ST_PointOnSurface.

The ST_Area function takes an ST_Polygon and returns its area as a double-precision number.

The ST_ExteriorRing function takes an ST_Polygon and returns its exterior ring as an ST_LineString.

The ST_NumInteriorRing takes an ST_Polygon and returns the number of interior rings that it contains.

The ST_InteriorRingN function takes an ST_Polygon and an index and returns the nth interior ring as an ST_LineString.

The ST_Centroid function takes an ST_Polygon and returns an ST_Point that is the center of the ST_Polygon's envelope.

The ST_PointOnSurface function takes an ST_Polygon and returns an ST_Point that is guaranteed to be on the surface of the ST_Polygon.

ST_MultiPoint

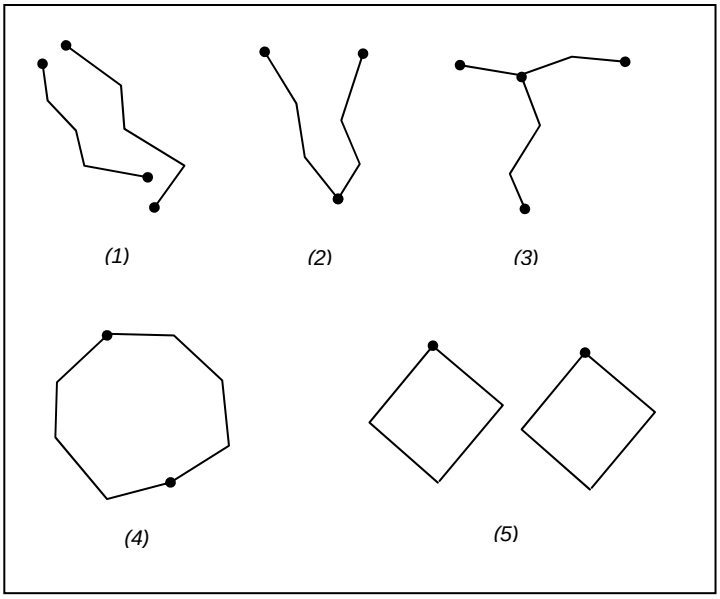
An ST_MultiPoint is a collection of ST_Points and, just like its elements, it has a dimension of 0. An ST_MultiPoint is simple if none of its elements occupy the same coordinate space. The boundary of an ST_MultiPoint is NULL. ST_MultiPoints define aerial broadcast patterns and incidents of a disease outbreak.

ST_MultiLineString

An ST_MultiLineString is a collection of ST_LineStrings. ST_MultiLineStrings are simple if they only intersect at the endpoints of the ST_LineString elements. ST_MultiLineStrings are nonsimple if the interiors of the ST_LineString elements intersect.

The boundary of an ST_MultiLineString is the nonintersected endpoints of the ST_LineString elements. The ST_MultiLineString is closed if all its ST_LineString elements are closed. The boundary of an ST_MultiLineString is NULL if all the endpoints of all the elements are intersected. In addition to the other properties inherited

from the superclass ST_Geometry, ST_MultiLineStrings have length. ST_MultiLineStrings are used to define streams or road networks.



Examples of ST_MultiLineStrings: (1) a simple ST_MultiLineString whose boundary is the four endpoints of its two ST_LineString elements; (2) a simple ST_MultiLineString because only the endpoints of the ST_LineString elements intersect. The boundary is two nonintersected endpoints; (3) a nonsimple ST_MultiLineString because the interior of one of its ST_LineString elements is intersected. The boundary of this ST_MultiLineString is the three nonintersected endpoints; (4) a simple nonclosed ST_MultiLineString. It is not closed because its element ST_LineStrings are not closed. It is simple because none of the interiors of any of the element ST_LineStrings intersect; (5) a simple closed ST_MultiLineString. It is closed because all its elements are closed. It is simple because none of its elements intersect at the interiors.

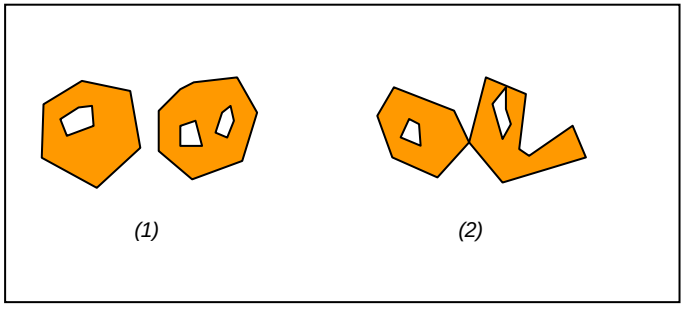
Spatial DataBlade functions that operate on ST_MultiLineStrings include ST_Length and ST_IsClosed.

The ST_Length function takes an ST_MultiLineString and returns the cumulative length of all its ST_LineString elements as a double-precision number.

The ST_IsClosed predicate function takes an ST_MultiLineString and returns t (TRUE) if the ST_MultiLineString is closed and f (FALSE) otherwise.

ST_MultiPolygon

The boundary of an ST_MultiPolygon is the cumulative length of its elements' exterior and interior rings. The interior of an ST_MultiPolygon is defined as the cumulative interiors of its element ST_Polygons. The boundary of an ST_MultiPolygon's elements can only intersect at a tangent point. In addition to the other properties inherited from the superclass ST_Geometry, ST_MultiPolygons have area. ST_MultiPolygons define features such as a forest stratum or a noncontiguous parcel of land such as a Pacific island chain.



Examples of ST_MultiPolygon: (1) an ST_MultiPolygon with two ST_Polygon elements. The boundary is defined by the two exterior rings and the three interior rings; and (2) an ST_MultiPolygon with two ST_Polygon elements. The boundary is defined by the two exterior rings and the two interior rings. The two ST_Polygon elements intersect at a tangent point.

Spatial DataBlade functions that operate on ST_MultiPolygons include ST_Area, ST_Centroid, and ST_PointOnSurface.

The ST_Area function takes an ST_MultiPolygon and returns the cumulative ST_Area of its ST_Polygon elements as a double-precision number.

The ST_Centroid function takes an ST_MultiPolygon and returns an ST_Point that is the center of an ST_MultiPolygon's envelope.

The ST_PointOnSurface function takes an ST_MultiPolygon and returns an ST_Point that is guaranteed to be normal to the surface of one of its ST_Polygon elements.

APPENDIX D

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, the 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 GCDRULES table. Each row in the 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 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(s), 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

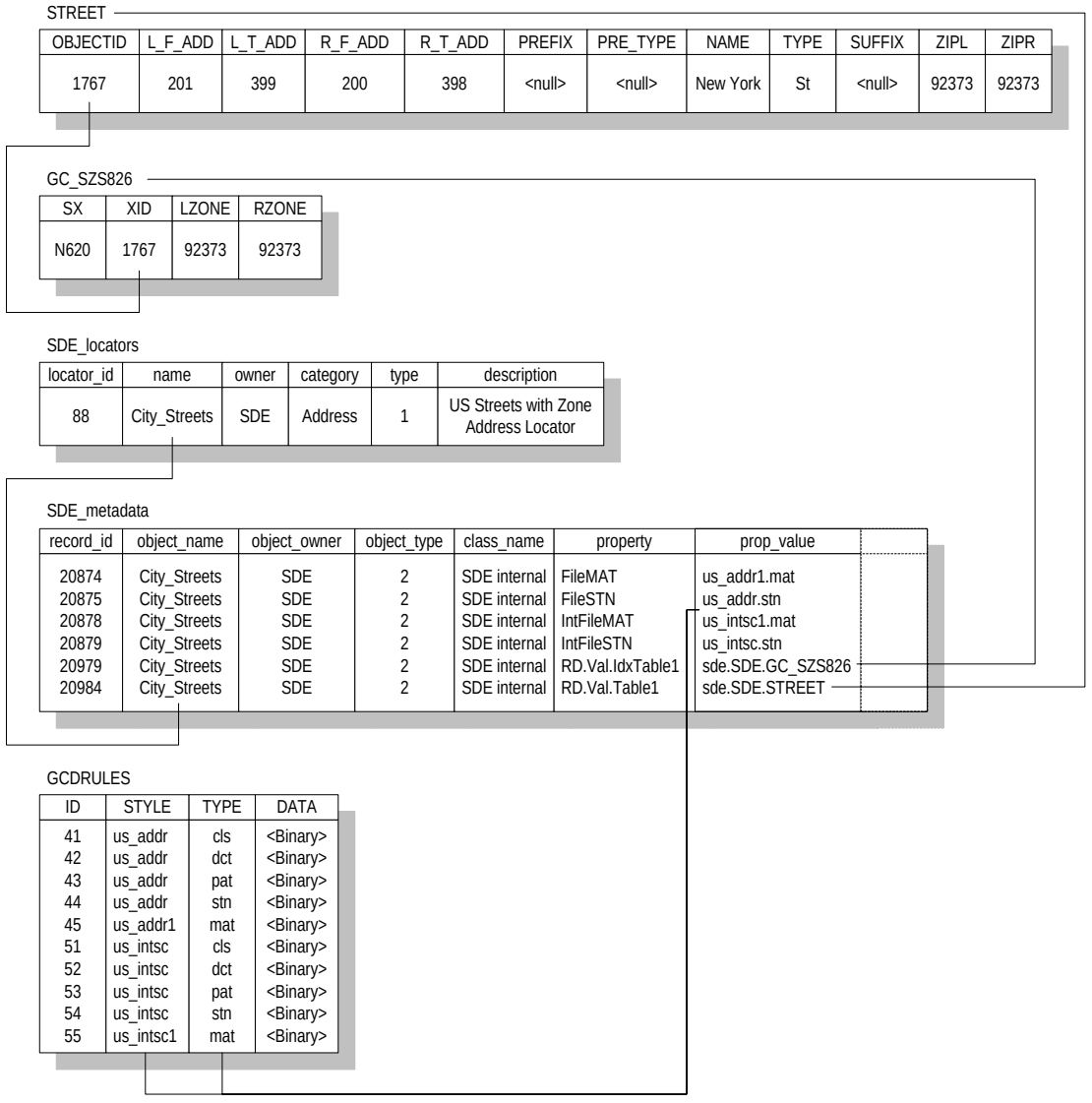
When you create a locator in an ArcSDE database, ArcSDE adds a record to the SDE_locators table that defines the locator. ArcSDE also adds a record to the SDE_metadata table for each property of the locator. The object_name column in the SDE_metadata table is a foreign key to the Name column in the SDE_locators 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 GCDRULES table, the locator uses to match addresses. The locator uses the value of these properties in the SDE_metadata table to query the 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 described here.



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. By default, ArcSDE stores geometry for feature

classes in a separate feature table in the ArcSDE compressed binary format, which is described in Appendix A.

NAME	DATA TYPE	NULL?
OBJECTID	INT(4)	NOT NULL
L_F_ADD	INT(4)	NULL
L_T_ADD	INT(4)	NULL
R_F_ADD	INT(4)	NULL
R_T_ADD	INT(4)	NULL
PREFIX	VARCHAR(2)	NULL
PRE_TYPE	VARCHAR(5)	NULL
NAME	VARCHAR(30)	NULL
TYPE	VARCHAR(5)	NULL
SUFFIX	VARCHAR(2)	NULL
ZIPL	VARCHAR(5)	NULL
ZIPR	VARCHAR(5)	NULL
Shape	INT(4)	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_INTEGER_TYPE) – a foreign key to the feature table containing 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	VARCHAR(4)	NULL
XID	INT(4)	NULL
LZONE	VARCHAR(5)	NULL
RZONE	VARCHAR(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	INT(4)	NOT NULL
Name	VARCHAR(32)	NOT NULL
Owner	VARCHAR(32)	NOT NULL
Category	VARCHAR(32)	NOT NULL
Type	INT(4)	NOT NULL
Description	VARCHAR(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 – define locator styles
- 1 – define locators (i.e., locators that can be used to find locations)
- 2 – define attached locators (i.e., 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	INT(4)	NOT NULL
object_database	VARCHAR(32)	NULL
object_name	VARCHAR(160)	NULL
object_owner	VARCHAR(32)	NOT NULL
object_type	INT(4)	NOT NULL
class_name	VARCHAR(32)	NULL
property	VARCHAR(32)	NULL
prop_value	VARCHAR(255)	NULL
description	VARCHAR(65)	NULL
creation_date	DATETIME(8)	NOT NULL

SDE metadata table

- record_id (SE_INTEGER_TYPE) – the table's primary key
- object_database (SE_STRING_TYPE) – the ArcSDE database in which the described object is storedS; not used for locator properties
- 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

GCDRULES table

The GCDRULES table stores the geocoding rules that are used by address locators to match addresses. Each record in the 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	INT(4)	NOT NULL
STYLE	VARCHAR(32)	NULL
TYPE	VARCHAR(3)	NULL
DATA	image	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 E

Making a direct connection

Direct connect is another configuration option for ArcSDE and all the ArcSDE concepts and pre-requisites also apply to direct connect. The main difference between ArcSDE's application server and direct connect is where the ArcSDE processing takes place. This purpose of this appendix is to provide administrators information on how to setup 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 2 sets of ESRI-supplied files required for direct connect:

1 direct connect drivers. 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
- Oracle 8i and 9i

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.

2 database setup files. These are files needed by an administrator to setup and configure a DBMS for direct connect and include files like 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:

- **Oracle*8i*™, Oracle*9i*™**

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 9.0 or ArcView GIS 3.3 with Database Access 2.1f, MDAC version 2.6(SP1) or greater is required. If using ArcIMS 9.0 or ArcGIS 9.0 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 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-ROM's in the ArcSDE media kit. The ArcSDE media kit has CD-ROM's 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/platform. For example, if you are using IBM DB2 on a Sun Solaris server, you

will select the Sun Solaris CD-ROM from the ArcSDE media kit and install the DB2 version of ArcSDE on your Sun Solaris server. Please be sure to follow the post installation 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 to disk and then follow the post installation configuration instructions. The administration files that get installed (eg: sdesetup<dbms>, sdeconfig, sdedbtune, sdelayer) are useful for managing your connection parameters, dbtune table and manual registration/unregistration of 3rd party layers. Please see the *Managing ArcSDE Services* book 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 systems.

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 dynamicly linked library files get installed. For Windows applications, this is normally in the bin directory of your client applications 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 like this

```
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 specifiied.

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 9.0 C API (a list of these applications is in Chapter 1, 'Introducing direct connect')
- 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:and

   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 u sed 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 9.0 to a a database that is still at an 8.3 configuration. You would have to run the 9.0 setup configuration on that 8.3 database to be able to use direct connect from the ArcMap 9.0 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 an existing ArcSDE user, your ArcSDE 8.x keycode will not work with 9.0. To register in the United States, go http://service.esri.com. If you are not in the United States, please call your local distributor to register your software. If the Internet is not an option, you can contact ESRI Customer Service or your local distributor to register and receive your 9.0 authorization file.

Setting up clients for Informix direct connect

Set up the database

You must set up and configure each database you'd like your users to be able to connect directly to. Use standard Informix tools, ArcSDE tools and documentation to

1. Install application server software

2. Perform the post installation 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 machine, you perform the following steps in order on the client machine:

1. Install the Informix Client-SDK or the I-Connect application on each remote client machine you want to direct connect to your database. This makes the Informix client setup files you need available to you.

   Note: if the Informix server is running on the same host as the client application, you can skip this step.

2. Ensure that the user on each client machine you want to connect directly to your database has the Connect permission on the database server.

3. For Windows:
   Use SetNet32 to configure the client so that it can work with the available objects. If you want to connect a client via an ODBC driver, use SetNet32 to add a new Informix server.

4. For UNIX:
   Set up the $INFORMIXDIR and add the appropriate database server entry in the sqlhosts file in the $INFORMIXDIR\etc directory.

5. Set environment variables.

6. Test the connection.

Set up an ODBC driver connection in Windows 2000

Note: instructions here are provided as a convenience and do not supercede or otherwise replace Informix documentation on this topic. Please consult your Informix documentation for further information or help on this topic.

1. Open the SetNet32 application (it comes with Informix Client-SDK and Informix I-Connect) by clicking Start | Programs | Informix | Informix SetNet32.

2. In the Informix Setnet32 dialog box, click Server Information.

3. In the Server Information tab, enter information in the text areas as needed.

In the Service Name text area, type the TCP port for the Informix database. If you don't know the TCP port, you can find it by opening the Informix sqlhosts file on the database server. You can also type an alias, provided you've aliased the TCP port with a service name and defined the service in the services file.

4. Click Apply.

5. In the Prompt that asks if you wish to define a new server, click OK.

6. In the Informix Setnet32 dialog box, click OK to close the SetNet32 application.

7. Click Start | Settings | Control Panel | Administrative Tools | Data Sources (ODBC).

8. In the ODBC Data Source Administrator dialog box, click the System DSN tab.

9. In the System DSN tab, click Add.

10. In the Create New Data Source dialog box, in the Name list area, click the Informix 3.80 32-bit driver, then click Finish.

11. In the Informix ODBC Driver Setup dialog box, click the General tab.

12. In the General tab, in the Data Source Name text box, type a unique data source name.

13. In the Connection tab, in the Server Name list area, select the server you just added in SetNet32. In the Database Name list area, select the database name to which you want the client to connect to.

14. In the User Id text box, type the user name for the database user that corresponds to the client machine you are setting up. In the Password text box, have your user type his or her password.

15. In the Environment tab, in the Cursor Behavior list area, select 1 - Preserve.

16. Click OK to close the Informix ODBC Driver Setup dialog box.

Set up a connection in UNIX

This procedure describes how to connect to another Informix server using direct connect from Unix.

1. Install the Client-SDK or the I-Connect application to access the Informix client files.

2. Set up the $INFORMIXDIR and add the appropriate database server entry in the sqlhosts file in the $INFORMIXDIR\etc directory.

A typical entry in the sqlhosts file is:

```
iron_directconnect        ontlitcp        iron    1526
```

3. Add an entry in the .odbc.ini file in the user's home directory. A sample entry follows:

```
[sdedirect]
Driver=/iron1/informix/lib/cli/iclis09b.so
Description=Informix 9.x ODBC Driver
Database=sde
HostName=iron
Service=informix
Protocol=onsoctcp
Servername=iron_net
CursorBehavior=1
```

Set environment variables

You must set the SDEHOME and SDE_DATABASE environment variables. Set SDEHOME to point to the directory where the client application's .dll files are stored.

If your client application is remote (is not running on the same host as the Informix server), edit the client machine's SDE_DATABASE variable in the dbinit.sde file so that it points to the remote database.

If your client application is local, set the client machine's SDE_DATABASE variable using system tools (do not use the dbinit.sde file to set this) to the name of the Informix database on the local machine that you want to connect the client to.

Connection syntax

There is a particular syntax to use when connecting with direct connect. For the Service (or instance) value,

```
sde:informix:<odbc dsn>
```

where

```
sde:informix
```

is a required part of the syntax and

```
<odbc dsn>
```

is the ODBC data source name used to setup the connection to the database server in step 3. If the client application is local (running on the same host as the Informix server), do not specify a value for Server. If the client application is remote, specify a Server value of remote.

Test the connection from the client application

Test the connection.

Index