ArcGIS® 9

ArcSDE® Configuration and Tuning Guide for SQL™ Server

Copyright © 1986–2005 ESRI

All Rights Reserved.

Printed in the United States of America.

The information contained in this document is the exclusive property of ESRI. This work is protected under United States copyright law and the copyright laws of the given countries of origin and applicable international laws, treaties, and/or conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage or retrieval system, except as expressly permitted in writing by ESRI. All requests should be sent to Attention: Contracts Manager, ESRI, 380 New York Street, Redlands, CA 92373, USA.

The information contained in this document is subject to change without notice.

RESTRICTED/LIMITED RIGHTS LEGEND

U.S. Government Restricted/Limited Rights: Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. In no event shall the Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (JUN 1987); FAR §52.227-19 (JUN 1987); and/or FAR §12.211/12.212 [Commercial Technical Data/Computer Software]; DFARS §252.227-7015 (NOV 1995) [Technical Data]; and/or DFARS §227.7202 [Computer Software], as applicable. Contractor/Manufacturer is ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

ESRI, MapObjects, ArcView, ArcIMS, SDE, and the ESRI globe logo are trademarks of ESRI, registered in the United States and certain other countries; registration is pending in the European Community. ArcSDE, ArcInfo Librarian, Spatial Database Engine, ArcCatalog, ArcToolbox, ArcMap, ArcGIS, ArcStorm, ArcInfo, ArcObjects, ArcExplorer, ArcEditor, and the ArcInfo logo are trademarks and www.esri.com is a service mark of ESRI.

The names of other companies and products mentioned herein are trademarks or registered trademarks of their respective trademark owners.

Contents

Contents iii

Getting started 1
Configuring SQL Server 1
Configuring dbtune storage parameters 2
Managing tables, feature classes, raster columns, and views 2
Connecting to SQL Server 2
National language support 2
Backup and recovery 3

Configuring SQL Server 5
Introduction 5
Installing and configuring Microsoft® SQL Server 6
Creating and managing databases and file groups 7
Managing security 11

Configuring Dbtune Storage Parameters 27
The SDE_dbtune table 27
Understanding keywords 31
Understanding parameter name—config string pairs 39

Managing tables, feature classes, raster columns, and views 49
Setting up the dbtune table 49
ArcSDE to Microsoft SQL Server data type mapping 49
Importing data 50
Managing data 56
Exporting data 58
Dropping data 58
Defining ArcSDE views 59
Choosing an ArcSDE log file configuration 62

Connecting to SQL Server 65
Connection modes 65
Making your first connection 66

Using the ArcSDE direct connect driver 67

National language support 71
SQL Server database collation designator 71
Character encoding standards supported by ArcSDE 73

Backup and recovery 75
Understanding backup and restore 75
Backing up ArcSDE spatial databases 77
Restoring and recovering ArcSDE spatial databases 81
Managing transaction logs 84
Moving data using backup and restore 87
Moving data using sp_detach_db and sp_attach_db 90

ArcSDE compressed binary 93
Compressed binary 93
The spatial grid index 96
Indexes 100

Storing raster data 109
Raster schema 112

The well-known binary representation 121
Numeric type definitions 121
XDR (big endian) encoding of numeric types 122
NDR (little endian) encoding of numeric types 122
Conversion between the NDR and XDR representations of WKB geometry 122
Description of WKBGeometry byte streams 122
Assertions for well-known binary representation for geometry 125

Storing locators 127
Locator schema 128

Making a direct connection 137
What files do you need? 137
How to get your database setup files 139
Environment variables 139
Client/database compatibility 141
Registration and authorization 141
Setting up clients for SQL Server direct connect 142

Storing XML Data 145
Configuring ArcSDE for SQL Server 145

ArcSDE XML columns database schema 149

CHAPTER 1

Getting started

Creating and populating a geodatabase is arguably a simple process, especially if you use ESRI's ArcCatalog or ArcToolbox to load the data. So why is there a configuration and tuning guide? While data loading might be relatively simple, the choices you as the ArcSDE database administrator must make about data storage, security, recoverability, and performance may be more complex.

This book provides instruction for configuring the logical and physical storage parameters of your data in SQL Server as well as important guidelines for performance and troubleshooting.

Configuring SQL Server

Building a geodatabase involves proper configuration of the SQL Server instance, management of the physical database files, and implementation of user security. Chapter 2, 'Configuring SQL Server' provides information on these three topics.

You will learn:

- Which SQL Server configuration parameters you might want or need to modify.
- How to use the Post-Installer to create a geodatabase.
- How to configure security settings for the 'SDE' and other users, on servers with Windows Authentication, or Mixed-mode authentication.

Configuring dbtune storage parameters

Every table and index in a database has creation parameters that control the physical storage within database files. These parameters include the type of index created (clustered or non-clustered), index fillfactor, placement on file groups, and text-in-row settings.

The storage configuration of ArcSDE data is controlled by settings in the SDE_dbtune table. Storage parameters are grouped under configuration keywords that you assign to data objects (tables and indexes) when you create them from ArcSDE client programs, such as ArcCatalog.

Chapter 3 'Configuring dbtune storage parameters' describes in detail the SDE_dbtune table and all possible storage parameters and default configuration keywords.

Managing tables, feature classes, raster columns, and views

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 a SQL Server database. With these tools, you can convert existing ESRI coverages and shapefiles into ArcSDE feature classes as well as import and export data between personal and ArcSDE geodatabases.

An alternative approach to creating spatial data in a SQL Server database is to use the administration tools provided with ArcSDE. These tools allow you to import tabular and spatial data from ArcSDE export files and raster datasets.

Chapter 4, 'Managing tables, feature classes, raster columns, and views' describes methods to create and maintain spatial data in a SQL Server database.

Connecting to SQL Server

Chapter 5, 'Connecting to SQL Server' covers how to make your initial connection to ArcSDE under a three-tiered configuration, how to connect in a two-tier configuration, and how to operate only in a two-tier environment without installing ArcSDE. Connections using ArcObjects and from ArcIMS are also covered.

National language support

Storing data in an ArcSDE SQL Server database using collation sets other than the SQL Server default, Latin1_General, requires some extra configuration on both the client and the server. Chapter 6, 'National language support', provides guidelines for configuring both the SQL Server database and the ArcSDE client environment to enable the use of collation sets other than Latin1_General.

Backup and recovery

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

Chapter 6, 'Backup and recovery' discusses SQL Server backup types, transaction log management, recovering database from backups, and using backup to move data between servers.

CHAPTER 2

Configuring SQL Server

Microsoft® SQL Server is easy to set up and configure. SQL Server has outstanding management tools and wizards to assist you in performing database functions. Due to its ease of use and rich user interface, there are many possible configurations with SQL Server. This chapter guides you through ArcSDE software-specific rules for creating and managing databases, logins, and transaction logs.

Introduction

This chapter picks up where a new ArcSDE installation leaves off. You've installed ArcSDE and let it create a spatial database and sde login/user for you. Now you must consider how you should configure SQL Server for ArcSDE, design databases, manage security accounts, and understand the Microsoft SQL Server transaction log.

To properly set up ArcSDE for Microsoft SQL Server you must:

1. Make some configuration changes to Microsoft SQL Server. These are detailed in the section entitled 'Installing and configuring Microsoft SQL Server'.
2. Design your database or databases. This is covered under the section 'Creating and managing databases and file groups'.
3. Design and configure your security. You can use either SQL Server or Windows Authentication. The section 'Managing security' describes this.

Installing and configuring Microsoft SQL Server

Installation of Microsoft SQL Server

1. **Use Mixed-Mode security:** Although you are not required to use Mixed-Mode security, setup is simpler if you do. Security modes can be changed within SQL Server through the server properties in the Enterprise Manager. Right-click your server in the Enterprise Manager, click Properties, then click the Security tab. Enable Mixed-Mode security. See 'Managing security' for details on how to use Windows authentication with ArcSDE.

2. **Collation:** Choose a collation that is appropriate for your culture. ArcSDE will work with most collations if you use the single spatial database configuration.

Microsoft SQL Server configuration

1. **Limit SQL Server memory consumption:** By default, SQL Server will use as much memory as it requires. On busy systems, SQL Server can force other applications, including ArcSDE, to page, compromising their response time and scalability. Consider limiting SQL Server's memory consumption by using a fixed memory size or setting a maximum memory size. To do this, right-click the server in the Enterprise Manager and click Properties. Within the Properties page, click the Memory tab and hard code a memory limit or set an upper boundary to 50 percent of total RAM. Adjust up or down as necessary.

2. **Use lightweight pooling:** On busy systems, configure SQL Server to use lightweight pooling or Windows NT Fibers. Fibers, which run as processes within a thread, reduce the number of thread context switches the processors must make. Access this setting in the Enterprise Manager's server properties under the Processor tab or through sp_configure. Search the Microsoft SQL Server Books Online for 'lightweight pooling.'

3. **Boost SharedSection in the registry:** This setting controls the heap size for noninteractive desktop applications—the amount of memory available to spawn and support Windows services, such as ArcSDE client connections. Adjust this setting if you anticipate having more than 50 concurrent ArcSDE connections. Locate this setting in the registry under: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems` under the Windows entry. This entry contains a string, in which is contained SharedSection=1024,3072,512. Boost the last entry to 1024. See the book *Managing ArcSDE Application Servers* for more information, or refer to the ESRI online support center (at http://support.esri.com).

4. **CPUs:** If your database will support many multiversioned editors, consider that versioned queries are CPU intensive. Consider buying more CPU than less. Make sure you consult the ESRI System Integration system sizing tools before you buy a server to support this type of system.

5. **Number of SQL Server Licenses:** This depends on which licensing mode you employ. If you choose client access licensing, you will need as many licenses as

users who will connect simultaneously, plus one for the ArcSDE application server (the ArcSDE service).

Support for SQL Server named instances

At Microsoft SQL Server 2000, you can install multiple SQL Servers onto the same server. Multiinstance SQL Servers are known as “named instances”. ArcSDE supports named instances in the same manner as a default instance.

To configure ArcSDE to work with a named instance, do *one* of the following:

1. During the ArcSDE Post-Install, enter the named instance whenever you are prompted for a data source or SQL Server instance name.
2. Use the sdeservice -o create tool, located in %sdehome%\bin to create a service pointing to a particular named instance of SQL Server.

Creating and managing databases and file groups

Once you have installed the ArcSDE software and made some SQL Server configuration changes, you can design databases to support your applications. You can either use a single geodatabase or several geodatabases to store your data. You can employ file groups and files to create a simple non-Redundant Array of Independent Disks (RAID) striping strategy or employ a disk array for full data striping.

ArcSDE single spatial database model

You can store a geodatabase in a single SQL Server database, or a geodatabase can span multiple databases. At ArcSDE 9, the default is to use a single database as a container for a geodatabase. We refer to this as the 'single spatial database model'.

ArcSDE rules for a single spatial database:

1. 1 database = 1 geodatabase = 1 ArcSDE service.
   Each ArcSDE spatial database requires its own ArcSDE service.
2. No cross database querying is allowed, except through functions that allow user input SQL, such as Se_stream_prepare_sql or IWorkspace.ExecuteSql.
3. An sde user must own the sde metadata.

ArcSDE multiple spatial database model

If you wish to store data from a single geodatabase in more than one SQL Server database, you must create a database called 'sde' to store the geodatabase metadata. You can then create additional SQL Server databases to store feature classes, raster datasets, and non-spatial tables. These additional databases are permanently related to the central 'sde' database.

ArcSDE rules for a multiple spatial database"

1. 1 'sde' database (+ n additional databases) = 1 geodatabase = 1 ArcSDE service
   There can only be one Multiple Spatial Database configuration per SQL Server instance as this model is dependent on a database called 'sde'.
2. Cross database querying is permitted. Each connection must specify a database or the user will be connected to the 'sde' database. By default, all objects in all related databases will be listed. For controlling this behavior, see the *CROSS_DB_QUERY_FILTER parameter* topic in Chapter 3.
3. An sde user must own the sde metadata.
4. The sde user must be added to each database that will store spatial data
5. Each person who connects to this geodatabase must be a user in the sde database. Required statement (DDL) permissions will vary depending on log file configuration. For more information on log files see *Choosing an ArcSDE log file configuration* in Chapter 4.

How many geodatabases?

You can either put all your spatial data into a single geodatabase or use multiple geodatabases to store your spatial data. (Remember, a multiple spatial database is considered one geodatabase.) If you use more than one geodatabase, you have to create a single ArcSDE service for each. If you create multiple ArcSDE services, consider increasing SharedSection in the Windows registry (see *Boost SharedSection in the Registry* on the previous page).

Client-side considerations

ArcIMS—If ArcIMS® servers see a different database or user in an SDEWORKSPACE connection string, then a new set of gsrvrs will be spawned to support that map service. If you have a distributed ArcIMS solution with multiple ArcIMS servers hosting many virtual servers, you could create so many gsrvrs that Windows will exhaust its noninteractive desktop memory heap. Increase the SharedSection parameter listed above under Microsoft SQL Server configuration.

ArcGIS—ArcGIS® read/write clients intensively query and write to the sde_states, sde_state_lineages, and sde_mvtables_modified. Any selection from an ArcGIS client writes to an sde log file if the selection set contains more than 100 features. Set a high SelectionThreshold in each ArcGIS client's registry to store more selected features locally instead of on the sde server. Setting this to 1,000 features is reasonable for many client systems. Keep in mind that this setting requires more processing on the client side, so make sure you have enough memory and CPU to handle it. Increase the SharedSection parameter if you plan on serving more than 50 simultaneous ArcGIS connections.

Guidelines for database creation

Here are some guidelines for creating a geodatabase.

1. Use the Post-Installer to create a spatial database. The Post-Installer will create the spatial database, grant the sde login access to it, and install the ArcSDE system tables.
2. Size the data files large, then use the Enterprise Manager to increase the autogrowth increment of both the database and transaction log file. While you're at it, make a backup of it.
3. Store all your data files, transaction log files and tempdb separately from the paging file, unless you're sure your server will never page. Separate your data files from your transaction log files and tempdb.
4. If you have a RAID 5 solution, keep your transaction log files and tempdb out of the disk array. In general, put data files on fault-tolerant devices and tempdb and transaction log files on mirrored or standard devices.
5. Leave AUTO_CREATE_STATISTICS and AUTO_UPDATE_STATISTICS enabled. Disable autoshrink.
6. Employ a hardware striping solution, favoring RAID (disk array) over file groups and files.
7. Employ data segregation strategies (keeping tables from indexes or certain types of tables from other tables) only if you are certain it will improve performance or alleviate administrative burdens.

NOTE: If you wish to create a multiple spatial database (one central sde database and additional databases) only create the 'sde' database using the Post-Installer. Use SQL Server to create additional databases.

Hardware striping solutions

Large production ArcSDE systems should employ a hardware striping solution. Your best disk and data organization strategies involve spreading your data across multiple disks. With data spread across multiple disks, more spindles actively search for it. This can increase disk read time and decrease disk contention. However, too many disks can slow down a query. There are two main ways of achieving striping: file groups and RAID. You can also combine the two—create file groups within disk arrays.

File groups and files

File groups and files compose a database. Data resides in files that are grouped by file groups. The database's primary file of the primary file group has a *.mdf extension, while secondary files use *.ndf. Log files are stored in *.ldf files. A file group arranges files into administrative units. Database files cannot span disks, file groups can.

Files fill proportionately according to their membership in a file group. For example, if three files comprise a file group, and file A is 20 MB, file B is 40 MB, and file C is 60 MB, one extent is allocated from file A, 2 from B, and 3 from C whenever space is requested from a file group. All database objects are created by default on the primary file group. Placement onto other, nondefault file groups is controlled with the "on" keyword of the create statement. ArcSDE controls placement through the use of the SDE_dbtune table in the same format—by adding the keyword "on" to your storage parameters.

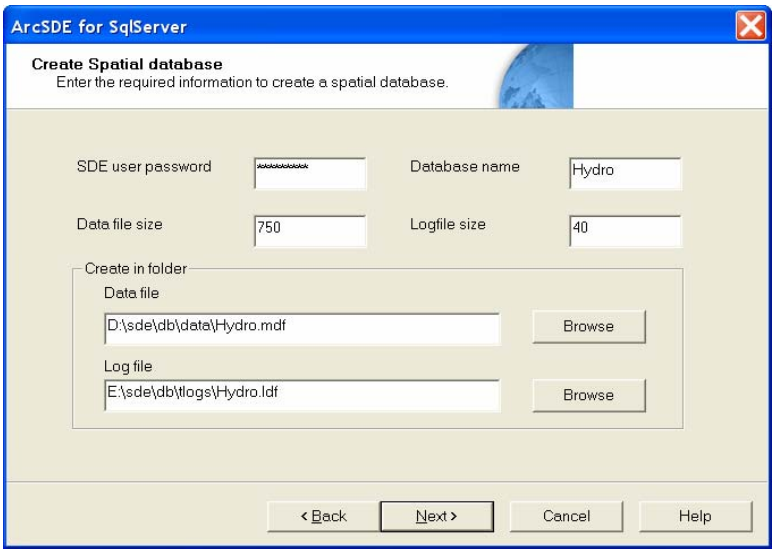
Not all ArcSDE installations will be on multiprocessor server class machines with disk arrays or many disks. ArcSDE is certified to scale from Microsoft Data Engine (MicrosoftDE) to SQL Server Enterprise Edition. If you are serving data to a few clients on a single processor machine with one or two disk drives, employing file groups offers no advantage.

RAID

Redundant Arrays of Inexpensive or Independent Disks (RAID) boosts fetching performance by striping data into slices across multiple disks in the 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. A thorough discussion of RAID is beyond the scope of this document.

Example: Creating a geodatabase with the ArcSDE Post-Installer

1. Start the Post-Installer: Start > Programs > ArcGIS > ArcSDE > ArcSDE for Microsoft SQL Server Post Installation.
2. The Post-Installer starts. Click the Custom button and click Next.
3. The Select ArcSDE Setup Wizard Option form appears. With all check boxes checked, click Next.
4. The User Information form appears. Input the correct SQL Server instance name (for example, Ocean or Ocean\instance01). Pick a security mode and click Next.
5. The Create Spatial database form appears.



6. Input your sde user password, even if you don't have an sde login created. The Post-Installer will create one for you.

   NOTE: The Post-Installer will only create a SQL Server authenticated sde login.

   Input a database name, data file size, transaction log file size, and paths to your data and transaction log files. Click Next to create the database.

7. Click Next in the ArcSDE Configuration Files form.

8. The next form, User Information—Connect to create SDE repository, will install or upgrade the ArcSDE system tables. Pick an authentication type and click Next, and the sde system tables will be created or upgraded.

9. Complete the software authorization forms and click Next until you get to the Create the ArcSDE Service form. Input a service name, port number, sde user password, spatial database name (use the same name as the database you just created), and the rest of the required information.

Managing security

Once you have designed and created your databases, you now must configure your security accounts. Create logins and users with specific functionality in mind, assign them to roles, and finally, grant permission to those roles to interact with data.

Introduction to SQL Server security

Microsoft SQL Server has logins and users. A login is a named account that has access rights to the database server but not necessarily access to a database. A user is a login that has been granted access to a database.

There are two types of authentication: SQL Server authentication and Windows authentication. SQL Server authenticated accounts are valid only in SQL Server. Windows authentication uses operating system accounts.

Microsoft SQL Server also supports roles. Roles are used to group users. Statement and object permissions are granted to roles, and each user in a role inherits those permissions. There are three types of roles: fixed server, database, and user-defined. Fixed server roles span the server and apply to all databases. Database and user-defined roles pertain to a specific database.

Logins and users: Rules for the single spatial database model

1. The sde user must have create table, create view, create procedure, and create function privilege within the spatial database.
2. Data owners must have create table and create procedure privilege in the spatial database.
3. A login's default database is used only when connecting with direct connect and not specifying a database argument.
4. Cross database querying capability is disabled for all functions except those that allow user-defined SQL queries (SE_stream_prepare_sql, IWorkspace.ExecuteSql, SeConnection.PrepareSql()).
5. Use of a separate account for each user is no longer required if you use session or stand-alone log files. However, it is still recommended.
6. You cannot use a SQL Server reserved word for a login or username. Examples of reserved words are Max, National, and Count. See the SQL Server Books Online for a complete listing.

ArcSDE users rule matrix

User	In Spatial database
sde User	create table, create view, create procedure, create function (SQL Server 2000)
Other User	create table, create procedure if will own data

All connecting users will require create table permission in the geodatabase unless your ArcSDE service uses the new session-based log file functionality. To enable the new session-based log file functionality, set these parameters using the sdeconfig.exe tool:

```
ALLOWSESSIONLOGFILE TRUE
LOGFILEPOOLSIZE     #       this value will depend on the number of simultaneous connections
HOLDLOGPOOLTABLES   TRUE
```

Set this config string in your sde_dbtune table:

```
LOGFILE_DEFAULTS    SESSION_TEMP_TABLE              1
```

Logins and users: Rules for the multiple spatial database model

1. The sde user must have create table, create view, create procedure, and create function privilege within the sde database.
2. Data owners must have create table and create procedure privilege in the databases where they will create data (feature classes, raster datasets, non-spatial tables)
3. A login's default database is used only when connecting with direct connect and not specifying a database argument. When making an application server connection, if no database is specified, a connection will be made to the sde database.
4. Cross database querying capability is enabled.
5. Use of a separate account for each user is no longer required if you will use the new sde log file functionality. However, it is still recommended.
6. You cannot use a SQL Server reserved word for a login or username. Examples of reserved words are Max, National, and Count. See the SQL Server Books Online for a complete listing.

ArcSDE users rule matrix

User	In sde database	In other database(s)
sde User	create table, create view, create procedure, create function	sde must be a user in each additional database but no statement permissions required
Other User	create table, create procedure if will own data, all users require create table is using shared log files	create table, create procedure if will own data

All connecting users will require create table permission in the sde database unless your ArcSDE service uses the new session-based log file functionality. To enable the new session-based log file functionality, set these parameters using the sdeconfig.exe tool:

```
ALLOWSESSIONLOGFILE TRUE
LOGFILEPOOLSIZE     #       this value will depend on the number of simultaneous connections
HOLDLOGPOOLTABLES   TRUE
```

Set this config string in your sde_dbtune table:

```
LOGFILE_DEFAULTS    SESSION_TEMP_TABLE              1
```

Designing your security

Setting up your security follows this general process:

1. Create logins and assign those logins to databases. The logins become users.
2. Create database roles or use existing roles and add users to them.
3. Grant statement or object permission to those roles.

Security considerations

1. For simplest administration
   a. Use SQL Server authentication only.
   b. The sde user must have create table, procedure, view, and function in the spatial database. Adding the sde login to the sysadmin fixed server role is not supported.
   c. Create individual SQL Server logins and add them to roles within a spatial database. If they need read permission to all tables, add them to the db_datareader role within the spatial database.
2. For tightest security
   a. Configure ArcSDE and SQL Server to use only Windows logins.
   b. Grant no permissions on data to the Public role.

c. Do not add any login to the db_owner database role.

d. Never save your username and password in an ArcGIS connection file.

e. Use direct connect.

3. Preserve the sde login as the ArcSDE system administrator; do not use it for other operations. Do not add the sde login to the sysadmin fixed server role.

Using SQL Server authentication

SQL Server authentication is easiest to administer but less secure than Windows authentication. This section demonstrates creating new SQL Server logins with both the Enterprise Manager and Transact-SQL, granting these logins access to databases, adding these users to database roles, and finally, granting statement or object permission to these roles.

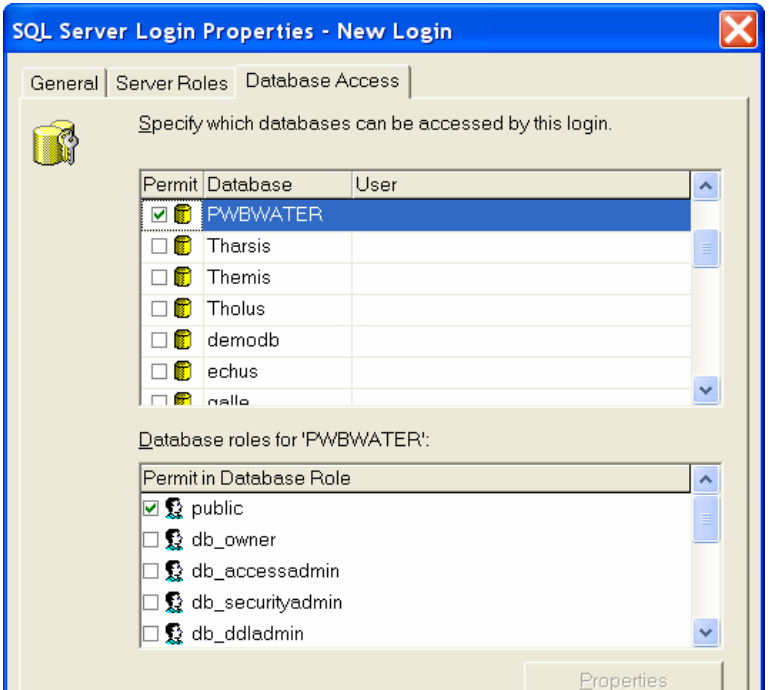
Enterprise Manager—Create logins and grant database access

1. Open the SQL Server Enterprise Manager, expand your server's node, expand the security node, and select the logins node.

2. In the details panel on the right, right-click and click New login. Click the SQL Server Authentication button, name the login, and provide a password. Assign a default database.

3. Click the Database Access tab. In the Permit column, check the desired spatial database. In the form below, a login is granted access to the PWBWATER database.

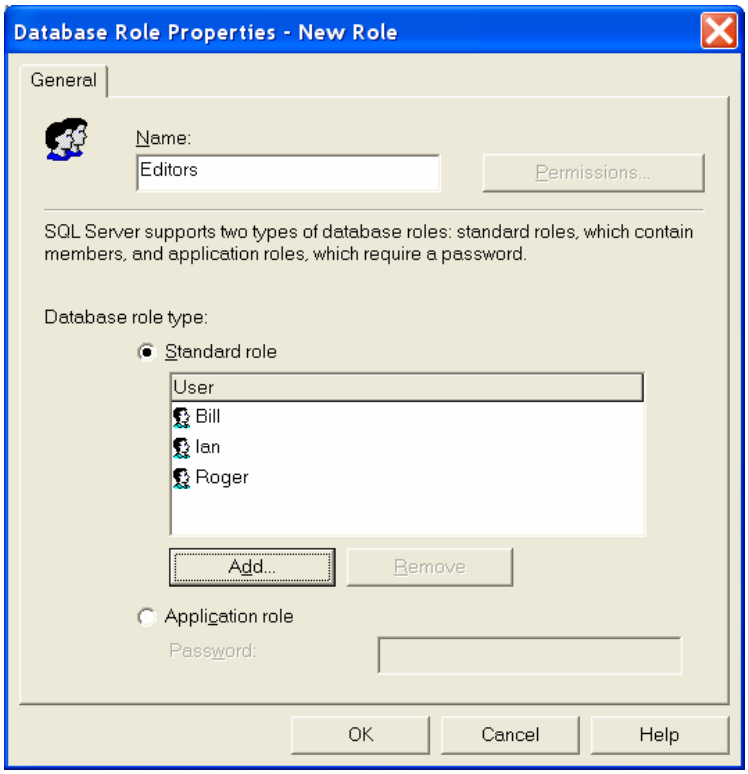
4. Click OK and you'll be prompted to confirm your password. The process both



creates the new login and grants it access to a database or databases. (In other words, it adds that login as a user to a database).

Enterprise Manager—Create a role and add users

1. Click the databases node in the Enterprise Manager. Expand your database. Click the roles node.
2. Right-click within the Details pane on the right side, click New database role, Name your role. Click Add and add users.



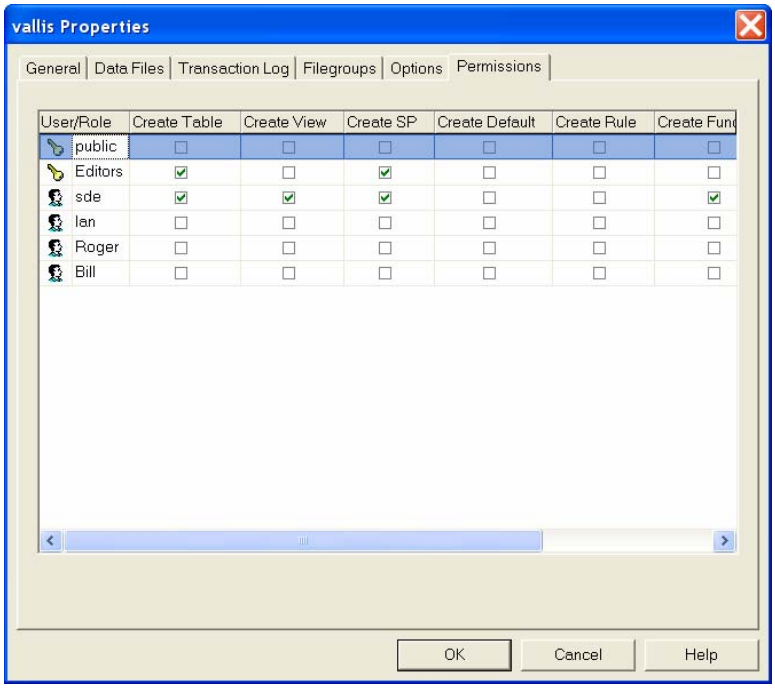
3. Click OK.

Enterprise Manager—Grant statement permissions to a role

Statement permissions are the right to create, alter, or drop something. ArcSDE data owners must have create table and create procedure privilege. To grant statement permissions to a role:

1. Expand your databases node in the Enterprise Manager.
2. Right-click your spatial database > click Properties > click the Permissions tab.

3. Locate your role in the User/Role column of this form. Check the desired statement permission. In the form below, the Editors role has been granted Create Table and Create SP.



In this example, Ian, Bill, and Roger are all members of the Editors role. These accounts inherit the right to create tables and procedures through membership in the Editors role.

Grant object permissions to a role

Object permissions are the right to access data through select, insert, update, or delete statements. For ArcSDE, this is best done through either the sdelayer -o grant command or ArcGIS. Using the Editors role from above, this example will grant all members of the Editors role the right to query and edit features in the buildings feature class:

```
sdelayer -o grant -l buildings,shape -U Editors -A
SELECT,INSERT,UPDATE,DELETE -i esri_sde -D spatialdb -u dataowner
```

From ArcCatalog™: Connect to your spatial database, find the buildings feature class, right-click it, then click Privileges. Type the role name in the form and check a permission.

Using Transact-SQL

Scripting the creation of your logins and users is much faster than using the Enterprise Manager. The same rules apply as with the Enterprise Manager. Below is a SQL script that will demonstrate this.

```
/*
        Add sde and non-sde login and user
*/

/*
        Step1: Add the logins to the server using
        the sp_addlogin stored procedure.
        sp_addlogin '<login_name>','<pass>','<default_db>'
*/

use master
go
sp_addlogin 'sde','spatial.data','spatialdb'
go
sp_addlogin 'Ian','design_nature','spatialdb'
go

/*
        Step2: Grant the login access to the spatial database
        using sp_grantdbaccess. Grant the user create table
        and stored procedure. You must switch to the spatial
        database to do this.
*/

use spatialdb
go
sp_grantdbaccess 'sde'
go
sp_grantdbaccess 'Ian'
go
grant create table to sde
grant create procedure to sde
grant create view to sde
grant create function to sde  --SQL Server 2k only!
Go
/* create a role called Editors */
sp_addrole 'Editors'
go
sp_addrolemember @rolename='Editors', @membername='Ian'
go
/* grant statement permissions to Editors */
Grant Create table to Editors
Grant Create Procedure to Editors
```

Using Windows authentication

Windows authentication offers some advantages over SQL Server authentication. Windows authentication is generally more secure than SQL Server authentication since it uses a certificate-based security mechanism. It is usually easier to configure since you can add existing Windows groups, containing Windows logins, to the SQL Server.

Finally, at connection time, the user is generally not required to enter a username and password. A single sign-on at login time provides access to all services that support Windows authentication.

ArcSDE 9 for Microsoft SQL Server supports Windows authentication with some restrictions:

1. Windows groups are supported for data manipulation language (DML) (select, insert, update, delete) operations only. Members of Windows groups cannot create data through ArcSDE.
2. If you must have Windows users create data, then those users must be granted access individually (not through a Windows group) and cannot have a domain name in their database username. If your login name is TERRA\bob, then your database username must be "bob".
3. You cannot connect through ArcSDE as a different Windows user than your present login. If you logged in as TERRA\Ian, you cannot make a Windows authenticated connection as GLOBE\Ian.
4. You cannot restrict access to the %SDEHOME%\bin directory on the server if you connect through the application server. The application server spawns processes in the security context of the connecting Windows user. To correctly spawn the process, this directory must not be restricted to the connecting user. This is not a concern for direct connect Windows authenticated connections.
5. Not all ArcSDE clients support single sign-on, meaning you'll have to enter your username to connect to ArcSDE services. At ArcSDE 9, only the ArcSDE administration commands support single sign-on. ArcGIS requires you to submit a username and password.

Because Windows authentication can add some complexity to managing an ArcSDE for SQL Server instance, it is recommended that novice ArcSDE users stick to SQL Server authentication until they are more comfortable with the product.

Using Windows groups with ArcSDE

Windows groups are supported only for DML (select, insert, update, delete) operations. Members of Windows groups cannot create data in their SQL Server database through ArcSDE. Follow this process to use Windows groups in ArcSDE:

1. Add the Windows group as a login to SQL Server:
    a. Enterprise Manager—Open your SQL Server Instance > Security branch > right-click logins > add new login. Click the ellipsis button next to the name text box to open the domain user browser form. Select your domain in the List Names From combo box and scroll to the Windows group listed in the Names box. Select your Windows group, click add, then click OK. Back in the SQL Server Login Properties > New Login form, click the Database Access tab. Select the database to which you want this group to have access. Verify that the group name does not contain the DOMAIN\. Click OK.

b. Transact-SQL—Open the Query Analyzer or OSQL and execute this statement:

```
exec sp_grantlogin @loginame='TERRA\gis_users'
```

2. Grant the Windows group database access:

   a. Enterprise Manager—See 1.a above or expand your database node, right-click the users item, scroll the Login name combo box, and select your Windows group. Once you select your Windows group, it will automatically populate the user name field. If, for example, your Windows group name is "TERRA\gis_users", the user name field will list "TERRA\gis_users". If you will add this Windows group to an existing SQL Server role, click the rolename from the Database role membership box and click OK. If you will not add this Windows group to an existing SQL Server role, remove the domain name and backslash ("TERRA\") from the user name field.

   b. Transact-SQL—Grant the Windows group database access using the sp_grantdbaccess system stored procedure. Run the command from the Query Analyzer or OSQL:

```
exec sp_grantdbaccess @loginame='TERRA\gis_users',
@name_in_db='gis_users'
```

3. Grant permission to your data to the Windows group. If you've created your data with ArcGIS, use ArcCatalog to grant permission. If you created your data with the sde command line administration tools, use the sdelayer or sdetable commands. To grant permission directly to a Windows group, the group name cannot contain the DOMAIN\ within it. See points 1 and 2 above. Alternatively, you can create a SQL Server role, add the Windows group to the role, and grant permission to the role.

   a. Using sdelayer:

```
sdelayer -o grant -U gis_users -A SELECT,INSERT,UPDATE,DELETE -i
5151 -D CityDb -u gisdataowner -p go -l parcels,shape
```

   b. Using ArcCatalog: Connect to ArcSDE as data owner, right-click the dataset to which you'll grant permissions, and click Privileges. Type in the group name or role name, then select the desired permission level (SELECT, INSERT, UPDATE, DELETE). Click OK.

      NOTE: If you wish to grant permission directly to a Windows group, that group name cannot contain the DOMAIN\ in its database username.

Windows accounts and data ownership

If you want a Windows account to create and own data through ArcSDE, you must add that account individually to SQL Server. Windows accounts that access SQL Server through group membership cannot own data because that account's database username contains the DOMAIN/ name and "\". This naming convention will cause problems within the database when creating tables. If you want a Windows account to create and own data, the login's database username cannot contain the DOMAIN\. For example, Windows account "TERRA\Ian" must only be "Ian" in a database.

Connecting to an ArcSDE service as a Windows login

When you connect with Windows authentication to an ArcSDE service, you must be presently logged in as that user. If not, your connection will fail.

With some ArcSDE clients, you can connect to an ArcSDE service without using a username or password. The examples below use the administration commands to demonstrate this functionality.

sdelayer:

```
C:\>sdelayer -o describe -i 5151

Database : VTEST
Table Owner : VTEST
Table Name : SANDIEGO_NET_JUNCTIONS
Spatial Column : SHAPE
Layer id : 8
Entities : npc
Layer Type : SDE
I/O Mode : NORMAL
User Privileges : SELECT, UPDATE, INSERT, DELETE
Layer Configuration: DEFAULTS
```

sdetable:

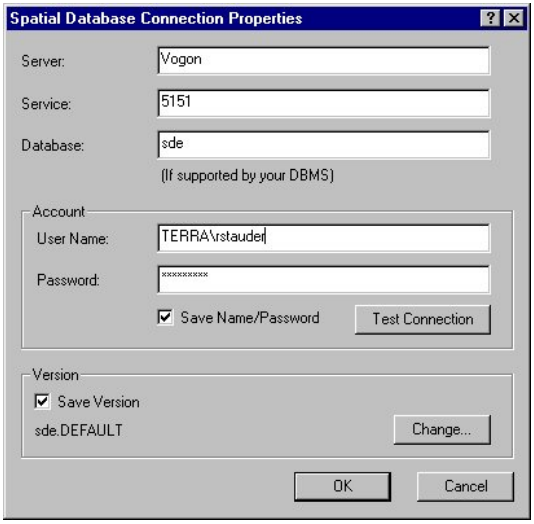
```
C:\>sdetable -o create -t authentication_test -d "colA string" -i 5151

Successfully created table authentication_test.
```

sdeimport:

```
E:\world_sdex>sdeimport -o create -l test,shp -i 5151 -f roads

Importing SDEX from roads ...
1165 features read.
1165 features stored.
```

ArcGIS clients must submit a username and password. The ArcSDE C application programming interface (API) (used by ArcGIS) compares the submitted username against the presently logged in user. If these two differ, the connection is rejected. For example, if you log in to GLOBE\bob but try to connect as user TERRA\bob, your connection will fail. The password is not used since authentication occurs within the operating system. However, you are still required to submit one.

When using Windows authentication from ArcGIS, all login forms will require the domain\username convention. Other forms, such as the Privileges form, require only the username. Following is an example of a Windows authenticated login from ArcGIS.



Using Windows sde login

You can use a Windows sde login instead of a SQL Server authenticated sde login. Keep in mind that the Post-Installer will always create a SQL Server sde authenticated login. Although you can do this, you may experience difficulties with some operations, such as compress.

To use an integrated sde login:

1. Create a Windows sde login. Add this login to the SQL Server logins collection on your SQL Server instance that will host ArcSDE. This account will be the logon account for the sde service. This account must be granted these security policies on the server: replace a process-level token, act as part of the operating system, and create a token object.

2. Create a spatial database. Add the sde login to this database as a user.

   NOTE: The sde username must be "sde" and cannot contain a domain name such as "TERRA\sde".

   Grant this user create table, view, stored procedure, and function privileges.

3. Start the ArcSDE Post-Installation Utility. When the application opens, click the Custom button. Click Next. The Select ArcSDE Setup Option Wizard form opens. Deselect the first option to Define Database and sde user and click Next.

4. The User Information form opens. Verify that the server name is correct and change the database name to the name of the database you created. Specify a login type and click Next. The Post-Installer will begin to create the ArcSDE system tables, stored procedures, view, and functions. Click Next to authorize your software.

5. Once your software is authorized, proceed to the ArcSDE Service Information Form. In this form, type your service name and port. Use "" for the sde user's password.

Type the database name you created in step 2 and verify that the remaining information is correct. Click Next. Do not start the ArcSDE service.

6. Open the Windows Services panel, double-click the service you just created. Click the Logon tab. Click the This Account button, then browse to the Windows sde account. Type the correct sde user password, click OK, and start your service.

NOTE: It is recommended that if you employ a Windows sde user and you wish to use Windows authenticated logins, you should use only direct connect for those connections.

Using roles

Once you have created logins and granted them user access to databases, you should create roles and assign those users to roles. Add your users to roles based on operations they'll perform. For example:

- Viewers: users that have read-only access to data
- Editors: users that will edit data in a database
- Developers: users that will create and deploy applications based on data in a database

There are four types of roles: database, user-defined, fixed server, and application. Application roles are not supported. Database roles are preexisting, SQL Server-defined roles. User-defined roles are created by an administrator within a database. Fixed server roles pertain to all databases.

Adding a database user to a role simplifies administration by reducing the number of things you have to do per user. If you define a role that has create table and procedure privileges and add all your ArcSDE users to it, you have reduced the number of times you have to grant a privilege to each one.

Create a user-defined role with `sp_addrole` or the Enterprise Manager. For example, `sp_addrole 'Editors'` will create a user-defined role "Editors" owned by the dbo user. You would now add users to this role with the Microsoft SQL Server Enterprise Manager or the `sp_addrolemember` stored procedure:

```
sp_addrolemember 'Editors','Ian'
```

Using database roles

Database roles are preexisting roles that carry implicit privileges and permissions. Public is a database role that all users belong to. Exercise extreme caution when using database roles, as they can produce unexpected side effects. The advantage of using database roles is that they carry implicit permissions and privileges, resulting in less work.

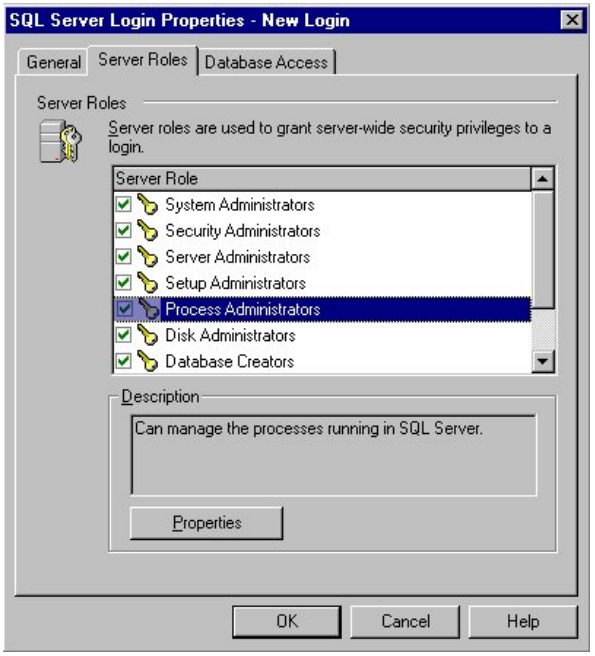
When to use database roles:

- Db_datareader: Grants/Select permission on all tables in a database to a role member.

- Db_ddladmin: Permits a user to run any Dynamic-Link Library (DDL) statement in a database.
- Db_owner: Makes a user sysadmin of a database without changing the username to "dbo".

Fixed server roles

Fixed server roles pertain to the server and, hence, all databases. There are two main issues to consider: if you add a user to the sysadmin fixed server role, anything that a user creates in the database is owned by "dbo" in any database; you can add a user to multiple, sometimes conflicting, roles. This last point will assuredly cause problems when you connect and attempt to view (or load) your data. It is easy to do this, so make sure you don't. The following screenshot shows SQL Server 2000 doing this. Be careful you don't do this!



*Adding a login to all fixed server roles—**Don't do this!***

Granting permission to data to those roles

Once you've created logins, added them as users to databases, and added those users to roles, it is either time to load data or grant permission on that data to the roles. Chapter 4 covers data loading; but before you can load and grant permissions, you must decide who should own the data and, henceforth, do the granting.

Who should own the data?

Designate one user as the owner of all your data. Load all your data as that user and grant permissions to that data to roles. The advantage of this system is that your database will not get cluttered with tables belonging to several different users. If you load all your data and build ArcGIS complex features, then add your data owner to the sysadmin fixed server role, you may cause problems maintaining those complex features.

Granting permissions

If you loaded your data using the ArcSDE command tools, use the sdelayer -o grant or -o revoke command to grant or revoke a privilege (select, insert, update, delete) to a role. For example, you have an "Editors" role in the spatial database. To grant select, insert, update, delete permissions against an ArcSDE feature class, 'county_hiways' use the following command

```
sdelayer -o grant -U highways_editors -A SELECT, INSERT, UPDATE, DELETE -l
county_hiways,shape -u county -p big.bob -i esri_sde -s vogon
```

If you used ArcGIS to load data, you must use ArcGIS to grant permission to that data. Right-click the feature class, table, raster dataset, or feature dataset in ArcCatalog and click Privileges.

NOTE: A user or role must have insert, update, and delete privileges to make multiversioned edits.

CHAPTER 3

Configuring Dbtune Storage Parameters

ArcSDE feature classes, rasters, and non-spatial tables are made up of one or more tables stored within a SQL Server database. When these objects are created, ArcSDE submits Create table and Create index statements directly to the underlying database. These statements are formulated by ArcSDE and are based in part on input provided by the user at the time of the table's creation.

Create table and Create index statements have optional arguments that are used to control the physical placement and structure of the object. These arguments are used to place an object on a specific file group or to control the structure of an index. You can customize the statements created by ArcSDE to include these optional arguments through the use of a *configuration keyword*, specified at load time. This configuration keyword references customization parameters stored in an ArcSDE metadata table, *SDE_dbtune*.

The SDE_dbtune table allows a database administrator to control data placement, index architecture, and binary data storage. This chapter explains the purpose of SDE_dbtune and the keywords, parameters, and configuration settings it contains.

The SDE_dbtune table

The SDE_dbtune table is created by the ArcSDE postinstallation wizard, or the sdesetupmssql.exe command. The SDE_dbtune table has the following definitions:

Column name	Data type	Length	Allows null
keyword	varchar	32	No
parameter_name	varchar	32	No
config_string	varchar	2048	Yes

After the SDE_dbtune table has been created, it is automatically populated with the contents of the dbtune.sde file, located in the %SDEHOME%\etc directory. If an SDE_dbtune table already exists, the postinstallation may update some of its contents.

The keyword field stores keywords referenced when creating data in the database. Within each keyword there are a number of parameters, each stored in the parameter_name field. Each parameter has a configuration string associated with it, stored in the config_string field.

The SDE_dbtune table is used for the following:

1. Loading tables and indexes onto specific file groups
2. Specifying an index created as either clustered or nonclustered
3. Dictating the percentage to which an index page is filled
4. Specifying the amount of binary data that can be stored in data pages (as opposed to image pages)
5. Setting specific server-wide parameters

How it works

An ArcSDE feature class or raster dataset is made up of a business table and several supporting tables. When each table is created ArcSDE issues Create table and Create index commands directly to SQL Server for execution. ArcSDE formats these statements based on inputs specified during the loading process, such as table name and field definitions. The SDE_dbtune table stores additional parameters that can be used to further customize the creation of ArcSDE objects. These parameters are normally controlled by the database administrator as they impact the physical storage of data. The database administrator would configure values in the SDE_dbtune table. A keyword is then specified during the loading process, and ArcSDE reads related values from the SDE_dbtune table to determine how to create tables and indexes. The values read from SDE_dbtune are appended to Create table and Create index statements.

For example, you will load a raster dataset into an ArcSDE spatial database. By default, ArcSDE will fill index pages to 75 percent capacity. In this example, the raster will not change once loaded so you decide you'd like to fill each index page to 100 percent capacity, thus reducing table size and the total number of index pages. You also want to place this index on a special file group, RAS_FG. You add a keyword, RASTERS, to the SDE_dbtune table that instructs ArcSDE to apply these nondefault values.

The RASTERS keyword has a parameter, BLK_INDEX_COMPOSITE, which allows you to specify an index fillfactor and file group for the composite index of the raster's block table. The entry in the SDE_dbtune table would look like this:

Keyword	Parameter_name	Config_string
RASTERS	BLK_INDEX_COMPOSITE	With fillfactor = 100 on ras_fg

You would reference this keyword by specifying a Configuration Keyword in ArcCatalog or by using the -k switch of the sderaster command, for example,

```
sderaster –o import –l ColoFloodPlain,image –g –f  cfp002.tif –k rasters…
```

When you execute the above command, the loader reads the -k RASTERS keyword and creates the block table's composite index with a fillfactor of 100 percent on the file group 'ras_fg'. The Transact-SQL statement that is issued to SQL Server would look like this:

```
CREATE UNIQUE CLUSTERED INDEX [SDE_blk_4_pk} ON [dbo].[SDE_blk_4]
([rasterband_id], [rrd_factor}, [row_nbr], [col_nbr]) WITH FILLFACTOR =
100 ON [ras_fg]
```

Parameter names not listed under the RASTERS keyword are read from the DEFAULTS keyword. Parameter_names will be redundant in this table; each keyword can specify a parameter already listed under DEFAULTS.

The rules for the SDE_dbtune table are:

1. SDE_dbtune is used to control data placement, index fill factors, index clustering, and binary data in lining.
2. If you don't specify a keyword when you load data, you use settings defined under DEFAULTS.
3. If you supply an incomplete keyword, missing parameters and config_strings are read from the DEFAULTS keyword.
4. You are not required to make any additional edits to SDE_dbtune to make the software work.
5. You can add new keywords with any valid name. You must use the parameter_name and config_string values supplied in this chapter. You cannot invent new parameter_names and config_strings.
6. You can modify existing config_string values.

Editing the SDE_dbtune table

Administrators can add keywords and change config_strings in SDE_dbtune. You can edit the contents of the SDE_dbtune table with the SQL Server Enterprise Manager, SQL statements, or the sdedbtune command.

To edit the SDE_dbtune table with the SQL Server Enterprise Manager, find the table in the tables list within the database node. Right-click the table and click Open table, then

click Return all rows. You can also select the table, click the Action menu, click Open table and click Return all rows. Make changes to config_strings by clicking within a cell and entering new values. Press return to commit the change. To insert new dbtune records, scroll to the end of the table and click in the cell to the right of the asterisk. Keyword values cannot be null.

Using Transact-SQL statements is more complex, as you'll have to be familiar with the insert, update, and delete statements. These topics are beyond the scope of this book. Consult Microsoft SQL Server Books Online for more information.

The sdedbtune command exports the contents of the SDE_dbtune table to a file. The file can then be edited and reimported into the SDE_dbtune table. In the following example, the SDE_dbtune table is exported, edited, and reimported into the source table.

This command creates a file, dbtune.out, in the %SDEHOME%\etc folder.

```
C:\>sdedbtune -o export -f dbtune.out -s theodolite -D sde -u sde
```

This command opens the dbtune.out file in WordPad.

```
C:\>write %sdehome%\etc\dbtune.sde
```

After editing, the file is reimported to the SDE_dbtune table using the following command:

```
C:\>sdedbtune -o import -f dbtune.out -s theodolite -D sde -u sde

ArcSDE   8.1      Build 664 Tue Dec 26 22:32:22 PST 2000
Attribute        Administration Utility
--------------------------------------------------
Import SDE_dbtune Table. Are you sure? (Y/N): y

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

The sdedbtune command always exports to and imports from the etc directory of the ArcSDE home directory.

Editing a dbtune file

The sdedbtune -o export command creates a text file in the %SDEHOME%\etc. This text file, which can have any name, contains the contents of the SDE_dbtune table at the time it was exported. It has been formatted for easy editing.

Each keyword appears only once, prefaced by two pound (#) signs:

```
##DEFAULTS
```

After each keyword comes a list of parameter_names and their config_string values:

```
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER              0
B_INDEX_ROWID               "WITH FILLFACTOR = 75"
```

```
B_INDEX_SHAPE                "WITH FILLFACTOR = 75 ON FG1"
```

The parameter_name listings are followed by the END keyword:

```
##TESTKEYWORD
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER              0
B_INDEX_ROWID               "WITH FILLFACTOR = 75"
B_INDEX_SHAPE               "WITH FILLFACTOR = 75 ON FG1"
END
```

Each keyword and list of related parameter_names must be followed by the END keyword. If the END keyword is missing from the end of any of the keyword listings, the file cannot be reloaded into the SDE_dbtune table.

Each parameter_name and its config_string are delimited by either spaces or tab keys. Additional empty lines or line feed characters are ignored by ArcSDE. Any nonnumeric config_string values must be enclosed in double quotes in the file if you are editing the SDE_dbtune table using Transact-SQL statements. If you are editing the table directly in the Enterprise Manager, double quotes are not used.

Understanding keywords

The SDE_dbtune table is grouped by keyword. At ArcSDE 9, the following keywords are included by default:

```
Select distinct(keyword) from sde.sde_dbtune

DATA_DICTIONARY
DEFAULTS
IMS_METADATA
IMS_METADATARELATIONSHIPS
IMS_METADATATAGS
IMS_METADATATHUMBNAILS
IMS_METADATAUSERS
IMS_METADATAVALUES
IMS_METADATAWORDINDEX
IMS_METADATAWORDS
LOGFILE_DEFAULTS
NETWORK_DEFAULTS
NETWORK_DEFAULTS::DESC
NETWORK_DEFAULTS::NETWORK
SURVEY_MULTI_BINARY
TOPOLOGY_DEFAULTS
TOPOLOGY_DEFAULTS::DIRTYAREAS
```

You reference a keyword when loading data with ArcSDE command tools or ArcCatalog or creating ArcIMS metadata server documents. When you build ArcGIS features, such as geometric networks, you can also reference a keyword.

DEFAULTS keyword

The DEFAULTS keyword has 78 parameter_names and config_string pairs. The DEFAULTS keyword is used when:

1. You create an object within an ArcSDE database and don't specify an SDE_dbtune keyword.
2. You customize or create new keywords but do not supply all parameter_names or config_strings. Missing parameter_names and config_strings will be read from the DEFAULTS keyword list.

You can modify config_strings within the DEFAULTS keyword to apply those settings to new objects (tables and indexes) created by the ArcSDE application.

SDE_dbtune sample showing some records of the DEFAULTS keyword:

```
keyword         parameter_name          config_string
DEFAULTS        A_CLUSTER_RASTER        0
DEFAULTS        A_CLUSTER_ROWID         0
```

Dbtune file sample showing same records:

```
##DEFAULTS
A_CLUSTER_ROWID     0
A_CLUSTER_RASTER  0
```

These parameter_name and config_string pairs apply to the DEFAULTS keyword. An explanation of each is included later in this chapter in the section entitled 'Understanding parameter name–config string pairs.'

```
##DEFAULTS
NUM_DEFAULT_CURSORS -1
CROSS_DB_QUERY_FILTER  0
UI_TEXT     ""
A_CLUSTER_ROWID     0
A_CLUSTER_SHAPE     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_CLUSTER_XML  0
A_CLUSTER_RASTER  0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_SHAPE       "WITH FILLFACTOR = 75"
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_INDEX_XML "WITH FILLFACTOR = 75"
A_INDEX_RASTER      "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER      0
B_CLUSTER_XML 0
B_CLUSTER_RASTER 0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_SHAPE       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
```

```
B_INDEX_XML "WITH FILLFACTOR = 75"
B_INDEX_RASTER      "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 75"
D_STORAGE   ""
F_CLUSTER_FID       1
F_INDEX_AREA        "WITH FILLFACTOR = 75"
F_INDEX_FID "WITH FILLFACTOR = 75"
F_INDEX_LEN "WITH FILLFACTOR = 75"
F_STORAGE   ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL       1
S_CLUSTER_SP_FID    0
S_INDEX_ALL "WITH FILLFACTOR = 75"
S_INDEX_SP_FID      "WITH FILLFACTOR = 75"
S_STORAGE   ""
RAS_STORAGE        ""
RAS_CLUSTER_ID   1
RAS_INDEX_ID    "WITH FILLFACTOR = 75"
BND_STORAGE        ""
BND_CLUSTER_ID   0
BND_INDEX_ID    "WITH FILLFACTOR = 75"
BND_CLUSTER_COMPOSITE   0
BND_INDEX_COMPOSITE    "WITH FILLFACTOR = 75"
AUX_STORAGE        ""
AUX_CLUSTER_COMPOSITE   1
AUX_INDEX_COMPOSITE    "WITH FILLFACTOR = 75"
BLK_STORAGE        ""
BLK_CLUSTER_COMPOSITE   1
BLK_INDEX_COMPOSITE    "WITH FILLFACTOR = 75"
END
```

DATA_DICTIONARY keyword

When the ArcSDE system tables are created during setup, the ArcSDE and geodatabase system tables and indexes are created with the storage parameters of the DATA_DICTIONARY keyword. To influence where and how these tables are created, you have two options:

1. Drop any existing ArcSDE system tables (be careful), edit the dbtune.sde file in %SDEHOME%etc, then run sdesetupmssql -o install.
2. Create a custom dbtune.sde file, run the ArcSDE post installation utility, and browse to it during the ArcSDE configuration files portion of the setup.

To alter placement of existing tables, use the Enterprise Manager or Transact-SQL to add files and file groups. See the SQL Server Books Online for more information.

If your site makes extensive use of a multiversioned database, consider separating the SDE_MVTABLES_MODIFIED, SDE_STATE_LINEAGES, and SDE_STATES tables and indexes into their own file groups to reduce I/O contention. 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.

These parameter names and config_strings apply to the DATA_DICTIONARY keyword:

```
##DATA_DICTIONARY
B_CLUSTER_ROWID     0
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
MVTABLES_MODIFIED_INDEX "WITH FILLFACTOR = 75"
MVTABLES_MODIFIED_TABLE ""
STATE_LINEAGES_INDEX    "WITH FILLFACTOR = 75"
STATE_LINEAGES_TABLE    ""
STATES_INDEX    "WITH FILLFACTOR = 75"
STATES_TABLE    ""
VERSIONS_INDEX  "WITH FILLFACTOR = 75"
VERSIONS_TABLE  ""
XML_INDEX_TAGS_INDEX        "WITH FILLFACTOR = 75"
XML_INDEX_TAGS_TABLE        ""
XML_TAGS_PK_INDEX   "WITH FILLFACTOR = 75"
XML_TAGS_TABLE      ""
XML_TAGS_UK_INDEX   "WITH FILLFACTOR = 75"
END
```

The XML_INDEX_TAGS_* parameters govern allocations on the sde_index_tags table. This table can grow and will be frequently accessed by applications, such as the ArcIMS Metadata Server, that use the XML document type. The XML_TAGS_* parameters control allocations of the SDE_XML_TAGS table. This table will also be actively used by the Metadata Server. The XML_TAGS_PK_INDEX is clustered by default and cannot be separated from its table.

LOGFILE keywords

ArcSDE maintains temporary and persistent sets of selected records through log file tables. There are two general types of log files, session based and standard.

Session log files employ a pool of log file tables while standard log files employ two log file tables per user, SDE_Logfiles and SDE_Logfile_data. Session log files are created during setup. Standard log files are created during the first connection a user makes or re-created if they have been deleted the next time a user connects.

Most installations of ArcSDE will function well using the LOGFILE_DEFAULTS parameters. However, for applications that make heavy use of log files, such as ArcInfo™, ArcEditor™, ArcIMS Metadata Server, and ArcView®, it may help scalability to spread the log files across separate file groups. ArcGIS clients write log files whenever a selection set exceeds 100 records. Following are log file parameters and config_strings from the dbtune.sde file:

```
##LOGFILE_DEFAULTS
LD_CLUSTER_ALL      0
```

```
LD_CLUSTER_ROWID    1
LD_INDEX_ALL        "WITH FILLFACTOR = 75"
LD_INDEX_ROWID      "WITH FILLFACTOR = 75"
LD_STORAGE  ""
LF_CLUSTER_ID       0
LF_CLUSTER_NAME     0
LF_INDEX_ID "WITH FILLFACTOR = 75"
LF_INDEX_NAME       "WITH FILLFACTOR = 75"
LF_STORAGE  ""
UI_TEXT     ""
END
```

Three extra log file parameters will appear in your SDE_dbtune table:

```
SESSION_INDEX               WITH FILLFACTOR=75
SESSION_STORAGE
SESSION_TEMP_TABLE          1
```

IMS_METADATA keywords

The IMS_METADATA keywords are a standard part of the SDE_dbtune table. If these keywords are not present in a dbtune file when it is imported into the SDE_dbtune table, ArcSDE will supply software defaults. The software defaults have the same parameters listed in the dbtune.sde file that is shipped with ArcSDE:

```
IMS_METADATA
IMS_METADATARELATIONSHIPS
IMS_METADATATAGS
IMS_METADATATHUMBNAILS
IMS_METADATAUSERS
IMS_METADATAVALUES
IMS_METADATAWORDINDEX
IMS_METADATAWORDS
```

These keywords are now obsolete. They will not be used by ArcSDE or ArcIMS Metadata Services. The gazetteer import scripts provided with ArcIMS 9.1 do not use the above keywords to load the gazetteer data into ArcSDE. For information on configuring XML storage, please read the *'XML type parameters'* topic on page 41 of this chapter and Appendix F 'Storing XML Data'.

NETWORK_DEFAULTS keyword

The NETWORK_DEFAULTS keyword contains the default parameters for the ArcGIS network class. If no keyword is supplied upon creation of a geometric network, parameters are read from this keyword. If a keyword containing incomplete parameter_names is used during network creation, missing parameter names will be supplied from the DEFAULTS keyword.

A geometric network will create several new tables within the database. The largest table will generally be the N_<id>_DESC table, so make sure its corresponding SDE_dbtune keyword, NETWORK_DEFAULTS::DESC, either explicitly lists or defaults to a file group that will accommodate large table and index growth.

```
##NETWORK_DEFAULTS
```

```
UI_NETWORK_TEXT     "The network default configuration"
COMMENT     "The base system initialization parameters for
NETWORK_DEFAULTS"
A_CLUSTER_ROWID     0
A_CLUSTER_SHAPE     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_SHAPE       "WITH FILLFACTOR = 75"
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_SHAPE       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 75"
D_STORAGE   ""
F_CLUSTER_FID       1
F_INDEX_AREA        "WITH FILLFACTOR = 75"
F_INDEX_FID "WITH FILLFACTOR = 75"
F_INDEX_LEN "WITH FILLFACTOR = 75"
F_STORAGE   ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL       1
S_CLUSTER_SP_FID    0
S_INDEX_ALL "WITH FILLFACTOR = 75"
S_INDEX_SP_FID      "WITH FILLFACTOR = 75"
S_STORAGE   ""
END
```

Network Composite 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 with 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.

Network composite keywords listed from the dbtune.sde file:

```
##NETWORK_DEFAULTS::DESC
A_CLUSTER_ROWID     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     1
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_CLUSTER_STATE_ROWID       0
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 75"
D_INDEX_STATE_ROWID "WITH FILLFACTOR = 75"
D_STORAGE   ""
END

##NETWORK_DEFAULTS::NETWORK
A_CLUSTER_ROWID     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     1
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 75"
D_STORAGE   ""
END
```

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 compounded. You may specify the DIRTYAREAS suffix to list the configuration string to be used to create the topology tables.

Topology keywords and parameter name–config string pairs from dbtune.sde:

```
##TOPOLOGY_DEFAULTS
UI_TOPOLOGY_TEXT    "The topology default configuration"
A_CLUSTER_ROWID     0
A_CLUSTER_SHAPE     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_SHAPE       "WITH FILLFACTOR = 75"
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_SHAPE       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 75"
D_STORAGE   ""
F_CLUSTER_FID       1
F_INDEX_AREA        "WITH FILLFACTOR = 75"
F_INDEX_FID "WITH FILLFACTOR = 75"
F_INDEX_LEN "WITH FILLFACTOR = 75"
F_STORAGE   ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL       1
S_CLUSTER_SP_FID    0
S_INDEX_ALL "WITH FILLFACTOR = 75"
S_INDEX_SP_FID      "WITH FILLFACTOR = 75"
S_STORAGE   ""
END

##TOPOLOGY_DEFAULTS::DIRTYAREAS
A_CLUSTER_ROWID     0
A_CLUSTER_SHAPE     1
A_CLUSTER_STATEID   0
A_CLUSTER_USER      0
A_INDEX_ROWID       "WITH FILLFACTOR = 75"
A_INDEX_SHAPE       "WITH FILLFACTOR = 75"
```

```
A_INDEX_STATEID     "WITH FILLFACTOR = 75"
A_INDEX_USER        "WITH FILLFACTOR = 75"
A_TEXT_IN_ROW   256
A_STORAGE   ""
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER      0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_SHAPE       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
D_CLUSTER_ALL       0
D_CLUSTER_DELETED_AT        1
D_INDEX_ALL "WITH FILLFACTOR = 75"
D_INDEX_DELETED_AT  "WITH FILLFACTOR = 90"
D_STORAGE   ""
F_CLUSTER_FID       1
F_INDEX_AREA        "WITH FILLFACTOR = 75"
F_INDEX_FID "WITH FILLFACTOR = 75"
F_INDEX_LEN "WITH FILLFACTOR = 75"
F_STORAGE   ""
F_TEXT_IN_ROW   256
S_CLUSTER_ALL       1
S_CLUSTER_SP_FID    0
S_INDEX_ALL "WITH FILLFACTOR = 75"
S_INDEX_SP_FID      "WITH FILLFACTOR = 75"
S_STORAGE   ""
END
```

Understanding parameter name—config string pairs

Parameter name—config strings pairs are used by ArcSDE to identify:

1. Where a table or index is created (file group)
2. Whether or not to cluster an index
3. How much to fill each index page (FILLFACTOR)
4. How much binary data should be stored inline to a data page (TEXT_IN_ROW)

There are a few additional parameter_names that are keyword specific. Parameter_names identify the object to be configured, while config_strings identify how that object will be configured. Parameter_names are fixed; you cannot invent new parameter_names. Likewise, config_strings accept only certain numeric values or SQL strings. These strings are appended to Transact-SQL CREATE TABLE and CREATE INDEX statements.

The SDE_dbtune table is ordered by keyword. Keyword/Parameter_name combinations are unique, but most parameter_names are not and are reused many times throughout the table.

Adds Table (Multi-Version Table) Parameters

An Adds table is a multiversioned table that stores edits made against a feature class in a multi-versioned database. It is almost identical in structure to the business table but has additional columns to track state ids. Nonspatial tables have no shape column, so you should cluster another index. Adds table parameters begin with "A".

A_CLUSTER_ROWID 0
Index type for rowid column on Adds table; 0 = non-clustered index; 1 = clustered index.

A_CLUSTER_SHAPE 1
Index type for Add table shape column; 0 = non-clustered index; 1 = clustered index.

A_CLUSTER_STATEID 0
Index type for Adds table stateid column; 0 = non-clustered index; 1 = clustered index.

A_CLUSTER_USER 0
Index type for any user defined indexes on Adds table; 0 = non-clustered; 1 = clustered.

A_CLUSTER_XML 0
Index type for xml doc type column of Adds table; 0 = non-clustered; 1 = clustered.

A_CLUSTER_RASTER 0
Index type for raster column in Adds table; 0 = non-clustered; 1 = clustered.

A_INDEX_ROWID "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for ROWID column index of Adds Table. To specify a file group, use the SQL 'ON' statement. Eg:
A_index_rowid "with fillfactor=99 on IDXfg"

A_INDEX_SHAPE "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for shape column index of Adds Table. To specify a file group, use the SQL 'ON' statement. Eg:
A_index_shape "with fillfactor=99 on SHAPEfg"

A_INDEX_STATEID "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for STATEID column index of Adds Table. To specify a file group, use the SQL 'ON' statement. Eg:
A_index_stateid "with fillfactor=99 on STATEIDXfg"

A_INDEX_USER "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for any user defined indexes on Adds Table. To specify a file group, use the SQL 'ON' statement. Eg:
A_index_USER "with fillfactor=99 on IDXfg"

A_INDEX_XML "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for XML index on Adds Table. To specify a file group, use the SQL 'ON' statement. Eg:
A_index_XML "with fillfactor=99 on XMLfg"

A_INDEX_RASTER "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for raster index on Adds Table. To specify a file group, use the SQL 'ON' statement. Eg: A_index_RASTER "with fillfactor=99 on RASfg"

A_TEXT_IN_ROW 256
Amount, in bytes, of image (SE_blob) data type (vector or raster) to store directly in the data page of the table.

```
A_STORAGE    ""
```
File group location for Adds table. Use the on keyword to control location. Eg: A_STORAGE "ON ADDS_FG"

NOTE: You cannot separate a clustered index from its table.

BUSINESS table parameters

The BUSINESS table is the ATTRIBUTE table of a FEATURE class. BUSINESS tables can also be nonspatial. If you have a nonspatial BUSINESS table, do one of the following:

1. Change the B_CLUSTER_ROWID parameter's config_string to 1 and the B_CLUSTER_SHAPE parameter's config_string to 0. This will create a clustered index on the ObjectID field. Any subsequent user-defined indexes you create will be nonclustered.
2. Change the B_CLUSTER_USER parameter's config_string to 1. The first user-defined index created by ArcSDE will be clustered. Change B_CLUSTER_SHAPE to 0.
3. Create the data and change whatever index (or composite indexes) you would like to be clustered.

BUSINESS table parameters begin with B.

```
B_CLUSTER_ROWID     0
B_CLUSTER_SHAPE     1
B_CLUSTER_USER      0
B_CLUSTER_XML 0
B_CLUSTER_RASTER 0
B_INDEX_ROWID       "WITH FILLFACTOR = 75"
B_INDEX_SHAPE       "WITH FILLFACTOR = 75"
B_INDEX_USER        "WITH FILLFACTOR = 75"
B_INDEX_XML "WITH FILLFACTOR = 75"
B_INDEX_RASTER      "WITH FILLFACTOR = 75"
B_STORAGE   ""
B_TEXT_IN_ROW   256
```

Business table parameters work in the same fashion as ADDS table parameters. To change them, see the section on ADDS tables.

DELETES table parameters

The DELETES table is used to track updates and deletes against multiversioned tables. All DELETES table parameters begin with “D”.

```
D_CLUSTER_ALL        0
```
Index type for SDE_STATES_ID, SDE_DELETES_ROW_ID, and DELETED_AT column. 0 = non-clustered; 1 = clustered.

```
D_CLUSTER_DELETED_AT          1
```
Index type for DELETED_AT column; 0 = non-clustered; 1 = clustered.

D_INDEX_ALL "WITH FILLFACTOR = 75"
FILLFACTOR and location (file group) for composite index on SDE_STATES_ID, SDE_DELETES_ROW_ID, and DELETED_AT columns. Eg:
D_INDEX_ALL "with fillfactor=99 on Deletes_fg"

D_INDEX_DELETED_AT "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for index on DELETED_AT column. Eg:
D_INDEX_DELETED_AT "with fillfactor=80 on Deletes_fg"

D_STORAGE ""
File group location for deletes table. Use the on keyword to control location. Eg: D_STORAGE "ON Deletes_fg"

NOTE: you cannot separate a table from its clustered index.

DELETES table parameters work in the same fashion as ADDS table parameters. To customize Deletes table config_strings, see the above section on the ADDS tables.

FEATURE table parameters

The Feature table stores each shape's extent and geometry. It will also contain records from multiversioned inserts and updates. All Feature table parameters begin with "F".

F_CLUSTER_FID 1
Index type for FID column. 0 = non-clustered; 1 = clustered.

F_INDEX_AREA "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for area column index. Eg:
F_INDEX_AREA "WITH FILLFACTOR = 90 ON F_IDX"

F_INDEX_FID "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for fid column index. Eg:
F_INDEX_FID "WITH FILLFACTOR = 90 ON F_IDX"

F_INDEX_LEN "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for len column index. Eg:
F_INDEX_LEN "With FILLFACTOR = 90 on F_IDX"

F_STORAGE ""
File group location for f table. Use the on keyword to control location. Eg:
F_STORAGE "WITH FILLFACTOR=90 on F_IDX"
NOTE: You cannot separate a table from its clustered index.

F_TEXT_IN_ROW 256
Amount, in bytes, of image (SE_blob) data type (vector or raster) to store directly in the data page of the table.

SPATIAL INDEX table parameters

The Spatial Index is a grid that overlays features and is used to identify features to fetch. The bounding box of spatial query is overlaid against the spatial index table to select candidate shapes satisfying the query. Spatial index table parameters begin with "S."

S_CLUSTER_ALL 1
Index type for primary key (all columns of table). 1 = clustered; 0 = non-clustered.

```
S_CLUSTER_SP_FID     0
Index type for sp_fid column. 1 = clustered; 0 = non-clustered.

S_INDEX_ALL "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for primary key. Eg:
      S_INDEX_ALL        "With FILLFACTOR = 90 on S_IDX"

S_INDEX_SP_FID       "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for sp_fid column index. Eg:
      S_INDEX_SP_FID  "WITH FILLFACTOR = 85 on S_IDX"

S_STORAGE    ""
File group location for S table. Use the on keyword to control location. Eg:
S_STORAGE  "WITH FILLFACTOR=95 on S_IDX"
NOTE: You cannot separate a table from its clustered index.
```

RASTER table parameters

Rasters in ArcSDE are stored as 5 separate tables: a band table (SDE_bnd_#), a block table (SDE_blk_#), a raster table (SDE_ras_#), an auxiliary table (SDE_aux_#), and a business table. Of these tables, only the block table will get large. Make certain that the BND_CLUSTER_COMPOSITE config_string is set to 1 to ensure that a clustered index is generated for this table. Raster table parameters begin with AUX, BLK, BND, and RAS. Rasters can be stored as embedded catalogs, or columns to sde feature classes or can be stand-alone layers. Business table parameters are read from the B_ settings.

```
RAS_STORAGE          ""
File group location for RAS table. Use the ON keyword to control location. Eg:
      RAS_STORAGE     " ON RAS_FG"

RAS_CLUSTER_ID    1
Index type for primary key of RAS table. 1 = clustered; 0 = nonclustered.

RAS_INDEX_ID     "WITH FILLFACTOR = 75"
Fillfactor and location (file group) for primary key. Eg:
      RAS_INDEX_ID     "WITH FILLFACTOR = 85 ON RAS_FG"

BND_STORAGE          ""
File group location for BND table. Use the on keyword to control location. Eg:
      BND_STORAGE " ON BND_FG"

BND_CLUSTER_ID    0
Index type for RASTER_ID, SEQUENCE_NBR columns. 1 = clustered; 0 = nonclustered.

BND_INDEX_ID     "WITH FILLFACTOR = 75"
Fillfactor and file group location for RASTER_ID, SEQUENCE_NBR column index. Eg:
      BND_INDEX_ID     "WITH FILLFACTOR = 90 ON BND_FG"

BND_CLUSTER_COMPOSITE    0
Index type for primary key; 1 = clustered; 0 = nonclustered.

BND_INDEX_COMPOSITE     "WITH FILLFACTOR = 75"
Fillfactor and file group location for primary key. Eg:
      BND_INDEX_COMPOSITE    "WITH FILLFACTOR = 90 ON BND_FG"
```

```
AUX_STORAGE          ""
File group location for Auxiliary table. Use ON keyword to specify location: Eg:
AUX_STORAGE          "ON AUX_FG"

AUX_CLUSTER_COMPOSITE    1
Index type for primary key. 1 = clustered; 0 = non-clustered.

AUX_INDEX_COMPOSITE     "WITH FILLFACTOR = 75"
Fillfactor and file group location for primary key. Eg:
     AUX_INDEX_COMPOSITE    "WITH FILLFACTOR = 90 ON AUX_FG"

BLK_STORAGE          ""
File group location for block table. Use ON keyword to specify location. Eg:
     BLK_STORAGE     "ON BLK_FG"

BLK_CLUSTER_COMPOSITE    1
Index type for primary key. 1 = clustered; 0 = non-clustered.

BLK_INDEX_COMPOSITE     "WITH FILLFACTOR = 75"
File group location for block table. Use ON keyword to specify location. Eg:
BLK_INDEX_COMPOSITE          "WITH FILLFACTOR = 95 ON BLK_FG"
```

XML type parameters

ArcSDE business tables that contain an SE_XML_TYPE column will employ two side tables to store the XML document and the XML document's index. XML document tables will have three parameters:

```
XML_DOC_INDEX       "WITH FILLFACTOR = 75"
XML_DOC_STORAGE     ""
XML_DOC_TEXT_IN_ROW 1024
```

XML_DOC_STORAGE provides the storage string for the table's creation statement. XML_DOC_INDEX has the index fillfactor and storage parameters, while the text_in_row pertains to in lining blob data. See the following section ‘Text in row parameters’ for more information.

XML document index tables are the most heavily accessed of the XML tables and have more dbtune parameters.

```
XML_IDX_CLUSTER_DOUBLE      0
XML_IDX_CLUSTER_ID  0
XML_IDX_CLUSTER_PK  1
XML_IDX_CLUSTER_TAG 0
XML_IDX_FULLTEXT_CAT        "SDE_DEFAULT_CAT"
XML_IDX_FULLTEXT_LANGUAGE   ""
XML_IDX_FULLTEXT_TIMESTAMP  1
XML_IDX_FULLTEXT_UPDATE_METHOD      "CHANGE_TRACKING BACKGROUND"
XML_IDX_INDEX_DOUBLE        "WITH FILLFACTOR = 75"
XML_IDX_INDEX_ID    "WITH FILLFACTOR = 75"
XML_IDX_INDEX_PK    "WITH FILLFACTOR = 75"
XML_IDX_INDEX_TAG   "WITH FILLFACTOR = 75"
XML_IDX_STORAGE     ""
XML_IDX_TEXT_IN_ROW 128
```

The first four parameters, XML_IDX_CLUSTER_DOUBLE, XML_IDX_CLUSTER_ID, XML_IDX_CLUSTER_PK, and XML_IDX_CLUSTER_TAG, dictate which index of the XML

document index table should be clustered. By default, the primary key's index (on the xml_key_column) is clustered. XML_IDX_FULLTEXT_CAT contains the name of the full text catalog you created with sp_fulltext_catalog. XML_IDX_FULLTEXT_LANGUAGE represents the language to be used in the full text catalog. The default is Neutral (“”), which can store multiple languages.

XML_IDX_FULLTEXT_TIMESTAMP and XML_IDX_FULLTEXT_UPDATE_METHOD control full text index maintenance. The update_method parameter dictates how changes made to the document table are propagated to the full text index. The time stamp parameter, by default (1), will add a time stamp column to the sde_xml_idx<xml_column_id> table. If set to 0, no such column is added.

If update_method is set to 0 and time stamp is set to 0, then no index maintenance is performed, and whenever ArcSDE is instructed to update the full text index (through SE_xmlindex_update_text_index), the index will be fully populated.

If update_method is set to 0 and time stamp is set to 1, no index maintenance is performed, and ArcSDE will perform an incremental index population—whatever has changed since the last incremental update.

If update_method is set to CHANGE_TRACKING MANUAL, the database maintains a list of changed rows but does not update the index.

If update_method is set to CHANGE_TRACKING BACKGROUND, the database tracks changes and automatically updates the index.

It is recommended that you use the default settings provided in the dbtune table. If your server is unable to service its workload, and your only recourse is to change indexing behavior, set change tracking to manual (CHANGE_TRACKING MANUAL).

The next parameters, XML_IDX_INDEX_DOUBLE, XML_IDX_INDEX_ID, XML_IDX_INDEX_PK, XML_IDX_INDEX_TAG, and XML_IDX_STORAGE, control index fill factor and storage on the sde_xml_idx<xml_column_id> table. The final parameter, xml_idx_text_in_row, controls how much of the XML document blob can be in lined. As with most text in row settings, it is recommended that you do not change the defaults. The next section discusses text in row.

Text in row parameters

SQL Server 2000 supports the ability to store image, text, and *n* text values directly in a data row. Previous versions of SQL Server would store the images types in image data pages. The advantage of “in lining”, or storing some image types in a data row is that you may speed fetching of your data since queries will traverse fewer pages to find your data. ArcSDE uses an image type column to store its geometry.

The SQL Server and ArcSDE default for text in a row is 256 bytes but can span from 24 to 7000 bytes. Be forewarned that the more image data you store in a row, the larger your table will become, eventually slowing your queries as more pages must be traversed. Raster data should not be stored in a row unless you can shrink the raster tile size to 83 X 83 to fit an image within the 7k page limit. This is only realistic if you store small

1 bit images or 16 X 16 byte icons. Points do not need to be stored in a row, as spatial queries against them reconstruct the features from their envelopes. By default, all ArcSDE objects are enabled for text in a row, up to 256 bytes.

If you want to change this default setting, try to inline up to 80 percent of your vector data up to 1k. You can ascertain this size by running the following query:

```
select top 20 percent(datalength(points)) from <feature table> order by
datalength(points) desc
```

Pick the smallest value from this result set. Raster data is usually much larger. If you attempt to inline too much raster or vector data, you'll end up creating too many extents and data pages.

Fill factor parameters

The *_INDEX_* parameter allows you to specify the FILLFACTOR argument for that index. The fillfactor argument specifies how full each page in the leaf level of an index should be. SQL Server uses a default value of 0, which means that the leaf pages of an index are almost full, but the nonleaf pages have room for at least two more rows. User-defined fillfactors can be between 1 and 100. If the fillfactor is 100, all pages are completely full. With a fill factor of 75, each clustered index page starts 75 percent full. Subsequent inserts and updates to that data add to the index page. When the page hits 100 percent capacity, it is full. Any subsequent insert or update to data in that page will split the page.

Use FILLFACTOR to balance full index pages and page splits. When a page is split, SQL Server moves approximately 50 percent of the data in the split page to a new page, most likely allocated from a different extent. Page splits will fragment your tables and compromise performance. Setting Fillfactor too low creates too many data pages and extents to traverse in a query, thus negatively impacting performance.

Decision criteria for choosing a fillfactor:

1. Is your data read-only? Will it never be edited? If yes, set all fillfactors on your data to 100.
2. Will your data be updated frequently? Use the defaults.
3. Will your data incur sporadic updates? Pick a range between 75–95 percent based on how often you want to defragment your tables.

Monitor fragmented tables and page splits with DBCC SHOWCONTIG.

Clustered index parameters

All *_CLUSTER_* parameters indicate whether or not a particular index should be clustered (1 = cluster; 0 = nonclustered). Clustered indexes store tabular data at their leaf nodes. The data pages at the clustered index leaf level derive their order from the clustered index key value. This has one important consequence with regard to the SDE_dbtune table: You cannot separate a table from its clustered index.

For example, you specify that a feature table's FID index be created on the FeatIdx file group while the feature table should be stored on the Feat file group. The FID index is created as clustered. SDE_dbtune might look like this:

Keyword	Parameter_name	Config_string
DEFAULTS	F_INDEX_FID	WITH FILLFACTOR=90 ON FEATIDX
DEFAULTS	F_STORAGE	ON FEAT

In the preceding example, both the feature table and feature table's index will reside on the FeatIdx file group. The feature table is created first, then a primary key constraint is applied to the FID column. The constraint creates a clustered index on the fid column and references the FEATIDX file group in this statement:

```
Alter Table features.dbo.f4 add constraint f4_pk primary key clustered
(fid) with fillfactor=75 on FEATIDX
```

Therefore, the ON FEAT configuration string is redundant, as the index is created after the table, and the F_INDEX_FID configuration string will overwrite that of F_STORAGE.

The next example specifies a file group for the feature table but not for the feature table's index.

Keyword	Parameter_name	Config_string
DEFAULTS	F_INDEX_FID	WITH FILLFACTOR=90
DEFAULTS	F_STORAGE	ON FEAT

In this case, both the feature table and clustered index on the FID column will reside on the Feat file group. This occurs because the table is created first, and when the alter table statement is applied no ON statement is appended because no such string is listed in the config_string column above.

```
Alter table features.dbo.f5 add constraint f5_pk primary key clustered
(fid) with FILLFACTOR=75
```

CROSS_DB_QUERY_FILTER parameter

The CROSS_DB_QUERY_FILTER parameter has two possible settings, 0 and 1. It only applies to multidatabase models (where an sde database holds the sde and geodatabase metadata). By default CROSS_DB_QUERY_FILTER is set to 0. CROSS_DB_QUERY_FILTER controls whether or not a connecting user can view rasters or feature classes across database boundaries. In a multidatabase ArcSDE service, you can access rasters and feature classes in any database that participates in that service, regardless of the database they connect to. By setting CROSS_DB_QUERY_FILTER to 1, you can only view and access rasters and feature classes in the database they've explicitly connected to. For example, given a multidatabase ArcSDE service comprised of a SDE, fisheries, watershed, and coasts databases, if CROSS_DB_QUERY_FILTER is set to 1, a user that connects to the fisheries database cannot view rasters or feature classes in the watershed database. It is recommended that you migrate their data from multidatabase services to single database services. In a single database service, CROSS_DB_QUERY_FILTER is not used.

GEOMETRY_STORAGE parameter

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

- ArcSDE compressed binary format. This is the default spatial storage method of ArcSDE for SQL Server. Set the GEOMETRY_STORAGE parameter to SDEBINARY if you wish to store your spatial data in this format. If the GEOMETRY_STORAGE parameter is not set, the SDEBINARY format is assumed.
- OGC Well-known binary geometry type. This type provides a portable representation of geometry as a contiguous stream of bytes. Set the GEOMETRY_STORAGE parameter to OGCWKB if you wish to store your spatial data in this format. If you wish to make this format the default, set the GEOMETRY_STORAGE parameter to OCGWKB in the DEFAULTS configuration keyword.

The OGC well-known binary representation supports only simple 2D geometries. Please see Appendix C for a description of the OGC well-known binary representation.

The default value for GEOMETRY_STORAGE is SDEBINARY.

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

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

CHAPTER 4

Managing tables, feature classes, raster columns, and views

ArcSDE spatial databases contain feature classes, nonspatial tables, tables with raster columns, and spatial views. This chapter describes the use of ArcSDE command tools to create, drop, and manage these entity types.

Setting up the dbtune table

ArcSDE controls data loading, including table and index file group placement, index type (clustered or nonclustered), and index fillfactor through the dbtune table. Dbtune tables contain keywords, parameters, and configuration parameters columns. Keywords identify specific loading parameters. Parameters represent ArcSDE objects such as tables and indexes. Configuration parameters specify how something should be loaded.

For further information on the dbtune table, see Chapter 3, ‘Configuring dbtune storage parameters’.

ArcSDE to Microsoft SQL Server data type mapping

ArcSDE uses 12 general data types. These types are mapped to Microsoft SQL Server 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	Microsoft SQL Server Data Type
SE_STRING_TYPE	Char, varchar

ArcSDE Data Type	Microsoft SQL Server Data Type
SE_NSTRING_TYPE	Nchar, nvarchar
SE_NCLOB_TYPE	Text
SE_INT16_TYPE (SE_SMALLINT_TYPE)	Smallint, tinyint, numeric (precision < = 4, scale = 0)
SE_INT32_TYPE (SE_INTEGER_TYPE)	Int, numeric (precision < = 9, scale = 0)
SE_INT64_TYPE	Bigint, numeric (precision < 19, scale = 0) Disabled by default; not supported by ArcGIS; enable with sdeconfig. Unsupported with SQL Server 7.0
SE_FLOAT32_TYPE (SE_FLOAT_TYPE)	Float, numeric (precision < 7, scale > 0)
SE_FLOAT64_TYPE (SE_DOUBLE_TYPE)	Real, money, smallmoney, numeric (precision > = 7, scale > 0)
SE_DATE_TYPE	Datetime, smalldatetime
SE_UUID_TYPE	Uniqueidentifier
SE_BLOB_TYPE	Image, binary, varbinary

Importing data

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

1. ArcGIS Desktop—Use ArcCatalog to manage and populate your database. If you use ArcGIS to access your data, you must use ArcCatalog or ArcObjects™ to manage your spatial data.
2. ArcSDE administration commands located in the bin directory of SDEHOME.

   NOTE: None of the ArcSDE admin commands are "geodatabase" enabled, meaning they do not register the data they create, drop, or alter with the geodatabase.
3. ArcInfo Workstation—Use the Defined Layer interface to create and populate the database.
4. ArcView GIS 3.2—Use the Database Access extension.
5. MapObjects®—Custom Common 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 C or Java APIs.

This document focuses on the ArcSDE administration tools. Generally, using ArcGIS to build your data model is preferable to using the command line tools. The command line tools are best for batch processing or for moving large amounts of data quickly.

For information on using ArcGIS to build your data, consult the following documentation:

1. In the documentation folder of your ArcGIS installation
   a. Geoprocessing_Quick_Guide.pdf
   b. Geoprocessor.pdf
   c. Topology_rules_poster.pdf
2. Printed manuals
   a. *Building a Geodatabase*
   b. *Using ArcCatalog*

Before you begin—know your data

Before you begin loading your data, get to know it. SQL Server 2000 has approximately 174 reserved words and 135 future keywords. The use of reserved words for table and column names is not supported. See the SQL Server Books Online for more information.

The use of nonstandard identifiers is also unsupported. Such an identifier could be naming a table "3d". You cannot begin an ArcSDE table name with a number, pound sign (#), or symbol.

Watch the length and naming of your columns as well. ArcSDE enforces a 32-character limit for a column name. Column names with embedded spaces are not supported. Finally, be aware of your data ranges, especially with date data types. SQL Server cannot accept datetime values that precede January 1, 1753. If you attempt to load data into SQL Server through ArcSDE with values preceding this range, the load will fail.

Creating and populating feature classes

A feature class, as defined by the ArcGIS geodatabase, is a collection of features with the same type of geometry and attributes. ArcSDE command line tools that create feature classes include:

- shp2sde: Load a shapefile into an ArcSDE feature class.
- sdeimport: Load an ArcSDE export file into an ArcSDE feature class.
- cov2sde: Load an ArcInfo coverage, Map LIBRARIAN, or ArcStorm™ layer into an ArcSDE feature class.

- sdelayer: When used with the -o create option, create an ArcSDE feature class from an existing business table.
- sdegroup: Group or tile features from an existing feature class into another feature class. Use for rapid display of large amounts of data, keeping in mind that original attribute information is not retained.

Importing vector data from existing file-based data sources

Shp2sde, cov2sde, and sdeimport include a "-o create" operation, which allows you to import an existing shapefile, coverage, or sde export file into an 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.

  NOTE: The command line tools do not register the feature class with the geodatabase. A feature class that has not been registered with the geodatabase cannot participate in ArcGIS functionality, such as topologies, geometric networks, and feature-linked annotation.
- Puts the feature class into load_only_io mode. Load-only mode means that the feature class has no spatial index.
- Inserts data into the feature class.
- When all the records are inserted, puts the feature class into normal_io mode. Normal_io mode signifies that a spatial index has been built.

`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.

```
shp2sde -o create -f rdshp -l roads,shape -u eric -p sea.floor -i 5151 -s
GISData
```

In this example, the -k roadsys switch identifies a dbtune parameter.

```
shp2sde -o create -f rdshp -l roads,shape -u eric -p sea.floor -i 5151 -s
GISData -k roadsys
```

To load data using a particular projection ID, specify the -G option. A full listing of projection ID is available in the ArcSDE developer help. In this example, -G 26746 corresponds to NAD 1927 SPCS Zone California VI. The connection is made using Windows authentication (no -u -p arguments).

```
shp2sde -o create -l parcel,shape -i 9000 -f pars -G 26746 -D SanDiego
```

```
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 l+ -u eric -p sea.floor -i 5151 -s GISData
```

```
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". ArcSDE export files are created with the sdeexport command. This command does not export special ArcGIS types, such as feature-linked annotation, networks, and topologies.

```
sdeimport -o create -l roads,shape -f roadexp -u eric -p sea.floor -i 5151
-s GISData
```

```
sdegroup
```

If you have large datasets, you'll want to discourage your users from drawing these at full extent. There is no use in drawing a million polygons at full extent, as you can see no detail. However, your users may want to visualize what the overall shape or extent of their data looks like. The command sdegroup takes the existing shapes from a feature class and groups them into multipart shapes. While this may not be useful for analysis, it does allow rapid drawing of large amounts of data because it reduces the number of records. Polygons are generalized into polylines so as not to destroy topological relationships. In this example, features from the parcels feature class are generalized into tiles of 250 square meters.

```
sdegroup -o create -S parcel_p,shape -T parcel_group,shape -e a -a all -D
idb -i 9000 -t 250 -u dataowner -p tax.man
```

For a more complete discussion of sdegroup, see the ArcSDE administrator command references.

Appending data to an existing feature class

To append data from a shapefile, coverage, or sde export file to an existing feature class, use the -o append option of the applicable command tool. The append option will add data to the existing data in the feature class.

```
shp2sde -o append -f updated_rdshp2 -l roads,shape -u eric -p sea.floor -i
5151 -s GISData
```

```
sdeimport -o append -l roads,shape -f updated_roads_exp -u eric -p
sea.floor -i 5151 -s GISData
```

If the feature class is multiversioned, you can append features under a specified version. In this example, features from an export file, new_structures, are appended to an existing layer, buildings, but under "AssessorVersion", connecting with Windows authentication.

```
sdeimport -o append  -l buildings,shape -V AssessorVersion -f
new_structures -i 9000
```

Shp2sde, sdeimport, and cov2sde also allow you to truncate existing data from a feature class before appending data to it. This is done using the -o init option.

```
cov2sde -o init -l roads,shape -f roadlib,arcstorm -u eric -p sea.floor -i
5151 -s GISData
```

```
shp2sde -o init -f replacement_rdshp2 -l roads,shape -u eric -p sea.floor
-i 5151 -s GISData
```

Creating and populating object classes (DBMS tables)

The geodatabase defines an object class as a table that stores nonspatial data. These are also referred to as business tables. The sdetable command has options to create new ArcSDE business tables or register existing database management system (DBMS) tables. Registering a DBMS table allows ArcSDE to manage certain operations against that table.

Importing tabular data from existing file-based data sources

ArcSDE provides a command line tool, tbl2sde, to import existing dBASE, INFO, or SDE tables into an ArcSDE database. In this example, a .dbf file is imported into an ArcSDE database using a Windows authenticated connection:

```
tbl2sde -o create -t elevationpts -T dbase -f elevpnt.dbf -a all
-i 9000
```

Tbl2sde has an sde option that allows you to create a new sde business table from an existing sde table. In this example, table elevationpts is re-created in the database as elevback:

```
tbl2sde -o create -t evelback -f elevationpts -T SDE -a all -i 9000
```

Tbls2sde also supports the init and append operation on that shp2sde, cov2sde, and sdeimport support.

Registering existing business tables with ArcSDE

Existing DBMS tables can be registered with ArcSDE either through the sdetable -o register command or with ArcCatalog. Registering a table with ArcSDE is not the same as registering with the geodatabase. Registering an existing table with ArcSDE enables that table to be used with ArcView 3.x, MapObjects, and ArcIMS (through ArcSDE). It also enables that table to be used with some ArcSDE functionality, such as versioning.

In this example, an existing table, tax_pars, is registered with ArcSDE to be multiversioned and have a new SDE managed row_id column, OBJECTID, added to it. The connection is made via Windows authentication.

```
sdetable -o register -t tax_pars -c OBJECTID -C SDE -V MULTI -i 9000 -D
SanDiego
```

Creating and populating raster datasets

Imagery can be imported into ArcSDE from either ArcGIS or by using the sderaster command tool. For information on importing raster data into ArcSDE with ArcGIS, consult the ArcGIS Desktop Help topic "rasters".

The sderaster command line tool can be used to import .tif and .bsq file-based rasters. To import a file-based raster into ArcSDE, use the -o import option:

```
sderaster -o import -l seatemps1Jan03,image -f 010103.tif -c lz77 -G
file=projection.prj -L -1 -u eric -p sea.floor -i 9000 -D idb
```

In this example, a .tif file, 001.tif, is imported into an ArcSDE database, using lz77 compression. The -G option instructs ArcSDE to read the supplied projection file and apply its parameters to the imported raster. The -L -1 option instructs ArcSDE to build an optimal number of pyramids on the imported raster.

Rasters can be mosaicked or grouped into catalogs using the sderaster command. If you will load a lot of imagery into your SQL Server ArcSDE database, you must either mosaic your rasters or load them into raster catalogs to reduce the total number of tables you have in your spatial database. Both mosaicked rasters and embedded raster catalogs store all raster data in a single raster dataset. Rasters mosaicked with sderaster -o mosaic must share the same cell registration size and projection ID, while rasters loaded into embedded catalogs do not. For a more complete discussion of mosaics and embedded raster catalogs, see *http://support.esri.com*.

Raster datasets can also be loaded from sde export files as long as the export file contains a raster.

Understanding load_only and normal_io modes

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 elimination 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 eric -p sea.floor -i 5151 -s
GISData -D fisheries
```

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 sdelayer -o alter operation. The alter operation will apply an exclusive lock on the feature class, preventing all modifications by ArcGIS until the operation is complete.

Once the feature class is in load_only mode, you can insert features in bulk into it using either the -o init or -o append options of shp2sde, cov2sde, or sdeimport.

Switching back to normal_io mode

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

```
sdelayer -o normal_io -l roads,shape -u eric -p sea.floor –i 5151 –D
fisheries –s GISData
```

Managing data

Multiversioning your data

Feature classes and business tables (object classes) can be multiversioned. Versioning is a process that allows for multiple representations of your data without requiring duplication or copies of the data. Versioned data is used for modeling, controlling work flow of edits, and enabling long transactions. Versioning eliminates three fundamental problems with relational databases: deadlocking, no support for long transactions, and restrictive isolation levels that negatively impact scalability. Finally, ArcGIS requires data to be multiversioned for editing. For further information on working with versioned data in ArcGIS, refer to the *Building a Geodatabase* book.

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

```
sdetable -o alter_reg -t states -c objectid -C SDE -V multi –i 5151 –s
GISData –u eric –p sea.floor
```

The -C option instructs ArcSDE to manage the data's row_id column. This column is used by the application to allocate unique IDs. When registering a class as multiversioned, it is recommended you always allow ArcSDE to manage your row_id column.

Tables can also be registered as multiversioned:

```
sdetable –o alter_reg –t population –c objected –c SDE –V multi –i 5151 –s
GISData –u eric –p sea.floor
```

Modifying your schema

There will be occasions when it is necessary to modify the schema of some tables. Use ArcCatalog to add or remove columns, although this is not possible if the class is multiversioned. Both ArcCatalog and sdelayer allow you to modify the spatial index (grids) and add and drop column indexes.

To change the size of an ArcSDE spatial index, use the sdelayer -o alter command. In this example, a spatial index's size is changed to a cell size of 1,000. The connection made uses Windows authentication.

```
sdelayer -o alter  -l parcel_p,shape -g 1000 -i 9000
```

The sdetable command can be used to add an index. Control index creation parameters using dbtune keywords.

```
sdetable -o create_index -t parcels -n idxOwner -c ownerfield -i 9000 -u
dataowner -p tax.man
```

Similarly, the sdetable command can be used to rebuild indexes when its table has become fragmented:

```
sdetable -o rebuild_index -t parcels -x all -i 9000 -t parcels -u
dataowner -p tax.man
```

The -x all flag instructs the sde application to rebuild all indexes on this table.

Granting access to data

Once you have the data loaded, grant other users access to the data for read (select) and write (insert, update, delete) operations. Initially, only the user who has created the business table has access to it. To make the data available to others, the owner of the data must grant permissions to other users. The owner can use the sdelayer command to grant permissions. Privileges can be granted to either another user or to a role.

In this example, a user "eric" gives a role "readers" select privileges on a feature class "depths".

```
sdelayer -o grant -l depths,shape -U readers -A SELECT -u eric -p
sea.floor -i 5151 -D oceans -s GISData
```

In this example, full permissions are granted to the editor's role on table ocean_temps:

```
sdetable -o grant -t ocean_temps -U editors -A SELECT,INSERT,UPDATE,DELETE
-u eric -p sea.floor -i 5151 -D oceans -s GISData
```

To grant permissions on a raster dataset or embedded raster catalog, use the sdetable -o grant command. In this example, select permission is granted a role on a raster dataset:

```
sdetable -o grant -t seatemps -U ocean_viewers -A select -i 9000 -u eric -
p sea.floor
```

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 same privilege.

Revoking access to data

To revoke access rights to data to a user or a role, use the -o revoke option of sdelayer or sdetable. The command operates in the same fashion as -o grant. In this example, user

"eric" revokes "select" permission from the "readers" role against the "depths" feature class:

```
sdelayer –o revoke –l depths,shape –U readers –A select –u eric –p
sea.floor –i 5151 –s GISData
```

To revoke access to a table or raster, use sdetable -o revoke. This example revokes access to a raster dataset:

```
sdetable –o revoke –t seatemps –U ocean_viewers –A select –i 9000 –u eric
–p sea.floor
```

Exporting data

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

sdeexport—Creates an ArcSDE export file to easily move feature class data between ArcSDE instances. Also supports rasters.

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 an relational database management system (RDBMS) table

These tools export ArcSDE simple features only; you cannot export geodatabase feature-linked annotation or topologies.

Dropping data

Business tables and layers can be removed from the database with the sdetable -o delete option. This command will drop the table or layer and remove any registration information from the ArcSDE metadata. An example of dropping a table (not registered with the geodatabase):

```
sdetable –o delete –t ocean_temps –u eric –p sea.floor –i 9000 –D
```

NOTE: This only removes the layer, not the business table.

```
sdelayer –o delete –l parcels,shape –i 9000
```

NOTE: If you build your data with ArcGIS and drop it with sdetable -o delete, you will not clean up the geodatabase metadata. You must use ArcGIS tools (ArcCatalog, ArcObjects) to drop data if you created data with it.

To drop a raster, use sderaster -o drop:

```
sderaster –o drop –t seatemps –i 9000 –u eric –p sea.floor
```

Using -o drop with the -t <business table> argument will drop the entire raster dataset. If you want to retain the business table and only remove the raster column, use -o drop -l <raster dataset>,<image column name>.

Defining ArcSDE views

Views are stored queries or virtual tables. A view's select statement is stored in the database instead of the table. SQL Server 2000 allows you to create unique clustered indexes on views and partition views across multiple servers.

Indexed views are stored in the database in the same manner as a table, while nonindexed views store only the SQL statements comprising them.

With ArcSDE, it is possible to define views between two feature classes or between a feature class and a table or create more complex views containing subqueries or that span databases. ArcSDE views are created with the sdetable -o create_view command. When you create an ArcSDE view, three views are created: business table, feature table, and spatial index.

Creating an ArcSDE view

Use the sdetable -o create_view command to create a view and the sdetable -o delete command to remove one. The syntax of sdetable -o create is:

```
sdetable -o create_view
    -T <view_name>
    -t <table1,table2...tablen>

    -c <table_col1,table_col2...table_coln>

    [-a <view_col1,view_col2...view_coln>]
    [-w "where_clause">]
    [-i <service>]
    [-s <server_name>]
    [-D <database>]
    -u <DB_User_name> [-p <DB_User_password>] [-N] [-q]
```

You must list the columns (-c) you want in the view as well as a where clause (-w) within the sdetable command. Be sure to include the business table's spatial column to make the view spatial. You must have create view permissions to execute this command.

This example creates a view of all U.S. counties with a 1990 population per square mile greater than 50. The -s option specifies a server.

```
sdetable -o create_view -T CountyPopView -t us_counties -c
shape,name,state_name -w "pop90_sqmi > 50" -s GISCensus -u census -p
00census#people
```

This example uses a subquery to select the state whose capital city has the largest male population.

```
sdetable -o create_view -T ManView -t us_states -c
us_states.shape,us_states.state_name --w "us_states.state_name = (select
us_capitals.state_name from us_capitals where p_male = (select max(p_male)
from us_capitals))" -s GISCensus -u census -p 00census#people
```

You can also alter the business table's view directly. In this example, a simple sde view is created with this statement:

```
sdetable –o create_view –T taxpars –t taxlots –c taxlots.shape
```

Alter this view in the Query Analyzer to join to another table:

```
ALTER  VIEW vwtaxlots
AS
SELECT  tx.shape, p.owner1, p.owneradd, p.ownercity,
 p.ownerstat, p.bldgval, p.landval, p.totalval,
 p.bldgsqft
FROM  taxlots tx
inner join taxlotinfo p on tx.shape = p.shape
GO
```

Cross database views

In the multidatabase model it is possible to create views between tables and SDE feature classes that do not reside in the same database. When you do this, you must correctly qualify the tables involved in your queries and use the -D (database) switch to identify which database contains the data on which to create a view.

In the single spatial database model, you cannot have cross database views between two feature classes in different databases. However, you can create a cross database view between a feature class in one database and a table in another.

To create a cross database view involving a feature class and a table, create a simple SDE view of the feature class first, then alter it in Query Analyzer to query the tables in the other database.

Building on the American National Standards Institute (ANSI) SQL example above, create a simple sde view:

```
sdetable –o create_view –T taxpars –t taxlots –c taxlots.shape
```

Now alter this view in Query Analyzer to add in an ANSI join to another table:

```
ALTER  VIEW vwtaxlots
AS
SELECT  tx.shape, p.owner1, p.owneradd, p.ownercity,
 p.ownerstat, p.bldgval, p.landval, p.totalval,
 p.bldgsqft
FROM  taxlots tx
inner join parcelinfodb.dbo.taxlotinfo p on tx.shape = p.shape
```

Creating indexed views (SQL Server 2000)

Indexed views are a new feature of SQL Server 2000 and are advantageous when your view definition employs a complex join. They are a good choice on static data as they are schema bound to their source. Also, they are stored in the same way as a regular table with a clustered index. Before you try this, make sure you are familiar with the SQL Server Books Online documentation covering indexed views.

Creating an indexed view in SQL Server 2000 requires extra work and has many conditions. These conditions are summarized as follows:

- The view must be created with Schemabinding.
- The view must be created with QUOTED_IDENTIFIER on.
- The view cannot reference other views.
- Tables referenced in views must be two part.
- You cannot reference all columns of a table with *.

There are additional rules and conditions. Refer to the SQL Server Books Online under the topic 'Creating an Indexed View' for more information.

As ArcSDE does not create schema bound views by default, follow this process:

1. Create an ArcSDE view with sdetable -o create_view.
2. Open SQL Server Query Analyzer and drop the business table view with drop view <view name>. Before you do this, make sure you copy the view definition.
3. Re-create the business table view with the create view statement, but make sure that you use the WITH SCHEMABINDING option and qualify your view name and table name with the owner. <table or view name> convention.
4. Create a unique clustered index on the view with the create unique clustered index statement.

This is an example that creates an indexed view. This example was done in SQL Server 2000 using a SQL Server authenticated user.

1. Create the view with sdetable -o create_view:

```
C:\sde90\etc>sdetable -o create_view -T vwWithIndx -t cagis_cent -c
parcelid,object__id,x_coord,y_coord,book,page,parcel,shp -w
"parcel>0393" -u census -p 00census#people -i sql90k -s GISCensus -D
parcels
```

2. Query Analyzer—Copy the view statement, then drop the view of the business table.

```
drop view vwWithIndx
```

3. Re-create the view. Make sure you qualify the view and table names with the owner name and use the WITH SCHEMABINDING statement.

```
CREATE VIEW vtest.vwWithIndx
WITH SCHEMABINDING AS SELECT  parcelid, object__id, x_coord, y_coord,
book, page, parcel, shp FROM  vtest.cagis_cent WHERE parcel>0393
```

4. Create a unique clustered index on this view:

```
create unique clustered index shpIdxView on vtest.vwWithIndx (shp)
```

Choosing an ArcSDE log file configuration

ArcSDE allows you to configure the allocation of ArcSDE log files to your users. You can allow your users to own their own log files, or they can check out a log file from a pool of log files owned by the sde user. Log files can be shared, session-based, or 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 are exhausted, ArcSDE will attempt to create a session-based log file if it is allowed; otherwise, a shared log file is created. If the shared log file cannot be created, ArcSDE returns an error.

Shared ArcSDE log files

Shared log files are shared by all sessions that connect as the same user. Essentially, all sessions insert and delete records from the same log file data table. The log files are created the first time any session connects and remain in the user's schema. Shared log files require a connecting user to have CREATE TABLE in the database. To configure your server to use only shared log files, use the sdeconfig -o alter command to 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, even if all connections are made as the same user.

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

```
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 log files

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

Using an sde user pool of ArcSDE log files

The sde user can create a pool of log files that can be checked out and used as either session-based or 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 formulas:

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 user's schema. If the log file cannot be created, an error is returned.

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

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

CHAPTER 5

Connecting to SQL Server

You can operate ArcSDE under three different configurations: three tiered with your clients connecting to a middle tier (giomgr) service; two tiered with your clients making direct connections to the database; and mixed, in which your server accepts both direct and three-tier connections. This chapter covers how to make your initial connection to ArcSDE under a three-tiered configuration, how to connect in a two-tier configuration, and how to operate only in a two-tier environment without installing ArcSDE.

Connection modes

Application server

The application server is the ArcSDE service that can run on the server hosting SQL Server, or a remote server that connects to a remote SQL Server. The application server is the giomgr.exe process that spawns gsrvr.exe processes on that same server. This type of connection is also known as three tier.

Direct connect

The functionality of the application server is also embedded into the ArcSDE client. The files comprising the application server and gsrvr are sdesqlsrvr90.dll and gsrvrsql90.dll. In direct connect, the application server processes (giomgr.exe and gsrvr.exe) run in process on the client machine. This is also known as two tier.

Mixed mode

Mixed mode implies both an application server and direct connect clients are used to connect to the spatial database. ArcSDE spatial databases are built such that you can share databases across multiple ArcSDE services (application servers) and direct connect clients.

Making your first connection

If you have your ArcSDE service started, then you have already successfully made a connection to the data store (as the sde user). To successfully connect to an ArcSDE data store as a non-sde user, these conditions must be true:

1. The connecting user has permission to access the spatial database.
   a. Usernames in a spatial database cannot use reserved words such as `min`, `max`, `national`, or `select`. Refer to the SQL Server Books Online for reserved keywords.
   b. You are no longer required to have create table permission in a database unless they wish to create data.
   c. An sde database (to store ArcSDE system tables) is no longer required. ArcSDE system tables can be stored in any database.
2. If the connecting user will own spatial data—load spatial data with the ArcGIS data loaders or the admin tools, such as shp2sde or sdeimport—that user must have create table and create procedure permissions in the spatial database.

Example: Connecting to ArcSDE from ArcCatalog

Open ArcCatalog. Navigate to the Database Connections node. Expand it and select Add Database Connection. Double-click this to bring up the connection properties dialog box. Under server, type the name of the ArcSDE server. For Service, type either your service instance name or your TCP/IP port (for example, 5151). For Database, type the spatial database name. Type a username and password for the remaining text boxes.

Example: Connecting to ArcSDE through ArcObjects

To use this C# sample, reference ESRI.ArcGIS.DataSourcesGDB, ESRI.ArcGIS.Geodatabase, ESRI.ArcGIS.System in your VS.NET Project.

```
PropertySetClass pProp = new PropertySetClass();
IWorkspaceFactory workfact = new SdeWorkspaceFactoryClass();

pProp.SetProperty("server","GISData");
pProp.SetProperty("instance","5151");
pProp.SetProperty("database","fisheries");
pProp.SetProperty("user","Eric");
pProp.SetProperty("password","sea.floor");
pProp.SetProperty("version","sde.DEFAULT");
try
```

```
{
        IWorkspace pWork = workfact.Open(pProp,0);
        Console.WriteLine("connected to " + pWork.Type);
}
catch (System.Exception e)
{
        Console.WriteLine(e.Message + "\n" + "Cannot connect");
}
```

Using the ArcSDE direct connect driver

To directly connect to your database, change the service (aka instance) property of your sde connection to sde:sqlserver:<sqlserver instance name>. You must specify a database name as well. For example:

```
sdelayer –o describe –i sde:sqlserver:GISData –D fisheries –u Eric
```

To connect to a named instance of SQL Server, input the named instance in the third part of the -i argument:

```
sdelayer –o describe –i sde:sqlerver:GISData\data02 –D fisheries –u Eric
```

Understanding direct connect

The ArcSDE application server technology is also embedded into the ArcSDE client software. Any application built with the ArcSDE C API, including ArcGIS and ArcIMS, can bypass the ArcSDE application server to directly connect to a Microsoft SQL Server instance, provided that instance has the ArcSDE system tables.

The ArcSDE application server converts spatial requests from an ArcSDE client into SQL statements. The relational database executes the statements and returns result sets to the application server. The application server, in turn, packages this result set into the ArcSDE communication stream and ships them to the client. In a direct connection, the application server runs within the same process as the client program on the ArcSDE client machine.

Why use direct connect?

In direct connect, the application server runs as a thread within the client's executing process. This means that resources used normally by the gsrvr.exe process on the application server are free to any process requiring them. Therefore, use direct connect as a way of offloading a busy server. Employ direct connect when the client machine's CPU is significantly faster than that of the server.

When using direct connect, some operations may be slower, particularly those that process large quantities of data within the application server. Always employ a simple benchmark of both modes before putting direct connect into production.

Making a direct connection

Whether or not you have an ArcSDE service running or have installed the ArcSDE software, you are ready to make a direct connection. Here's what you must do:

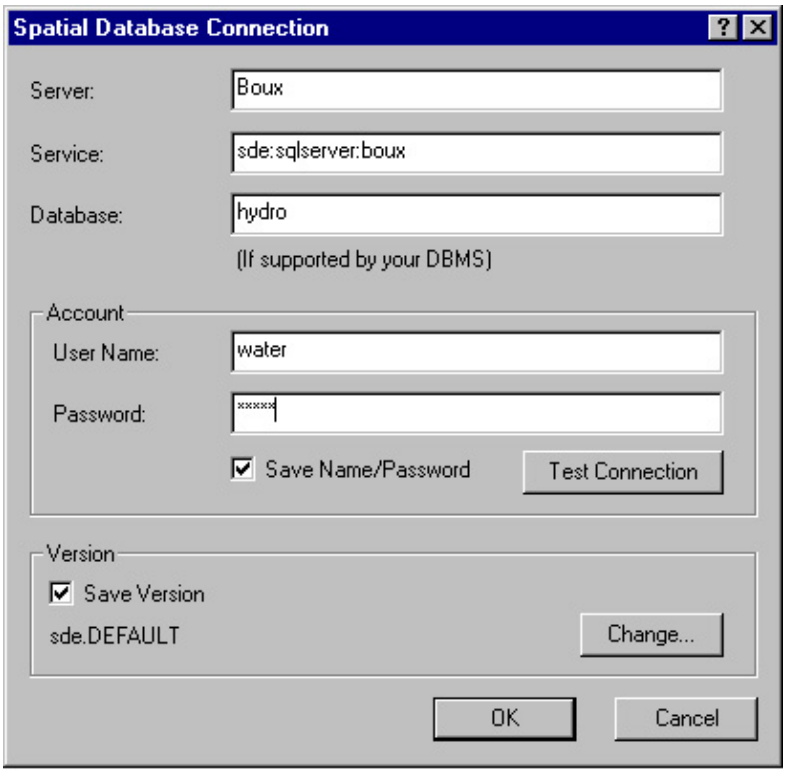
1. Ensure that you have a valid installation of the Microsoft Data Access Components (MDAC) on the client machine. To connect to an ArcSDE spatial database using an ArcSDE 9 client, such as ArcIMS or ArcGIS, you should have the latest release of MDAC installed on that client. Check msado15.dll in \program files\common files\system\ado or download the component checker from http://www.msdn.microsoft.com and search for Data Access Downloads.
2. Set the sdehome environment variable to the directory above the bin directory holding the gsrvrsql90.dll; the sdesqlsvr90.dll; and the client libraries sg90.dll, sde90.dll, and pe90.dll.
    a. You can substitute SDEHOME with ARCHOME.
    b. From an ArcGIS client, set neither and let the search default to: `HKEY_LOCAL_MACHINE\SOFTWARE\ESRI\CoreRuntime\InstallDir`.
3. Change the service (aka instance) argument of your connection to sde:sqlserver:<SQL server instance name>
    a. Sde:sqlserver:GISData\data02

For example, to direct connect from ArcCatalog, where ArcGIS resides in c:\arcgis, C:\arcgis\engine\bin contains gsrvrsql90.dll; sdesqlsvr90.dll; and the three client libs, sg90.dll, pe90.dll, and sde90.dll, do one of the following:

1. Set SDEHOME to c:\arcgis\engine and connect.
2. Clear SDEHOME, set ARCHOME to c:\arcgis\engine and connect.
3. Clear SDEHOME and ARCHOME, and let the ArcGIS client find the path from the registry under `HKEY_LOCAL_MACHINE\SOFTWARE\ESRI\CoreRuntime\InstallDir`.

Now Connect:
- Server name = ArcSDE Server
- Instance = sde:sqlserver:<SQL Server Instance Name>
- Database = database name (required parameter)
- User = any valid DBMS/ArcSDE user
- Password = any valid DBMS/ArcSDE user's password

Change the service parameter to direct connect from ArcCatalog:



ArcObjects could connect with this:

```
Dim pPropSet as iPropertySet
Dim pWorkspaceFactory as iWorkspaceFactory
Dim pWorkspace as iWorkspace

set pPropSet = new PropertySet
set pWorkspaceFactory = new SDEWorkspaceFactory
with pPropSet
        .setProperty "Server" = "GISData"
        .setProperty "Instance" = "sde:sqlserver:GISData"
        .setProperty "Database" = "fisheries"
        .setProperty "User" = "Eric"
        .setProperty "Password"="sea.floor"
        .setProperty "version" = "sde.DEFAULT"
end with
set pWorkspace=pWorkspaceFactory.Open(pPropSet)
if pWorkspace is nothing then
        msgbox "Connection Failed"
else
        msgbox "Success"
end if
```

Connecting from ArcIMS

The ArcIMS spatial server can use ArcSDE direct connect. However, you must author your ArcXML file first using an ArcSDE application server connection, then edit the file, adding in the sde direct connection string. Here's how:

1. Use ArcExplorer—Java/ArcIMS author to connect to an ArcSDE application server (traditional 3-tier service via TCP/IP).
2. Create your AXL.
3. Edit your AXL:
   a. Open the AXL in notepad or any other text editor.
   b. Locate the `SDEWORKSPACE` tag.
      i. Change the instance attribute from
         1. `port:number` (i.e., port:9000)
         2. to `sde:sqlserver:<sql instance name>` (i.e., sde:sqlserver: GISData)
      ii. Make sure you reference a database name in the tag: database="fisheries"
   c. Save the AXL.
4. Create the service in ArcIMS.

For example:

Change

```
<SDEWORKSPACE name="sde_ws-0" server="GISData" instance="port:9000"
database="fisheries" user="Eric" encrypted="true" password="UIUX"
geoindexdir="C:\DOCUME~1\eric\LOCALS~1\Temp\" />
```

To

```
<SDEWORKSPACE name="sde_ws-0" server="GISData"
instance="sde:sqlserver:GISData" database="fisheries" user="Eric"
encrypted="true" password="UIUX"
geoindexdir="C:\DOCUME~1\eric\LOCALS~1\Temp\" />
```

CHAPTER 6

National language support

Storing data in an ArcSDE SQL Server database using collation sets other than the SQL Server default, Latin1_General, requires some extra configuration on both the client and the server. This section provides guidelines for configuring both the SQL Server database and the ArcSDE client environment to enable the use of collation sets other than Latin1_General.

SQL Server database collation designator

The default collation designator for SQL Server 2000 databases is Latin1_General. The collation is selected when SQL Server 2000 is installed but can be changed later by running the rebuildm.exe utility. This utility is documented in the SQL Server Books Online. Running it will remove any existing databases. SQL Server 2000 also supports different collations set at the database level. This enables a single SQL Server to have a default collation with numerous databases, each set to a different collation.

The default collation will support U.S. English and most Western European languages. Change this default only if you are not using a language from this group. Consult the SQL Server Books Online to determine the collation designator that is appropriate for your data. Also, select how data will be sorted. Sort order can be binary, case sensitive, accent sensitive, kana sensitive, or width sensitive. If the binary option is not selected, SQL Server 2000 follows sorting and comparison rules as defined in dictionaries for the associated language or alphabet. It is also possible to choose an alternative collation if backward compatibility with previous installations of SQL Server 7 or earlier is desired. The choice depends on the collation choice made in the earlier instance of SQL Server. If the previous installation accepted the defaults, select Dictionary Order, Case Insensitive. This is backward compatible with the 1,252 character set, which was the default in SQL Server 6.5 and SQL Server 7.

NOTE: ArcSDE reads collation from each spatial database. If you use the older multidatabase model (in other words, you have multiple spatial databases with an sde database as repository), collation is read from the master database.

Setting the SDE_SQLCHARSET variable for Windows clients

If you want to load data into or retrieve data from an ArcSDE spatial database using non-ArcGIS clients, and those clients are configured to different character encoding standards or code pages, you may have to set the SDE_SQLCHARSET environment variable.

For instance, an ArcSDE SQL Server spatial database using the Shift JIS character encoding for Japanese characters would require that command line tools such as sdeimport or shp2sde run from an environment set to the same encoding (Shift JIS). If that client is not set to the same encoding, you must set the SDE_SQLCHARSET variable to be able to send and receive character data to and from the ArcSDE Server. For a C API or ArcSDE command utility application, the SDE_SQLCHARSET must be set as a system environment variable.

```
set SDE_SQLCHARSET=eucjis
```

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

```
java -DSDE_SQLCHARSET=eucjis [Java client program]
```

Exercise care in setting SDE_SQLCHARSET on Windows. In Windows, there are two different code page environments, Windows ANSI and Original Equipment Manufacturer (OEM). Windows applications such as ArcInfo and ArcView run in the ANSI code page environment. ArcSDE administration tools and C API applications invoked from the MS–DOS command prompt run in the OEM code page environment. If you use both Windows applications and MS–DOS applications together, then set the SDE_SQLCHARSET variable for MS–DOS applications by opening the MS–DOS command prompt and issuing an MS–DOS SET command. Initially verify that the data to be loaded is in either ANSI or OEM format.

If the ArcSDE Windows client, using Baltic language as the local language, is running from Windows, set the code page to cp1257.

Also, ArcSDE—Java API client-based applications run from the DOS command window require the setting of the file.encoding Java System property, if the DOS OEM file.encoding property is different from the file.encoding used by applications run in the Windows environment. Consider for example, that the Windows client machine is set up in the French Locale (cp1252) and that the SQL Server database contains French data, created using code page 1252. The OEM code page required to read this data is cp850. To read and display all French data, the file.encoding Java System property must be set at the DOS command prompt as shown below:

```
java -Dfile.encoding=CP850 -DSDE_SQLCHARSET=CP850 <Java program name>
set SDE_SQLCHARSET=cp1257
```

If the client is running from the MS–DOS prompt, set the code page to cp775, which is the OEM code page for Baltic language.

```
set SDE_SQLCHARSET=cp775
```

If the ArcSDE client is running from a UNIX platform, set the code page to iso_13

```
set SDE_SQLCHARSET iso_13
```

Note that there should not be any blank spaces at the end of the command.

Character encoding standards supported by ArcSDE

Currently ArcSDE only supports conversions between the character encoding standards listed in the following list.

Character encoding	Description
big5	BIG5 16-bit Traditional Chinese
Euccns	EUC 32-bit Traditional Chinese
eucgb	CGB2312-80 16-bit Simplified Chinese ZHS16CGB231280
sjis	Shift-JIS 16-bit Japanese
eucjis	EUC 24-bit Japanese
eucksc	KSC5601 16-bit Korean
iso_1	ISO 8859-1 West European
iso_2	ISO 8859-2 East European
iso_3	ISO 8859-3 South European
iso_4	ISO 8859-4 North and North–East European
iso_5	ISO 8859-5 Latin/Cyrillic
iso_6	ISO 8859-6 Latin/Arabic
iso_7	ISO 8859-7 Latin/Greek
iso_8	ISO 8859-8 Latin/Hebrew
iso_9	ISO 8859-9 West European and Turkish
Iso_10	ISO 8859-10 North European
iso_13	ISO 8859-13 Baltic
Iso_15	ISO 8859-15 West European
cp437	IBM-PC Code Page 437 8-bit American
cp850	IBM-PC Code Page 850 8-bit West European
cp851	IBM-PC Code Page 851 8-bit Greek/Latin

cp852	IBM-PC Code Page 852 8-bit East European
cp855	IBM-PC Code Page 855 8-bit Latin/Cyrillic
cp857	IBM-PC Code Page 857 8-bit Turkish
cp860	IBM-PC Code Page 860 8-bit West European
cp861	IBM-PC Code Page 861 8-bit Icelandic
cp862	IBM-PC Code Page 862 Hebrew
cp863	IBM-PC Code Page 863 8-bit Canadian French
cp864	IBM-PC Code Page Arabic
cp865	IBM-PC Code Page 865 8-bit Norwegian
cp866	IBM-PC Code Page 866 8-bit Latin/Cyrillic
cp869	IBM-PC Code Page 869 8-bit Greek/Latin
cp720	Arabic Transparent ASMO
cp737	IBM-PC Code Page 737 8-bit Greek/Latin
cp775	IBM-PC Code Page 775 8-bit Baltic
cp1250	MS Windows Code Page 1250 8-bit East European
cp1251	MS Windows Code Page 1251 8-bit Latin/Cyrillic
cp1252	MS Windows Code Page 1252 8-bit West European
cp1253	MS Windows Code Page 1253 8-bit Latin/Greek
cp1254	MS Windows Code Page 1254 8-bit Turkish
cp1255	MS Windows Code Page 1255 8-bit Latin/Hebrew
cp1256	MS Windows Code Page 1256 8-Bit Latin/Arabic
cp1257	MS Windows Code Page 1257 8-bit Baltic
cp1258	MS Windows Code Page 1258 8-bit Vietnamese

CHAPTER 7

Backup and recovery

Backup and restore are critical functionality to understand and implement for the security of your data. Microsoft SQL Server offers multiple backup strategies and restoration types. ArcSDE 9 single spatial databases can be backed up and restored in the same manner as any nonspatial database.

Understanding backup and restore

Backup and restore are designed to minimize exposure to data loss and database downtime. A backup is a full or partial record of data or changes within a database or transaction log. Data restoration is the process of applying backups followed by database recovery. Once a database has recovered, you can no longer restore backups, unless you begin the process over. Data loss potential is governed by a database's recovery model and backup frequency. Database downtime is mitigated by the number and size of backups that are restored to SQL Server.

Understanding recovery models

Recovery models dictate how the transaction log is used and a database's exposure to data loss. There are three models: full, bulk-logged, and simple. Recovery model is a database setting, changed either through the Enterprise Manager or the alter database statement. For example:

```
Alter database gis_data set recovery bulk_logged
```

Full recovery model

Full recovery allows point in time (or point of failure) recovery, provided you have the transaction log backups at that point in time. Point in time recovery signifies the ability to

recover a database to a specific time, restoring all committed transactions and rolling back all incomplete transactions.

To support this type of restore, all operations are logged to the SQL Server transaction log. As the SQL Server transaction log becomes critical for data restoration, it must be backed up regularly. Since all operations are fully logged, the transaction log is active and will grow. You control this growth through transaction log backups.

Production SQL Servers that must have point-in-time recovery should use the full recovery model for their databases. Such databases should follow this general backup strategy:

- Make periodic full database backups.
- Make more frequent differential database backups.
- Back up the transaction log several times between differential and full database backups.

Recovery to a point of failure or point in time involves:

- Backing up the currently active transaction log
- Restoring the most recent full database backup without recovery
- Restoring the most recent differential database backup without recovery
- Restoring in sequence any transaction log backups without recovery
- Restoring the last transaction log backup with recovery

Bulk-logged recovery model

Databases using the bulk-logged recovery model minimally log bulk operations to the Microsoft SQL Server transaction log. These operations include CREATE INDEX, SELECT … INTO, writetext, updatetext, and BULK INSERT. The transaction log does not record sufficient information to recover these changes if media failure occurs after a bulk operation. You can recover a database to the point of failure, but your data may not be consistent if it was changed by a bulk operation. The process of restoration is the same as in full database recovery.

ArcSDE 9 log files are populated using BULK INSERT statements. Set your recovery model to bulk-logged to minimize logging of inserts into log files if your work flow performs extensive selections.

Simple recovery model

Simple recovery model does not use the transaction log for recovery. You can only restore full database backups if you use the simple recovery model. The advantage to using the simple recovery model is that there is less transaction log management. SQL Server will begin to manage the truncation of the transaction log after you perform your first full database backup. After this, if the log fills to 70 percent, or a database checkpoint occurs (upon alter database, for example), SQL Server will attempt to

truncate the transaction log. Space for new transactions becomes available as log truncation clears out transactions that precede the oldest active transaction and reside in an earlier virtual log. Use the backup log statement to manually truncate the log:

```
Backup log spatial with truncate_only
```

Understanding backup types

You back up data, changes, or transaction logs. The difference between the three depends on how much you are willing to lose in the event of system failure. Database, file group, and file backups back up the entire entity. Differential backups back up only changes made to the data since the last full database, file group, or file backup. Transaction log backups back up only the transaction log.

Database, file group, and file backups

Database, file group, and file backups are full copies of that entity. They are the foundation for any backup and restore strategies.

Differential database, file group, and file backups

Differential backups record changes made to a database, file group, or file since the last full database backup. Changes are identified through the differential changed map, which represents all changed extents in a database. If an extent's value is 1 within the map, that extent is backed up. During the next full backup, any changed values within the map are set back to 0.

Transaction log backups

Transaction log backups back up the transaction log and control the log's size. Transaction log backups are only useful in full or bulk-logged recovery models.

During a transaction log backup, the entire log is first backed up. All committed or rolled back transactions that precede the oldest active transaction (MinLSN) and reside in a previous virtual log are truncated from the transaction log. This controls the size and growth of the transaction log. In simple recovery model, in which the transaction log cannot be used for recovery, backup is used to truncate the log.

Backing up ArcSDE spatial databases

ArcSDE single spatial databases can be backed up and recovered as any other SQL Server database. If you have deployed a multidatabase sde instance (one sde database holds metadata for all spatial databases) the process becomes more complex. In this configuration, all spatial databases depend on the sde database, so you must back up all databases as one. The following sections detail how you should plan for backup and recovery of your ArcSDE single spatial databases.

Deciding on a recovery model

Review the sections above and the SQL Server Books Online, topic "Administering SQL Server—Backing Up and Restoring Databases" and choose a recovery model for your spatial database.

If your data will never change or changes sporadically through batch imports (sdeimport, and so forth), choose simple recovery.

If you can tolerate some data loss exposure, index creation for example, choose bulk-logged recovery. The advantage to using bulk-logged recovery is that selections into the ArcSDE log files are minimally logged. Both ArcGIS and ArcIMS use sde log files to persist selected features. If your work flow includes large selections, consider setting your database recovery model to bulk-logged.

If you must have point-of-failure or point-in-time recovery use the full recovery model.

Once you've made your choice, set your recovery model in your spatial database. From Query Analyzer:

```
-- for bulk logged recovery model:
Alter database spatial set recovery bulk_logged
-- for full recovery model:
alter database spatial set recovery full
-- for simple recovery model:
alter database spatial set recovery simple
```

Designing a restoration strategy

In the event of a catastrophic system failure, you may have to restore some or all of your databases. To facilitate this, design a restoration plan, then practice restoring your databases.

Creating a restoration plan

Create a document that details:

- All SQL Server installations, their patch levels, OS service packs, server names, database names, collations.
- For each SQL Server, document what databases are backed up, their recovery models, and their backup schedules. Include explanations for the recovery model and backup schedule.
- Document who performs the backups and who can serve as a point of contact for backup and restore questions.
- Decide and document where the backups will be stored. Also consider how long to retain backup media.
- Create a backup schedule.

Your restoration plan is useless unless you practice it. Make sure that you test your backups and your restoration plan before real problems occur.

Creating a backup schedule

Automate your backups using the SQL Server Enterprise Manager. Backups can be automated through SQL Server jobs or the SQL Server maintenance wizard.

If your database is set to simple recovery model, your data either does not change or changes infrequently. Therefore, you only need to backup your database after a change or once after you've built your data.

If your database is set to bulk-logged or full recovery modes, you'll have to make more frequent and varied types of backups. For data that changes on a daily basis, follow this general outline:

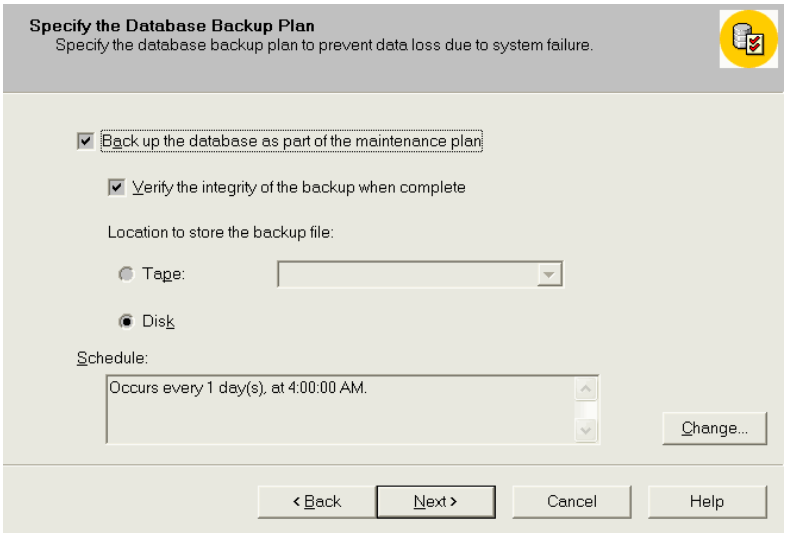
- Make a full database backup at the beginning of your production day.
- Make a differential database backup at the midpoint of your production day.
- Make periodic transaction log backups between your differential and full database backups.

Customize this outline to fit your organization's needs. This outline presumes your data changes rapidly throughout the day. If your data does not, change the timing of your backups to accommodate your change rate.

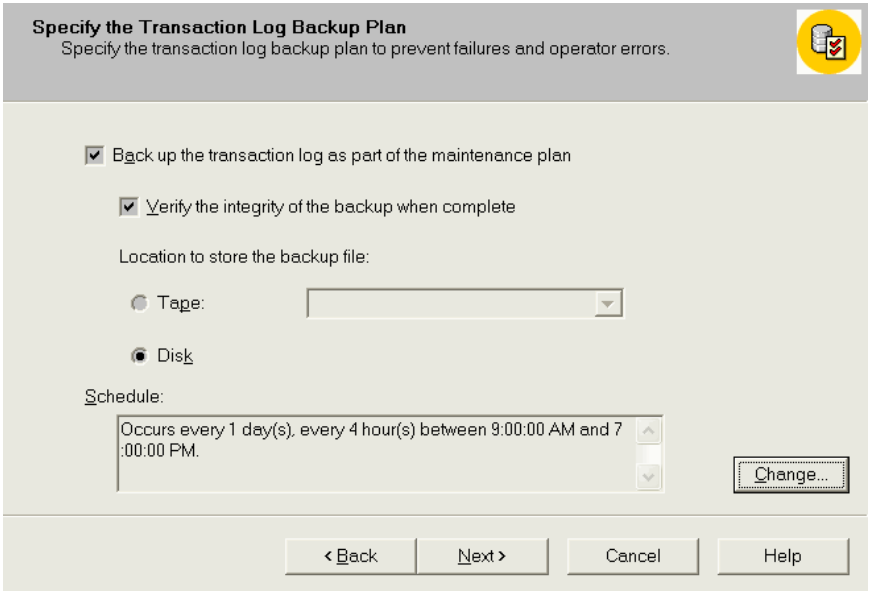
Creating a backup schedule with the SQL Server Enterprise Manager

The SQL Server Enterprise Manager has several ways of automating backups. The easiest method is to use the database maintenance wizard. The following example demonstrates how to use this wizard to create a schedule for a full database and transaction log backup.

1. Start the database maintenance wizard: Right-click your database in Enterprise Manager, select all tasks, then click Maintenance Plan.
2. In the Select Databases form, verify that your database is checked.
3. Click Next until you arrive at Specify the Database Backup Plan.



4. Change the schedule to whenever you want your full database backups to occur. In this example, the database will be backed up every day at 4 a.m. (4:00).
5. Click Next to proceed to the Specify Backup Disk Directory form. Use this form to specify where to place your backup files and how long to keep them.
6. Click Next to advance to the Specify the Transaction Log Backup Plan form. Use this form to schedule your transaction log backups.



7. In this example, the transaction log is backed up daily every 4 hours between 9 a.m. (9:00) and 7 p.m. (19:00).
8. Click Next to advance to the Specify Transaction Log Backup Disk Directory form. In this form, specify where to store your transaction log backups and how long to keep them.
9. Click Next through the remaining forms to specify reports and history. Finish the wizard to create your backup schedule.

Further references

SQL Server Books Online under "Administering SQL Server—Backing Up and Restoring Databases".

Restoring and recovering ArcSDE spatial databases

The goal of data recovery or database restoration is logical consistency of data. Logical consistency refers to the state of a database in which changes made by uncommitted transactions are not permanently written to disk. If a transaction is rolled back, any changes written to disk are also undone.

Disaster strikes!

Disk failures happen, servers malfunction, data loss occurs. Sooner or later, you will be confronted with a catastrophic failure from which you'll have to recover. If you use the simple recovery model, all you have to do is review your data restoration strategy and restore your most recent full database backup. If you use either the full database or bulk-logged recovery model, you'll have to review your restoration strategy and perform manual recovery.

Manual database recovery

Manual recovery involves the restoration of database backups to a SQL Server. When the last backup is applied, either full, differential, or transaction log backup, the database is allowed to "recover"—the process of rolling back uncommitted transactions and rolling forward committed transactions. Once a database has recovered, no further backups can be applied.

Restoring a spatial database

When a disk or server failure occurs, data may be lost. If data is lost, you must restore database backups and recover the database. Restoring an ArcSDE single spatial database is no different from restoring any other database. That process involves:

1. Determining a restoration sequence
2. Restoring the most recent full database backup

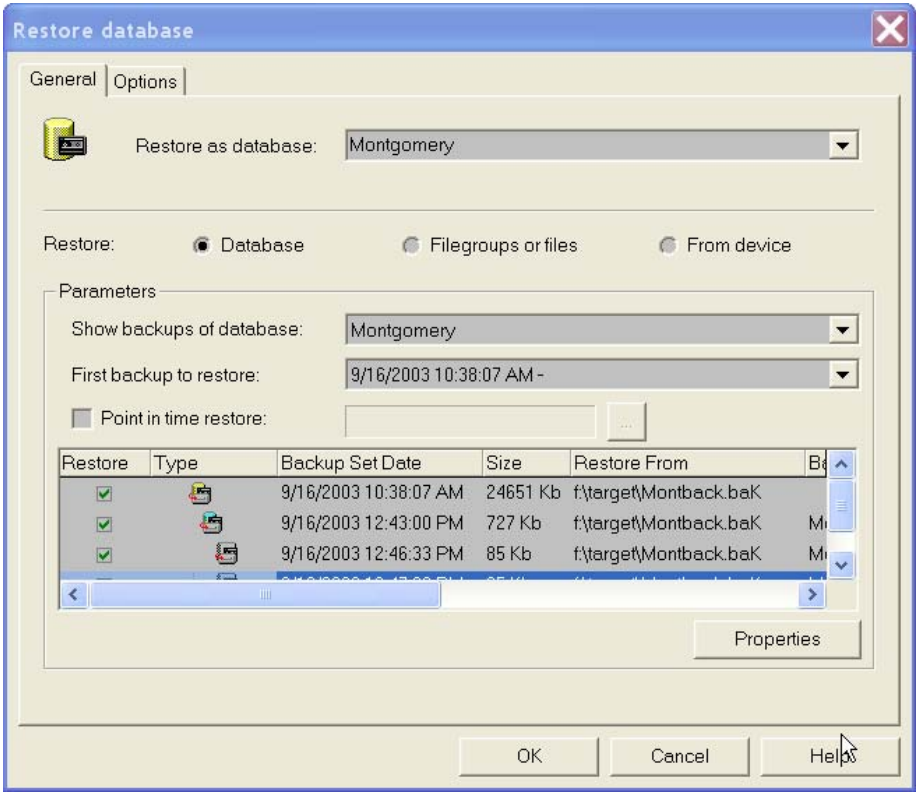
3. Restoring the most recent differential database backup
4. Restoring, in sequence, all transaction log backups since the most recent full or differential database backup
5. After restoring the last transaction log backup, allowing the database to recover

NOTE: You cannot change the name of the spatial database during your restore. You must use the same name the database had during its backup.

If your spatial database uses simple recovery model, points 3–5 above are not applicable.

Restoring a spatial database using Enterprise Manager

Backup and restore history are stored in the msdb database. If you will use Enterprise Manager to restore your database, and you have not lost the msdb database, Enterprise Manager will prompt you with all necessary information and proper sequence of restorations. To begin a restore, right-click the database in question, select all tasks, Restore database. You'll see a form like the one shown below:



Click OK to begin restore. Enterprise Manager will restore all backups in their proper sequence and recover the database after the last transaction log backup is applied. Once completed, your database is fully restored and recovered.

Restoring a spatial database using Transact-SQL

Using Transact-SQL to restore your database is more involved. To determine your restoration sequence, you'll have to read the backup device (Montgomery.bak) to determine its content. In this example, the Montback.bak file is read to determine what it contains, then a proper restoration sequence can be determined. Use the restore headeronly command to determine what a backup device contains:

```
restore headeronly from disk='f:\target\Montback.bak'
```

This returns the following result set:

	BackupName	BackupDescription	BackupType	ExpirationDate	Compressed	Position	Device
1	NULL	NULL	1	NULL	0	1	2
2	Montgomery backup	NULL	5	NULL	0	2	2
3	Montgomery backup	NULL	2	NULL	0	3	2
4	Montgomery backup	NULL	2	NULL	0	4	2

From the BackupType column, you know that Row 1 represents a full database backup, Row 2 a differential database backup, and Rows 3 and 4 are transaction log backups. You also know the order in which these backups were taken by the Position column. Row 4 in the result set, where Position = 4, was the last backup done. Your restore sequence will be Positions 1, 2, 3, and 4.

First you'll restore the most recent full database backup without letting the database recover. Identify the full database backup by its value in the position column of the RESTORE HEADERONLY statement. Use this value in the FILE= clause.

```
--most recent full db backup
restore database Montgomery from disk='f:\target\Montback.bak'
with file=1, norecovery
```

Next, restore the most recent differential database backup, again without letting the database recover. The differential backup's position from RESTORE HEADERONLY is 2, so you'll identify this backup set with the FILE=2 clause.

```
--restore the most recent differential db backup
restore database Montgomery from disk='f:\target\Montback.bak'
with file=2, norecovery
```

Finally, restore, in sequence, all transaction logs taken between the most recent differential database backup and the last transaction log backup. In this example, there are two transaction logs, at Positions 3 and 4, from the RESTORE HEADERONLY result set. Omit the "norecovery" clause from the final restore statement to allow the database to recover.

```
/* restore all t-logs, in sequence, between last differential db
 * backup and most recent t-log backup */

restore log Montgomery from disk='f:\target\Montback.bak'
with file=3, norecovery
```

```
--last t-log, leave off "norecovery" option to recover db
restore log Montgomery from disk='f:\target\Montback.bak'
with file=4
```

There are other ways of restoring a database and many other scenarios, including multiple backup devices, media sets, and so forth. This document cannot cover them all. Be sure to review other documentation, including the references listed above, before you go into production.

Managing transaction logs

Transaction logs are composed of a physical log, virtual logs, and a logical log. The physical log represents the transaction log file or log files on disk, while virtual logs are internal segments of those files. The logical log contains the active portion of the transaction log.

Transaction logs record transactions, data modifications, extent allocations and deallocations, and creation or destruction of tables and indexes. This record allows you to:

1. Roll back changes made within a transaction.
2. During recovery, roll back incomplete transactions initiated before a database failure.
3. During recovery, roll forward modifications listed in the log but not yet written to disk.

Each record written to the log has a Log Sequence Number (LSN). All new records are written to the end of the log and have a higher LSN than their predecessors. The logical log will grow until it consumes all open space in the physical log. This will cause the physical log to autogrow if allowed or throw an error. If the database uses the simple recovery model, the logical log will truncate itself when automatic checkpoints occur. If the database uses the full or bulk-logged recovery model, log truncation will occur after a BACKUP LOG statement completes.

The logical log cannot truncate itself past the Minimum Log Sequence Number (MinLSN), the oldest uncommitted transaction recorded.

Transaction logs are used to recover a database in the full or bulk-logged recovery models. The simple recovery model only allows restoration of a database to its last full backup. If you require point-in-time recovery of your data, use the full recovery model for your database. If your data will never change, use the simple recovery model. The bulk-logged recovery model performs minimal logging of certain operations and does not support point-in-time recovery.

If you have never made a backup of your database, you'll notice that your transaction logs have grown to a considerable size. Transaction logs in ArcSDE will grow due to data loading, editing, and making selections into sde log files. When you make selections with ArcGIS clients, such as ArcMap and ArcIMS, all selections exceeding 100 records are written into an sde log file.

Sde log files are database tables used to store selected features for use in other selection operations. Many other operations, such as reconcile from ArcGIS or metadata server searches, use sde log files. Rows written to an sde log file use the BULK INSERT statement, which makes them minimally logged in the transaction log if your database is set to the bulk-logged recovery model. If your database uses either the full or simple recovery modes, SQL Server logs more information for each BULK INSERT statement.

Sde log files incur deletes when clearing a selected set from the client software. The delete will either be a truncate table or delete from statement. Either way, the continuous inserting and deleting from these tables will make the database transaction log grow.

Backing up your database and log file should control the size of the log by truncating it. If you have not made regular backups of the transaction log, you'll need to take action to reduce its size. To reduce the amount of space used within the physical log file by the logical log, truncate it. To reduce the overall size of the physical log file, shrink it.

Truncating transaction logs

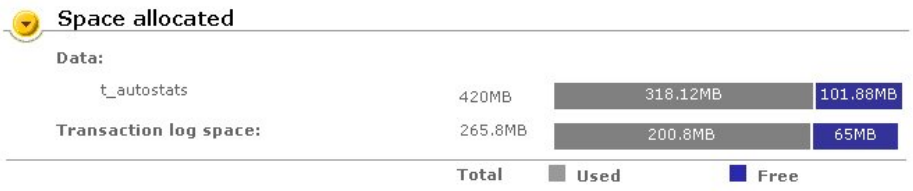
Truncating a log removes committed or rolled back transactions that precede the MinLSN of the active portion of the logical log. Truncating the log will not shrink the size of the physical log.

To truncate your log do one of the following:

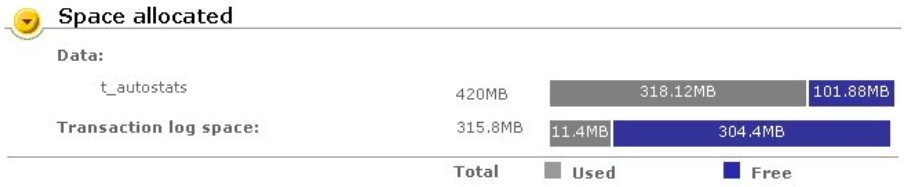
1. Back up the database using either Enterprise Manager or the Backup statement.
2. Back up the log file using Enterprise Manager or the Backup statement.
3. Use the simple recovery model. This model will automatically truncate your database whenever a checkpoint occurs or when the log fills to 70 percent.

Once you've issued the Backup statement, all records preceding the MinLSN are marked for reuse, and the reported size of the transaction log's logical log is smaller. The following figures demonstrate this.

A transaction log after multiversioning some large feature classes



The same log after a database backup. The log has been truncated.



Shrinking the transaction log

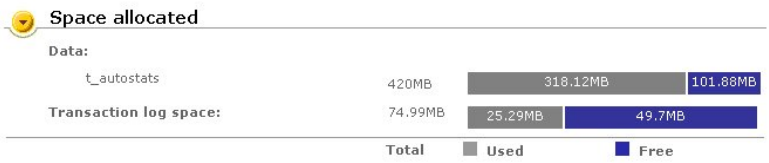
Truncating the log removes all nonactive transactions but does not reclaim any space, only delaying any more growth to the physical log file. In the lower case shown above, the physical log file still consumes 315.8 MB even though only 11.4 MB are active. You may find that you need some of this extra space. Shrinking the transaction log will return allocated space to the operating system.

To shrink the transaction log, do one of the following:

1. Use the Enterprise Manager's Shrink database utility: Right-click your database and select all tasks, Shrink Database. Input your desired resulting free space and click OK.
2. Use DBCC SHRINKDATABASE.

In the figure below, the Enterprise Manager was used to shrink the log file to 49.7 MB.

The log file's new size after shrinking it



Help! It won't shrink!

Under certain circumstances, you will not be able to shrink the size of the log file. As the logical log progresses through serial lists of Log Sequence Numbers, the active portion of the log will arrive at the end of the physical file. You cannot return space preceding the MinLSN to the operating system. You must advance the MinLSN to the beginning of the physical file. Here's how:

1. Back up the database. This will mark the preceding records to the minimum log sequence as reusable. Check the size of the log file.
2. Execute some updates using Transact-SQL. This should move the active portion of the log toward the beginning of the physical file.
3. Execute dbcc shrinkdatabase (dbname) to create a "shrinkpoint".

4. Back up the log file.

At this point you should see more free space in the log file. Repeat a few times if necessary.

Moving data using backup and restore

This section assumes you are going to backup an ArcSDE database and apply that backup to another server that does not have a preexisting ArcSDE installation.

1. Back up the spatial database you wish to apply to another server.
2. Copy the backup file or files to the new server.
3. Restore the backup file or files to the new server.
4. Run sp_change_users_login to synchronize the sysuser's table in each newly restored database with the master database's sysxlogins table.

NOTE: You cannot change the name of the database during restore. You must use the original database name.

1. Backup the spatial database

Use either Transact-SQL's BACKUP command or Enterprise Manager's backup utility to back up the spatial database you want to move. In this example, you'll use Transact-SQL to back up the database "Montgomery".

```
backup database Montgomery to disk='g:\Montback.bak'
```

2. Copy the backup file or files to the new server

In the example above, the backup file is located on the g: drive. Copy the backup file to the new server.

3. Restore the backup file or files to the new server

Using the restore command, you will restore the database on the new server and change its data file locations. The backup file (Montback.bak) has been copied to f:\target folder on the new server. When you restore the database, you want to store the data file in the f:\sdedata folder and the transaction log in the H:\transactionlogs folder. This means you must instruct SQL Server to move the data file and transaction log. To do this, you have to know the logical filenames within the backup file.

To determine the logical filenames within the backup file, execute RESTORE FILELISTONLY from the Query Analyzer on the new server:

```
restore filelistonly from disk = 'f:\target\Montback.bak'
```

This returns a result set that lists the logical filenames:

	LogicalName	PhysicalName	Type	FileGroupName	Size	MaxSize
1	Montgomery_Data	G:\Montgomery_Data.MDF	D	PRIMARY	31457280	35184372080640
2	Montgomery_Log	G:\Montgomery_Log.LDF	L	NULL	3211264	35184372080640

Using LogicalName from the above result set, execute the RESTORE DATABASE command with the MOVE option to restore the database:

```
RESTORE DATABASE Montgomery
 from disk = 'f:\target\Montback.baK'
 with MOVE
 'Montgomery_Data' to 'f:\sdedata\Montgomery_Data.mdf',
 MOVE
 'Montgomery_log' to 'H:\transactionlogs\Montgomery_log.ldf'
```

Once the restore finishes, open the database in Enterprise Manager and verify that your tables are there. Now you must synchronize the database users with their logins located in master.dbo.sysxlogins.

4. Run sp_change_users_login

When you restore a database from a backup created on another machine, there is a chance that a database user's identification number (either an SID or a GUID) does not match the same entry in the master.dbo.sysxlogins table. There is also a chance that the database user does not exist in the master.dbo.sysxlogins table. Since a restore also restores a database's sysusers table, it is possible for a record to exist here but not be present in master.dbo.sysxlogins. Hence, there are two scenarios you need to consider:

- The user exists as a login on the server but has a different identification (SID or GUID).
- The user does not exist (added as a login) on the server.

This does not apply to Windows authenticated logins.

Scenario 1: The user exists on both servers but has a different identifier.

In this example, both servers have a login "sde". After you restore the spatial database to the new server, the sde user's SID differs in the spatial database from the sde login's SID in the master.dbo.sysxlogins table.

```
Use Montgomery
go
Select SID from sysusers where name = 'sde'
Use master
go
select SID from sysxlogins where name = 'sde'

SID
------------------------------------------------------------
0x76695419BFAED41184FD00C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------
0xEDDFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)
```

To synchronize the two accounts, run sp_change_users_login. For exact usage, see the SQL Server Books Online. In this example, you'll use the "update_one" option to update just the sde user. You must run sp_change_users_login within the database you restored (or want to change the login for).

```
Use Montgomery
go
sp_change_users_login 'update_one','sde','sde'
```

Rerun the SID query above to verify your results:

```
0xEDDFCA8E56B0D411850000C04F8D0451
0xEDDFCA8E56B0D411850000C04F8D0451
```

Scenario 2: The user does not exist on the destination (restore) server.

This scenario differs from the first one in that a user exists in the database but not as a login in the master.dbo.sysxlogins table. This can happen when you create a login on a server, back up the databases on that server, then restore the databases to another server that does not have a record for this login. In this example, the spatial database is restored onto a machine that never had an "sde" login.

```
Use Montgomery
go
Select SID from sysusers where name = 'sde'
Use master
go
select SID from sysxlogins where name = 'sde'

SID
------------------------------------------------------------
0x76695419BFAED41184FD00C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------

(0 row(s) affected)
```

To rectify this, you must add an sde login to the server but not grant access to the spatial database. To add a login, use the sp_addlogin stored procedure or use the logins tool with Enterprise Manager:

```
sp_addlogin @loginame='sde',@passwd='sde.password'
```

Now run sp_change_users_login to synchronize the SID values in master.dbo.sysxlogins and the restored database's sysusers table. You must run sp_change_users_login in the spatial database since its sysusers table is out of sync with the master.dbo.sysxlogins table.

```
use Montgomery
go
sp_change_users_login 'update_one','sde','sde'
```

Now verify that the SIDs are equal by rerunning your SID query.

```
Use Montgomery
```

```
Go
Select SID from sysusers where name = 'sde'
Use master
Go
select SID from sysxlogins where name = 'sde'
SID
------------------------------------------------------------
0xF6DFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)

SID
------------------------------------------------------------
0xF6DFCA8E56B0D411850000C04F8D0451

(1 row(s) affected)
```

Once you've synchronized the users, you are ready to start the ArcSDE service.

Moving data using sp_detach_db and sp_attach_db

You can also move databases using the SQL Server stored procedures sp_detach_db and sp_attach_db. A detached database can be one file or several files, depending on how you created the database. The procedure is similar to the backup and restore examples, but differs in that you don't have a backup file of your database. Be careful! Once you attach your database, you don't have a copy of it, so if you delete it, you'll lose your data.

To move your database with sp_detach_db and sp_attach_db:

1. Detach your database from one server.
2. Copy the data files and transaction log files to the target server and reattach the database there with sp_attach_db.
3. Run sp_change_users_login to synchronize the sysuser's table in each newly restored database with the master database's sysxlogins table (SQL Server logins only).

1. Detach the database

Read the SQL Server Books Online for a complete discussion of sp_detach_db. When you detach the database, you can unhook it from SQL Server without having to shut down SQL Server. For example:

```
use master
sp_detach_db 'Montgomery'
go
```

2. Copy the database to another server and run sp_attach_db

The previously detached database will reside in its previous data folder. Copy it and its log file to the target server. Then execute the procedure sp_attach_db to reattach the database to the server:

```
sp_attach_db 'Montgomery',
@filename1='d:\SQL2K_DATAFILES\MSSQL\Data\Mont.mdf',
@filename2='d:\SQL2K_DATAFILES\MSSQL\Data\Montlog.ldf'
```

You must supply a pathname for the database files and log files.

3. Run sp_change_users_login to synchronize the sysuser's table

Follow the same procedure outlined above in step 4.

APPENDIX A

ArcSDE compressed binary

This appendix describes the state of ArcSDE with no configuration changes made to the server or any spatial databases. It also details how the ArcSDE application manages your data with regard to indexes, triggers, and specific SQL Server configuration settings. This appendix will help you understand how to manage spatial data.

Compressed binary

After the client verifies the geometry, it compresses and sends it to the server, where it is stored in compressed binary format in a feature table. Compressing the geometry on the client unloads the task from the ArcSDE server and reduces the transmission time to send the geometry to the ArcSDE server. Storing compressed geometry data reduces the space required to store data by as much as 40 percent.

Compressed binary metadata tables

A compressed binary feature class comprises three tables: the business table, the feature table, and the spatial index table.

The business table contains attributes and a spatial column. The spatial column is a key to the feature and spatial index tables.

The relationship between the business table and the feature table is managed through the spatial column and the FID column. This key, which is maintained by ArcSDE, is unique.

NAME	DATA TYPE	Allow Nulls
fid	INTEGER(4)	NOT NULL
numofpts	INTEGER(4)	NOT NULL
entity	SMALLINT(2)	NOT NULL

NAME	DATA TYPE	Allow Nulls
eminx	FLOAT(8)	NOT NULL
eminy	FLOAT(8)	NOT NULL
emaxx	FLOAT(8)	NOT NULL
emaxy	FLOAT(8)	NOT NULL
eminz	FLOAT(8)	NULL
emaxz	FLOAT(8)	NULL
min_measure	FLOAT(8)	NULL
max_measure	FLOAT(8)	NULL
area	FLOAT(8)	NOT NULL
len	FLOAT(8)	NOT NULL
points	IMAGE	NULL
anno_text	VARCHAR(255)	NULL

Feature table schema

The feature table stores the geometry, annotation, and CAD in the image type POINTS column. The internal SQL Server data type is listed in the table above. The ArcSDE data type for each column is defined below.

- fid (SE_INTEGER_TYPE)—contains the unique ID that joins the feature table to the business table
- entity (SE_INTEGER_TYPE)—the type of geometric feature stored in the shape column (for example, point, linestring)
- numofpts (SE_INTEGER_TYPE)—the number of points defining the shape
- eminx, eminy, emaxx, emaxy (SE_FLOAT_TYPE)—the envelope of the shape
- eminz (SE_FLOAT_TYPE)—the minimum z-value in the shape
- emaxz (SE_FLOAT_TYPE)—the maximum z-value in the shape
- min_measure (SE_FLOAT_TYPE)—the minimum measure value in the shape
- max_measure (SE_FLOAT_TYPE)—the maximum measure value in the shape
- area (SE_FLOAT_TYPE)—the area of the shape
- len (SE_FLOAT_TYPE)—the length or perimeter of the shape
- points (SE_SHAPE_TYPE)—contains the byte stream of point coordinates that define the shape's geometry
- anno_text (SE_STRING_TYPE)—contains the feature annotation string

NAME	DATA TYPE	Allow Null?
sp_fid	INTEGER(4)	NOT NULL
Gx	INTEGER(4)	NOT NULL

NAME	DATA TYPE	Allow Null?
Gy	INTEGER(4)	NOT NULL
Eminx	INTEGER(4)	NOT NULL
Eminy	INTEGER(4)	NOT NULL
Emaxx	INTEGER(4)	NOT NULL
Emaxy	INTEGER(4)	NOT NULL

Spatial index table schema

The spatial index table defines the grid range and extent of all geometry in an ArcSDE feature class.

- sp_fid (SE_INTEGER_TYPE)—contains the unique ID that joins the feature table to the business table
- gx/gy (SE_INTEGER_TYPE)—defines the feature's extent in grid cells
- eminx/eminy/emaxx/emaxy (SE_INTEGER_TYPE)—defines the extent of the feature in system units

In this example the FEATURE-ID column from the WELLS business table references features from the feature and spatial index tables:

WELL_ID	DEPTH	ACTIVE	FEATURE-ID
1	30029	Yes	101
2	13939	No	102
3	92891	No	103
…	…		…

FID	AREA	LEN	EMINX,EMINY,…	POINTS
101				<compressed feature>
102				<compressed feature>
103				<compressed feature>
…				…

SP_FID	GX	GY	{EMINX,EMINY,EMAXX,EMAXY}
101	70	100	
102	70	100	
103	71	100	
…			

A business/feature/spatial index key reference

The spatial grid index

The spatial grid index is a two-dimensional index that spans a feature class, such as the reference grid you might find on a common road map. You may assign the spatial grid index one, two, or three grid levels, each with its own distinct cell size. The mandatory first grid level has the smallest cell size. The optional second and third grid cell levels are disabled by setting them to 0. If enabled, the second grid cell size must be three times larger than the first grid cell size, and the third grid cell size must be three times larger than the second grid cell size.

The spatial index table (S<feature class_id>) has seven integer columns that store the grid cell values, feature envelopes, and corresponding feature IDs. Adding a feature to a feature class adds one or more grid cells to the spatial index table. The number of records added to the spatial index table depends on the number of grid cells the feature spans.

The spatial index table contains two indexes. One index is on the SP_FID column, which contains the feature ID. The other is a clustered composite index that includes all of the columns of the spatial index table. Since all of the columns of the spatial index table are indexed, the values of the table are read from the leaf blocks of the index and not the table data blocks. The result is less I/O and better performance. In addition, the spatial index table is not accessed whenever the feature class is queried. Therefore, when considering how to position the tables and indexes to reduce disk I/O contention, you should be concerned about the positioning of the indexes of the spatial index table but not the table itself.

Building the spatial index

Every time a feature class is added to a business table, a persistent spatial index is built for it.

The ArcSDE server manages the spatial index throughout the life of the feature class. As features are inserted, updated, or deleted, the spatial index is automatically updated.

A load-only mode disables spatial index management until loading is completed. This boosts loading performance substantially and is imperative for bulk-loading efforts (no queries are allowed in the load-only mode except native SQL-based queries).

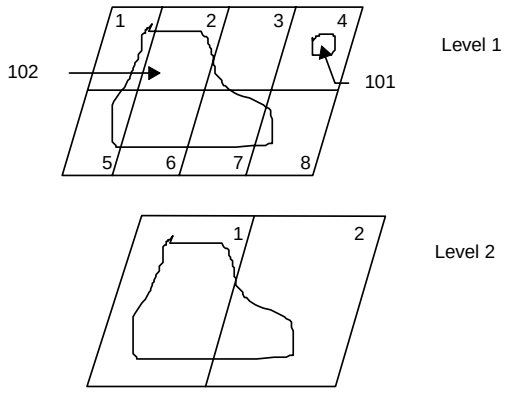
Once loading has been completed, the spatial index is enabled by returning it to normal I/O mode. The conversion from normal I/O mode to load-only I/O mode reconstructs the spatial index.

Inserting, updating, or deleting a feature updates the spatial index when the feature class is in normal I/O mode.

ArcSDE overlays the extent of each feature onto the lowest grid level to obtain the number of grid cells. If the feature exceeds four cells, ArcSDE promotes the feature to the next highest grid level, if you have defined one. ArcSDE will continue to promote the feature until it fits within four cells or less or until the highest defined grid level is reached. On the highest defined grid level, geometries can be indexed by more than four grid cells.

ArcSDE adds the feature's grid cells to the spatial index table with their corresponding shape ID and feature envelope. The grid level is encoded with each grid cell.

In the following example, the feature class has two grid levels. Area shape 101 is located in grid Cell 4 on Level 1. A record is added to the spatial index table because the feature resides within four grid cells (in this case it is one). Area feature 102's envelope is located in Cells 1 through 8 on Level 1. Because the feature's envelope resides in more than four grid cells, the feature is promoted to Level 2, where its envelope fits within two grid cells. Feature 102 is indexed at Level 2, and two records are added to the spatial index table.



Shape 101 is indexed on grid Level 1, while shape 102 is indexed on grid Level 2 where it is in only two grid cells.

Spatial queries and the spatial index

Spatial queries, such as finding all the lakes within a state boundary, use the spatial index. A spatial index is used unless the search order has been set to SE_ATTRIBUTE_FIRST in the SE_stream_set_spatial_constraints function. When the search order is set to SE_ATTRIBUTE_FIRST, ArcSDE ignores the spatial index, and the criteria of the attribute where clause determines which records of the feature class the query returns.

Whenever the spatial index is used, the ArcSDE service generally uses the following decision process when performing the query:

1. Define the envelope. The envelope could be defined directly by the application, such as the extent defined by the ArcMap zoom in tool. Alternatively, the envelope may be defined as the envelope of another feature.
2. Join the spatial index table with the feature table and return all features whose grid cells intersect the envelope.
3. Join the feature table with the business table and apply the criteria of the attribute where clause to further refine features returned.

Grid cell size impacts the size of the spatial index table. Setting up the spatial index means balancing the cell sizes—smaller cell sizes mean more cells per shape, which requires more entries in the spatial index table.

Guidelines for tuning the spatial index

Because client applications and spatial data profiles vary from one system to another, no single solution fits all. Experienced users of ArcSDE often experiment with the spatial index, trying different cell sizes and different grid level configurations.

The sdelayer command has several operations that can help you optimize the spatial index by changing the grid cell sizes and adding new grid levels with the alter operation. The stats and si_stats operations profile your spatial data and current spatial index.

The following guidelines can help improve the performance of spatial queries.

- Consider how many grid levels are needed and remember that the ArcSDE server scans the spatial index table once per grid level. Often a single grid level is the best solution for a feature class, despite the notion of distributing geometries evenly across many grid levels to minimize the spatial index entries.
- Use one grid level for pure point type feature class and consider making the cell sizes large. Spatial queries generally process point geometries faster than other geometry types.
- Monitor the spatial index. Tuning a spatial index is difficult if the data changes frequently. Tuning depends on the structure of the spatial data. Periodically assess the spatial index as your spatial data changes.
- Base the spatial index on the application. Match the spatial index grid cell sizes to the extent of the application window. By doing so, the application is probably viewing exact entries in the spatial index table. This helps to size the spatial index table suitably and reduces the amount of processing, because fewer candidate feature IDs must be evaluated against the feature table (see ‘Spatial queries and the spatial index’ above).
- For unknown or variable application Windows, start by defining one grid level with a cell size three times the average feature extent size. Query the feature table to obtain the average feature size with the following SQL statement:

  ```
  select (avg(emaxx - eminx) + avg(emaxy - eminy)) / 3
  from f<N>;

  (where <N> is the layer number of the feature class)
  ```

  Such spatial index configuration minimizes the number of rows in a spatial index table while maintaining the proficiency of the index because the majority of features can be referenced by less than one or two grid cells.
- Design the feature class around spatial data categories such as type, geometry size, and distribution. Sometimes a carefully designed feature class, using these categories, can substantially boost the performance of spatial queries.

Displaying spatial index statistics

The sdelayer command's spatial index statistics operation, si_stats, can help you determine optimum spatial index grid sizes. Optimum grid cell sizes depend on the spatial extent of all feature geometries, the variation in feature geometry spatial extent, and the types of searches to be performed on the map feature class. Below is a sample output generated by si_stats:

```
$ sdelayer -o si_stats -l victoria,parcels -u av -p mo -i sde81
ArcSDE   8.1          Wed Jan 17 22:43:09 PST 2000
Layer    Administration Utility
-------------------------------------------------------
Layer 1 Spatial Index Statistics:
Level 1,   Grid Size 200
|-------------------------------------------------------|
| Grid Records: 978341                                  |
| Feature Records: 627392                               |
| Grids/Feature Ratio: 1.56                             |
| Avg. Features per Grid: 18.26                         |
| Max. Features per Grid: 166                           |
| % of Features Wholly Inside 1 Grid: 59.71             |
|-------------------------------------------------------|
|             Spatial Index Record Count By Group       |
| Grids: <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes: 627392   0    0    0    0     0     0     0 |
| % Total: 100%  0%   0%   0%   0%    0%    0%    0%|
|-------------------------------------------------------|
Level 2,   Grid Size 1600 (Meters)
|-------------------------------------------------------|
| Grid Records: 70532                                   |
| Feature Records: 36434                                |
| Grids/Feature Ratio: 1.94                             |
| Avg. Features per Grid: 18.21                         |
| Max. Features per Grid: 82                            |
| % of Features Wholly Inside 1 Grid: 45.35             |
|-------------------------------------------------------|
|             Spatial Index Record Count By Group       |
| Grids: <=4    >4  >10  >25  >50  >100  >250  >500 |
|---------- ------ --- ---- ---- ---- ----- ----- ----- |
| Shapes: 35682  752   87   17    3     0     0     0 |
| % Total: 97%   2%   0%   0%   0%    0%    0%    0%|
|-------------------------------------------------------|
```

As the output shows, for each defined spatial index level the following values and statistics are printed:

- Grid level and cell size.
- Total spatial index records for the current grid level.
- Total geometries stored for the current grid level.
- Ratio of spatial index records per geometry.

- Geometry counts and percentages by group that indicate how geometries are grouped within the spatial index at this grid level. The column headings have the following meaning (where "N" is the number of grid cells):

  <=N Number of geometries and percentage of total geometries that fall within <= N grid cells

  >N Number of geometries and percentage of total geometries that fall within > N grid cells

Notice that the ">" groupings include count values from the next group. For instance, the ">4" group count represents the number of geometries that require more than four grid records as well as more than 10, and so on.

- Average number of geometries per grid.
- Maximum number of geometries per grid. This is the maximum number of geometries indexed into a single grid.
- Percentage of geometries wholly inside one grid. This is the percentage of all geometries wholly contained by one grid record.

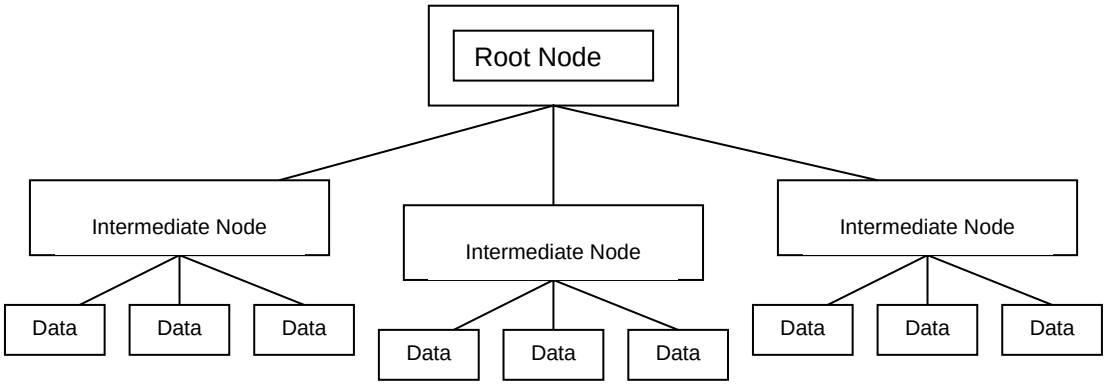
The output sample shows spatial index statistics for a map feature class that uses two grid levels: one that specifies a grid size of 200 meters, the other a grid size of 1,600 meters. When a geometry requires more than four spatial index records, it is automatically promoted to the next grid level if one is defined. That is, in no case will a geometry generate more than four grid records if more than one grid level exists. If a higher grid level doesn't exist, then a geometry can have more than four grid records.

In the example above, 627,392 features are indexed through Grid Level 1. Because the system automatically promotes geometries that need more than four spatial index records to the next defined grid level, all 627,392 geometries for Grid Level 1 are indexed with four grid records or less. Grid Level 2 is the last defined grid level, so geometries indexed at this level are allowed to be indexed with more than four grid records. At Grid Level 2, there are a total of 36,434 geometries and 70,532 spatial index records; 35,682 geometries are indexed with four grid records or fewer, 752 geometries with more than four grid records, 87 geometries with more than 10, 17 geometries with more than 25, and three geometries with more than 50 grid records. Percentage values below each column show how the geometries are dispersed through the eight groups.

Indexes

Every data table in ArcSDE for Microsoft SQL Server has a clustered index. The clustered key in an ArcSDE table is almost always the join key in a query such as the business table's spatial column or the f table's FID column. The SQL Server query processor favors clustered indices because a clustered index orders the table data at its leaf level based on the index key. A normal index would have a pointer to the data page. Clustered indices are b-tree structures containing pages with index rows holding a key value and a pointer to either a data page or a lower-level index page. All searches begin at the top level of the b-tree, the root node. This page is found in the sysindexes table in the

root field. Within this page are pointers to intermediate pages, themselves holding ranges of values and pointers to other pages.



B-tree diagram of a clustered index

Here's an example of the contents of the root node (shortened for brevity's sake). The Page Header displays 0:0 for both m_nextPage and m_prevPage, indicating that it is a root node. Also, two data slots are displayed, each with a key value and a pointer to another page. The pages pointed to are intermediate pages, themselves containing more key values and pointers.

```
PAGE HEADER:
------------
Page @0x19A8A000
----------------
m_pageId = (1:4713)       m_headerVersion = 1       m_type = 2
m_typeFlagBits = 0x0      m_level = 0               m_flagBits = 0x0
m_objId = 1526296497      m_indexId = 1             m_prevPage = (0:0)
m_nextPage = (0:0)        pminlen = 11              m_slotCnt = 133
m_freeCnt = 5968          m_freeData = 1958         m_reservedCnt = 0
m_lsn = (69:1958:6)       m_xactReserved = 0        m_xdesId = (0:0)
m_ghostRecCnt = 0         m_tornBits = 0


DATA:
-----
Slot 0, Offset 0x60
-------------------
Record Type = INDEX_RECORD
Record Attributes =  NULL_BITMAP
19a8a060: 00000116  00126800  02000100      0000 .....h........

Slot 1, Offset 0x6e
-------------------
```

```
Record Type = INDEX_RECORD
Record Attributes =  NULL_BITMAP
19a8a06e: 00001816  00126a00  02000100      0000 .....j........
```

Since a cluster orders the storage of a table, you cannot specify different file groups for the cluster and its table. For example, if you specify in the dbtune table that the feature table's clustered index should go on file group f_idx and the table should reside on f_storage, your table will be stored with the index on f_idx.

What will happen here?

DEFAULTS	F_INDEX_CLUSTER	1
DEFAULTS	F_INDEX_FID	WITH FILLFACTOR=75 ON F_IDX
DEFAULTS	F_STORAGE	ON F_STORAGE

Index maintenance

When you initially create a table with a clustered index and specify a fillfactor for that index, each data page is filled to that fill percentage when you initially insert rows. Later, when you edit this table with insert, delete, and update statements, the amount of data contained in each page will tend to increase.

If a data page fills to 100 percent and that particular page incurs an update or an insert, the page splits approximately in half. Half of the page moves into a newly allocated extent not contiguous to its former location. This will fragment your tables and increase query time as the disk spindles must travel to more locations to fetch data.

You can use the fillfactor clause in the dbtune table to delay page splits by limiting the fill percentage of a data page when it is initially allocated. Any ensuing inserts or updates will add to the fill percentage of a data page. Monitor page splits with DBCC SHOWCONTIG, and when your tables become fragmented rebuild your clustered indexes or run DBCC INDEXDEFRAG (SQL Server 2000 only) or DBCC DBREINDEX.

Rebuilding a clustered index will reorder its table and defragment it. Data pages are reset to their initial fillfactor setting. The simplest way to rebuild an index is to place an ArcSDE feature class into Load_only_io then back to normal_io. However, this won't work on versioned data.

If your data is multiversioned, you don't need to worry about monitoring the ArcSDE business table for extent fragmentation. Instead, your delta tables (a&d), feature tables, and spatial index will become fragmented. You can issue a DBCC DBREINDEX statement or invoke the DBCC INDEXDEFRAG command.

Here's an example of DBCC SHOWCONTIG run against a feature class that has incurred several edits—in this case, many updates and deletes. The output shown below reflects only fragmentation against the spatial index.

```
dbcc showcontig('hydro.s5')

DBCC SHOWCONTIG scanning 's5' table...
Table: 's5' (1622296839); index ID: 1, database ID: 7
TABLE level scan performed.
- Pages Scanned.............................: 57
- Extents Scanned...........................: 11
- Extent Switches...........................: 36
- Avg. Pages per Extent.....................: 5.2
- Scan Density [Best Count:Actual Count].....: 21.62% [8:37]
- Logical Scan Fragmentation ................: 33.33%
- Extent Scan Fragmentation .................: 27.27%
- Avg. Bytes Free per Page...................: 3780.6
- Avg. Page Density (full)...................: 53.29%
```

For an exact explanation of these numbers, see the SQL Server Books Online under DBCC SHOWCONTIG. In this case, 36 extent switches are far too many to cover 57 pages. The best count reflects an optimal amount of extent switches, which in this case would be 8. Both extent scan fragmentation and logical scan fragmentation should be as close to 0 as possible. These measurements reflect the ordering of pages and extents in the index allocation map. Running either of these commands on this table will defragment the table.

```
dbcc indexdefrag (sde,'vtest.s5',s5_pk)
dbcc dbreindex('hydro.s5')

DBCC SHOWCONTIG scanning 's5' table...
Table: 's5' (1622296839); index ID: 1, database ID: 7
TABLE level scan performed.
- Pages Scanned.............................: 41
- Extents Scanned...........................: 6
- Extent Switches...........................: 5
- Avg. Pages per Extent.....................: 6.8
- Scan Density [Best Count:Actual Count].....: 100.00% [6:6]
- Logical Scan Fragmentation ................: 4.88%
- Extent Scan Fragmentation .................: 33.33%
- Avg. Bytes Free per Page...................: 2096.6
- Avg. Page Density (full)...................: 74.10%
```

DBCC DBREINDEX and DBCC INDEXDEFRAG differ in that INDEXDEFRAG can be run online. It will not block queries or updates. For relatively unfragmented tables, DBCC INDEXDEFRAG should run faster than DBREINDEX. For more severely fragmented data, use DBREINDEX.

You only need to worry about business table fragmentation after running a full compress, when the delta tables have been flushed into the business table. Therefore, you should rebuild your business tables' clustered indexes after running a full compress.

Network packet size (SDEPACKETSIZE)

ArcSDE for Microsoft SQL Server stores geometry in an image data type column, points, of the feature table. Several geodatabase network tables use image type columns. Microsoft recommends increasing the size of the network packet size setting when employing image data type columns.

Network packet size is the size of the tabular data scheme (TDS) packets used to communicate between applications and the relational database engine. The default packet size is 4 KB and is controlled by the network packet size configuration option.

ArcSDE, by default, sets this to 8192K, double its default setting of 4096K. You can also make this setting global to your Microsoft SQL Server by using the sp_configure statement's network packet size setting.

```
sp_configure 'show advanced options',1

reconfigure with override

go

sp_configure 'network packet size',8192

reconfigure with override

go
```

You can control the network packet size setting with the environment variable SDEPACKETSIZE, although it is recommended that you stay with the default setting.

Text in row (SQL Server 2000 only)

SQL Server 2000 allows up to 7 KB of image, text, or *n* text data to be stored directly in a row on a data page. SQL Server 7 stored these data types in separate image pages, located with a 16-byte pointer in the table's data page. Text-in-row at SQL Server 20+00 permits you to place image type data directly in the data page, and anything exceeding the setting is placed in a traditional image page. All ArcSDE geometry is stored in an image type column in the feature table. ArcSDE raster layers and geodatabase network tables use image data types.

This setting is enabled on tables using the command sp_tableoption. ArcSDE does this automatically for you through the dbtune table. The SDE_dbtune table sets a default of 256 bytes for many settings.

Keyword	Parameter	Explanation
DEFAULTS	B_TEXT_IN_ROW	Business tables with BLOB data type
DEFAULTS	F_TEXT_IN_ROW	Feature tables points column
NETWORK_DEFAULTS	B_TEXT_IN_ROW	Network business table with BLOB type
NETWORK_DEFAULTS	F_TEXT_IN_ROW	Feature table for generated junctions class
NETWORK_DEFAULTS::NETWORK	B_TEXT_IN_ROW	N_ tables with image type columns

This may not be the best size for all your data. You don't want to allow too much text_in_row space on your data pages if you can't fill it. For example, it would be unwise to set this option to 7000 because you would create many more data pages and extents, thus lengthening the distance a query must travel to retrieve data. Likewise, underestimating this number would create more image data pages and could produce the same problem.

One way to determine an optimal table text_in_row setting is to estimate the eightieth percent size of all your image type records and store up to that size in the *_TEXT_IN_ROW setting. First you'll have to load your data, then run this query:

```
select top 20 percent(datalength(points)) from boris.f12 order by
datalength(points) desc
```

This query will list the top 20 percent largest data records in the "points" column of a feature class's feature table. You then pick the smallest value from its result set and use that for a dbtune entry and reload your data, referencing that keyword. In this case, the eightieth percent value equaled 455, so you would make this entry in the dbtune table:

Keyword	Parameter	Explanation
CARTO	F_INDEX_FID	WITH FILLFACTOR=75 ON FG_CARTO
CARTO	**F_TEXT_IN_ROW**	**455**

Now reload this data referencing this keyword to pick up the new F_TEXT_IN_ROW setting. You could also put this value in the DEFAULTS keyword, but then all data loads would use this size.

```
shp2sde -o create -l silos,shape -f silos -a all -D carto -s bigboy -u
boris -p badinoff -k CARTO
```

You may want to check if a particular table has been enabled for text in row and what its current size is set to. Use the OBJECTPROPERTY Transact-SQL metadata function:

```
select objectproperty(object_id('boris.f12'),'tabletextinrowlimit')
```

You may not want to reload your data. In this case, you can enable a table for text_in_row with sp_tableoption and write a query to copy all the existing image data types into a new column:

```
/*
  FeatureTable_Text_in_Row.sql
  script to copy existing points data into new column
  with new text in row value enabled

  SQL Server 2000 Only

  Script Unsupported by ESRI
*/

sp_tableoption 'boris.f12','text in row',455
go
alter table boris.f12
add newpoints image
```

```
go
update boris.f12
set newpoints=points
go
alter table boris.f12
drop column points
go
sp_rename 'boris.f12.newpoints','points','column'
go
```

If you have a lot of existing data in SQL Server 7.0 and desire to move this data into SQL Server 2000, you'll want to move this data into the new text_in_row format. How you perform your upgrade will determine how your data is moved from a traditional image page into text_in_row.

Text_in_row is enabled with the sp_tableoption command, and ArcSDE sets a text_in_row default (in the sde_dbtune table) for all feature tables to 255 bytes. This means that any layer or feature class creation in ArcSDE on SQL Server 2000 will use the text_in_row option. To upgrade an existing SQL Server 7.0 database to SQL Server 2000 and push your image data into text_in_row format, you can:

- Perform a geodatabase copy and paste between two ArcSDE servers: one running SQL Server 7.0 and the other SQL Server 2000.
- Reload all your data into ArcSDE on SQL Server 2000.
- Run the SQL Server 2000 upgrade wizard or restore your SQL Server 7.0 databases to a SQL Server 2000 instance. Once you have done this, run the `FeatureTable_Text_in_Row.sql` listed above.

Text_in_row for rasters

A raster stored in ArcSDE is composed of four tables. Of those, the sde_blk_<n> table holds the largest image type column. You can enable text_in_row for this table with the dbtune entry BLK_TEXT_IN_ROW. For example:

RASTER	BLK_TEXT_IN_ROW	2000

This will entail changing the raster tile size as well. We recommend that you stay with the defaults as setting a large text_in_row will create too many data pages and extents, eventually slowing your queries.

Triggers

All feature class business tables have a delete update trigger and an insert trigger. The insert trigger ensures that the spatial column cannot have duplicate values in the business table, while the delete–update trigger manages activity against the spatial column in the business, feature, and spatial index tables. You can view these triggers in the SQL Server Enterprise Manager.

These triggers are automatically dropped whenever a feature class, either stand-alone or in a feature dataset, is multiversioned. They are re-created when a feature class is

unregistered as versioned. If you edit one of these triggers, then multiversion the feature class, the trigger will be dropped.

"I" tables and stored procedures

You'll notice that once you've loaded data or run sdesetupmssql, you will have several "i" tables and stored procedures in your databases. These stored procedures and tables are used for fetching feature IDs for feature classes. Editing these tables or stored procedures is not supported and highly discouraged.

APPENDIX B

Storing raster data

A raster is a rectangular array of equally spaced cells, which 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.

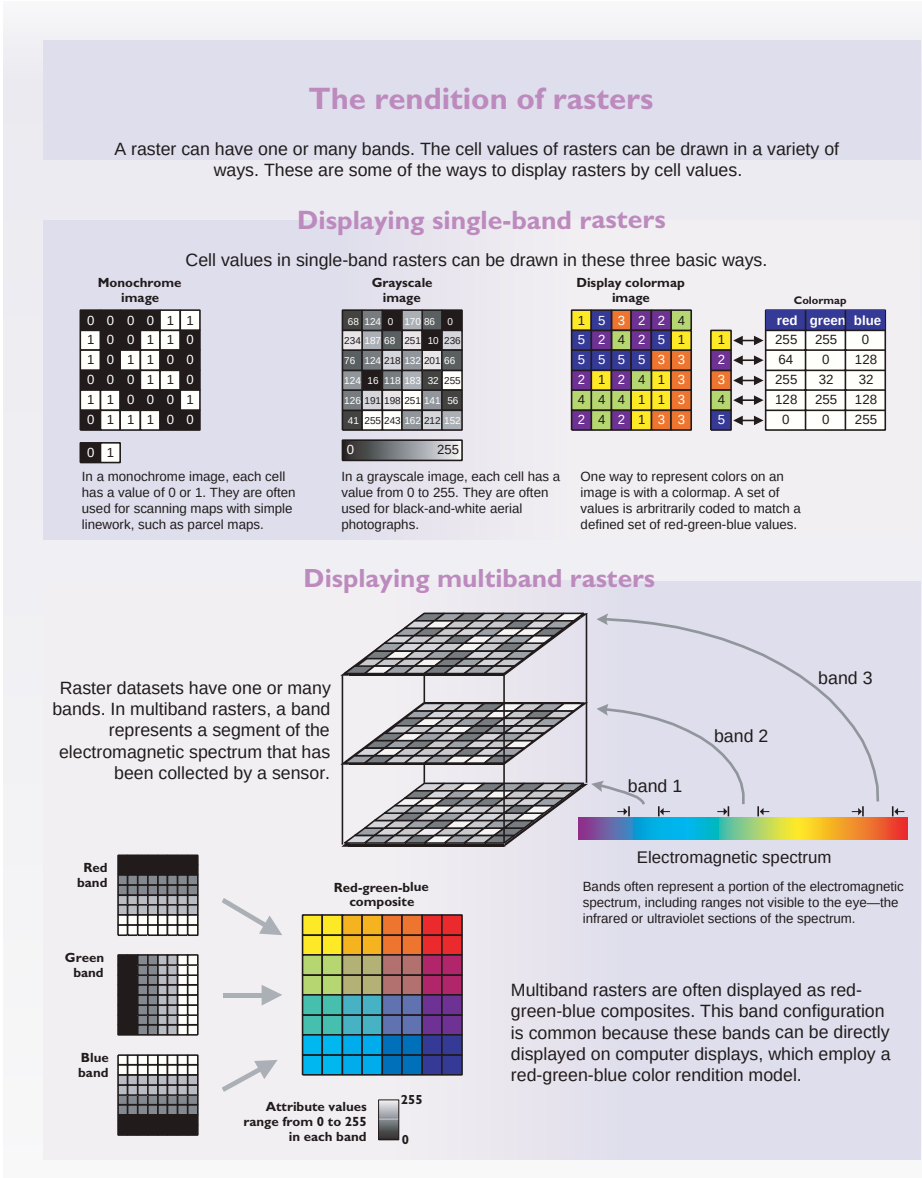
You are probably familiar with raster formats such as tag image file format (TIFF), Joint Photographic Experts Group (JPEG), and Graphics Interchange Format (GIF) that your Internet browser renders. These rasters are composed of one or more bands. Each band is segmented into a grid of square pixels. 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 A, 'ArcSDE compressed binary'). A raster column is added to a business table, and each cell of the raster column contains a reference to a raster stored in a separate raster table. Therefore, each row of a business table references an entire raster.

ArcSDE stores the raster bands in the raster band's table. ArcSDE joins the raster band's 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, or bit masks 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's table. ArcSDE joins the two tables on this primary/foreign key reference when accessing a raster band's metadata.



The raster block's 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. ArcToolbox™ and ArcCatalog, for example, use default raster block dimensions of 128 by 128 pixels per block. The dimensions of the raster block, along with the compression method if one is specified, determine the storage size of each raster block. You should select raster block

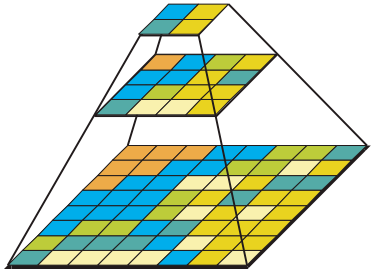
dimensions that, combined with the compression method, allow each row of the raster block table to fit within SQL Server.

The raster block's 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 bands.

ArcSDE populates the raster blocks table according to a declining resolution pyramid. The height of the pyramid is determined by the number of levels, specified by application. The application such as ArcToolbox or ArcCatalog, may allow you to define the levels, may request that ArcSDE calculate them, or may 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 addition of the pyramid levels increases the number of raster blocks by one-third. However, since it is possible for the user to specify the number of levels, the apex of the pyramid may not be obtainable, and therefore, the size increase of the raster blocks table is restricted to the number of resolution levels added.



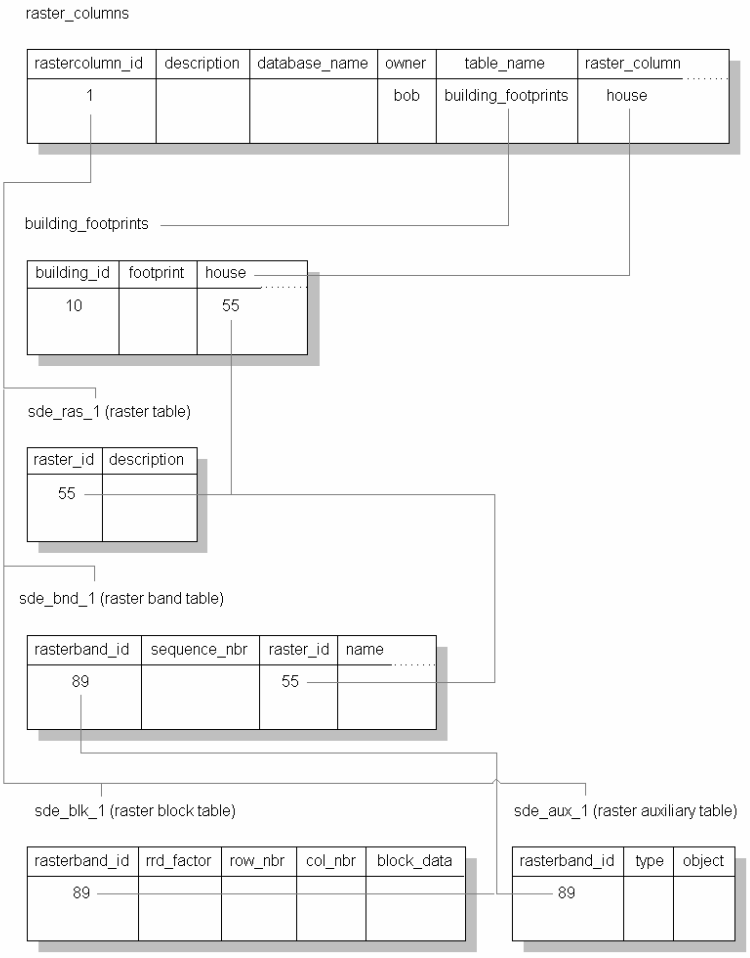
When you build a pyramid more rasters are created by progressively downsampling the previous level by a factor of two until the apex. As the application zooms out and the raster cells grow smaller than the resolution threshold, ArcSDE selects a higher level of the pyramid. The purpose of the pyramid 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 quicker 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 SQL Server'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 bands table by rasterband_id, stores the metadata of each raster band. The rasterband_id also joins the raster band's 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.



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

The sections that follow describe the schema of the tables associated with the storage of raster data. Refer to the previous illustration to see tables and the manner in which they are associated with one another.

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	INT(4)	NOT NULL
description	VARCHAR(65)	NULL
database_name	VARCHAR(32)	NULL
owner	VARCHAR(32)	NOT NULL
table_name	VARCHAR(160)	NOT NULL
raster_column	VARCHAR(32)	NOT NULL
cdate	INT(4)	NOT NULL
config_keyword	VARCHAR(32)	NULL
minimum_id	INT(4)	NULL
base_rastercolumn_id	INT(4)	NOT NULL
rastercolumn_mask	INT(4)	NOT NULL
srid	INT(4)	NULL

Raster columns table

- rastercolumn_id (SE_INTEGER_TYPE)—The table's primary key.
- description (SE_STRING_TYPE)—The description of the raster table.
- database_name (SE_STRING_TYPE)—The name of the database.
- 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 SQL Server 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.
- rid (SE_INTEGER_TYPE)—The spatial reference ID (SRID) is a foreign key reference to the SPATIAL_REFERENCES table. For images that can be geo-referenced, the SRID establishes the X and Y offset translation factor and the scale factor for storage of the image coordinates into the 32-bit integer ArcSDE coordinate storage system. It also stores the coordinate reference system the image was created under.

Business table

In the example that follows, the fictitious BUILD_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	INT(4)	NOT NULL
building_footprint	INT(4)	NOT NULL
house_picture	INT(4)	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 SQL Server, stores a record for each image stored in a raster column. 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	INT(4)	NOT NULL
raster_flags	INT(4)	NULL
description	VARCHAR(65)	NULL

Raster description table schema (SDE_RAS_<raster_column_id)

- raster_id (SE_INTEGER_TYPE)—The primary key of the raster table and unique sequential identifier of each image stored in the raster table
- raster_flags (SE_INTEGER_TYPE)—A bit map set according to the characteristics of a stored image
- description (SE_STRING_TYPE)—A text description of the image (not implemented at ArcSDE 8.1)

Raster band table (SDE_BND_<rastercolumn_id>)

Each image referenced in a raster may be subdivided into one or more raster bands. The raster band table, created as SDE_BND_<rastercolumn_id>, stores the raster bands of each image stored in the raster table. The raster_id column of the raster band table is a foreign key reference to the raster table's raster_id primary key. The rasterband_id column is the raster band table's primary key. Each raster band in the table is uniquely identified by the sequential rasterband_id.

NAME	DATA TYPE	NULL?
rasterband_id	INT(4)	NOT NULL
sequence_nbr	INT(4)	NOT NULL
raster_id	INT(4)	NOT NULL
name	VARCHAR(65)	NULL
band_flags	INT(4)	NOT NULL
band_width	INT(4)	NOT NULL
band_height	INT(4)	NOT NULL
band_types	INT(4)	NOT NULL
block_width	INT(4)	NOT NULL
block_height	INT(4)	NOT NULL
block_origin_x	FLOAT(8)	NOT NULL
block_origin_y	FLOAT(8)	NOT NULL
eminx	FLOAT(8)	NOT NULL
eminy	FLOAT(8)	NOT NULL

NAME	DATA TYPE	NULL?
emaxx	FLOAT(8)	NOT NULL
emaxy	FLOAT(8)	NOT NULL
cdate	INT(4)	NOT NULL
mdate	INT(4)	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, 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 sequence_nbr as a composite key.
- name (SE_STRING_TYPE)—The name of the raster band.
- band_flags (SE_INTEGER_TYPE)—A bit map set according to the characteristics of the raster band.
- band_width (SE_INTEGER_TYPE)—The pixel width of the band.
- band_height (SE_INTEGER_TYPE)—The pixel height of the band.
- band_types (SE_INTEGER_TYPE)—A bit map 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 a SQL Server 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	INT(4)	NOT NULL
rrd_factor	INT(4)	NOT NULL
row_nbr	INT(4)	NOT NULL
col_nbr	INT(4)	NOT NULL
block_data	image	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 bands 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 bit masks 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	INT(4)	NOT NULL
type	INT(4)	NOT NULL
object	image	NOT NULL

Raster auxiliary table schema

- rasterband_id (SE_INTEGER_TYPE)—The foreign key reference to the raster band table's primary key.
- type (SE_INTEGER_TYPE)—A bit map set according to the characteristics of the data stored in the object column.
- object (SE_BLOB_TYPE)—May contain the image color map, image statistics, and so forth.

APPENDIX C

The well-known binary representation

The well-known binary representation for OGC geometry (WKBGeometry) provides a portable representation of a geometry value as a contiguous stream of bytes. It permits geometry values to be exchanged between an ODBC client and an SQL database in binary form.

The well-known binary representation for geometry is obtained by serializing a geometry instance as a sequence of numeric types drawn from the set {Unsigned Integer, Double}, and then serializing each numeric type as a sequence of bytes using one of two well-defined, standard binary representations for numeric types (NDR, XDR). The specific binary encoding used for a geometry byte stream is described by a one-byte tag that precedes the serialized bytes. The only difference between the two encodings of geometry is byte order. The XDR encoding is big-endian, while the NDR encoding is little-endian.

Numeric type definitions

An unsigned integer is a 32-bit (4 byte) data type that encodes a nonnegative integer in the range [0, 4294967295].

A double is a 64-bit (8 byte) double-precision data type that encodes a double-precision number using the IEEE 754 double-precision format.

The above definitions are common to both XDR and NDR.

XDR (big endian) encoding of numeric types

The XDR representation of an unsigned integer is big endian (most significant byte first).

The XDR representation of a double is big endian (most significant bit is first byte).

NDR (little endian) encoding of numeric types

The NDR representation of an unsigned integer is little endian (least significant byte first).

The NDR representation of a double is little endian (sign bit is last byte).

Conversion between the NDR and XDR representations of WKB geometry

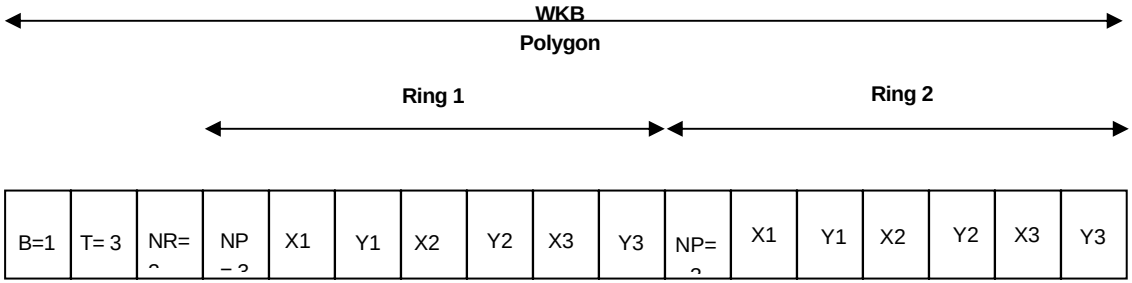
Conversion between the NDR and XDR data types for unsigned integers and doubles is a simple operation involving reversing the order of bytes within each unsigned integer or double in the byte stream.

Description of WKBGeometry byte streams

The well-known binary representation for geometry is described below. The basic building block is the byte stream for a point that consists of two doubles. The byte streams for other geometries are built using the byte streams for geometries that have already been defined.

```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer  (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Point, LinearRing
```

```
Point {
       double x;
       double y;
       };

LinearRing   {
       uint32 numPoints;
       Point  points[numPoints];
       }


enum wkbGeometryType {
       wkbPoint = 1,
       wkbLineString = 2,
       wkbPolygon = 3,
       wkbMultiPoint = 4,
       wkbMultiLineString = 5,
       wkbMultiPolygon = 6,
       wkbGeometryCollection = 7
       };

enum wkbByteOrder {

   wkbXDR = 0,             // Big Endian

   wkbNDR = 1              // Little Endian

};


WKBPoint {
       byte            byteOrder;
       uint32       wkbType;                      // 1
       Point           point;
       }

WKBLineString {
       byte            byteOrder;
       uint32       wkbType;                      // 2
       uint32       numPoints;
       Point           points[numPoints];
       }

WKBPolygon   {
       byte            byteOrder;
       uint32       wkbType;                      // 3
       uint32       numRings;
       LinearRing      rings[numRings];
       }

WKBMultiPoint{
       byte            byteOrder;
       uint32       wkbType;                      // 4
       uint32       num_wkbPoints;
       WKBPoint        WKBPoints[num_wkbPoints];
       }
```




```
WKBMultiLineString  {
       byte            byteOrder;
       uint32       wkbType;                        // 5
       uint32       num_wkbLineStrings;
       WKBLineString WKBLineStrings[num_wkbLineStrings];
       }


wkbMultiPolygon {
       byte            byteOrder;

       uint32       wkbType;                        // 6
       uint32       num_wkbPolygons;
       WKBPolygon      wkbPolygons[num_wkbPolygons];
       }


WKBGeometry  {
       union {
       WKBPoint                  point;
       WKBLineString          linestring;
       WKBPolygon                polygon;
       WKBGeometryCollection  collection;
       WKBMultiPoint          mpoint;
       WKBMultiLineString     mlinestring;
       WKBMultiPolygon        mpolygon;
       }
       };

WKBGeometryCollection {
       byte            byte_order;
       uint32       wkbType;                        // 7
       uint32       num_wkbGeometries;
       WKBGeometry     wkbGeometries[num_wkbGeometries]
       }
```

Well-known binary representation for a geometry object in NDR format (B=1) of type polygon (T=3) with two linears (NR = 2) and each ring having three points (NP = 3).

Assertions for well-known binary representation for geometry

The well-known binary representation for geometry is designed to represent instances of the geometry types described in the geometry object model and in the *OpenGIS Abstract Specification.*

These assertions imply the following for rings, polygons, and multipolygons:

Linear rings—Rings are simple and closed, which means that linear rings may not self-intersect.

Polygons—No two linear rings in the boundary of a polygon may cross each other. The linear rings in the boundary of a polygon may intersect, at most, at a single point but only as a tangent.

Multipolygons—The interiors of two polygons that are elements of a multipolygon may not intersect. The boundaries of any two polygons that are elements of a multipolygon may touch at only a finite number of points.

APPENDIX 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, 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 tables and which transformations (usually the Soundex function) to apply to the address components when searching for records in the geocoding index table. ArcSDE searches for records matching the geocoding index query in the geocoding index table. The resulting set of records from the geocoding index table is joined to the corresponding reference data table to generate a set of candidates for the address. ArcSDE uses the locator's properties to determine which columns in the reference data feature class or table correspond to address components used by the locator, then uses the geocoding rules to assign a score to each candidate.

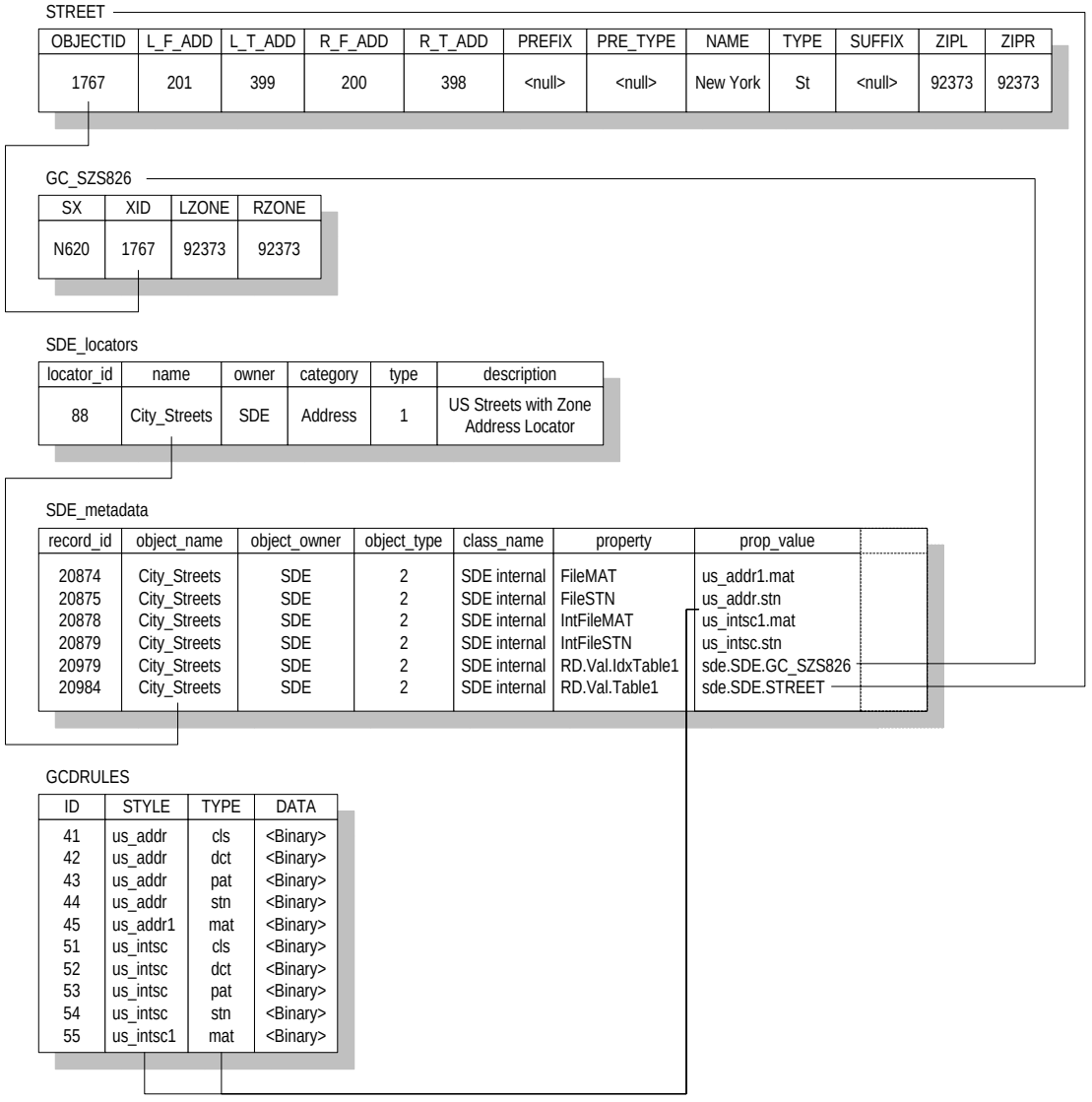
Locator schema

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.



STREET

OBJECTID	L_F_ADD	L_T_ADD	R_F_ADD	R_T_ADD	PREFIX	PRE_TYPE	NAME	TYPE	SUFFIX	ZIPL	ZIPR
1767	201	399	200	398	<null>	<null>	New York	St	<null>	92373	92373

GC_SZS826

SX	XID	LZONE	RZONE
N620	1767	92373	92373

SDE_locators

locator_id	name	owner	category	type	description
88	City_Streets	SDE	Address	1	US Streets with Zone Address Locator

SDE_metadata

record_id	object_name	object_owner	object_type	class_name	property	prop_value
20874	City_Streets	SDE	2	SDE internal	FileMAT	us_addr1.mat
20875	City_Streets	SDE	2	SDE internal	FileSTN	us_addr.stn
20878	City_Streets	SDE	2	SDE internal	IntFileMAT	us_intsc1.mat
20879	City_Streets	SDE	2	SDE internal	IntFileSTN	us_intsc.stn
20979	City_Streets	SDE	2	SDE internal	RD.Val.IdxTable1	sde.SDE.GC_SZS826
20984	City_Streets	SDE	2	SDE internal	RD.Val.Table1	sde.SDE.STREET

GCDRULES

ID	STYLE	TYPE	DATA
41	us_addr	cls	<Binary>
42	us_addr	dct	<Binary>
43	us_addr	pat	<Binary>
44	us_addr	stn	<Binary>
45	us_addr1	mat	<Binary>
51	us_intsc	cls	<Binary>
52	us_intsc	dct	<Binary>
53	us_intsc	pat	<Binary>
54	us_intsc	stn	<Binary>
55	us_intsc1	mat	<Binary>

Business table

In this example, the STREET feature class represents street centerlines within a particular geographic area and contains attributes that allow address locators to geocode addresses using this feature class. 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 (in other words, locators that can be used to find locations).
  - 2—Define attached locators (in other words, 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 stored; 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 two 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 8*i* and 9*i*

These drivers are automatically installed for ArcGIS (the whole product suite), ArcView GIS 3.x Database Access, ArcIMS, ArcInfo workstation and MapObjects 2. If you are using a non-ESRI custom application built from the ArcSDE C API, you may need to install the direct connect drivers from the ArcSDE Developer Kit CD-ROM located in the ArcSDE media kit. Check with the supplier of your non-ESRI custom application.

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 higher 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 higher or ArcGIS 9.0 or higher 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 (such as 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 Application Servers* book 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 dynamically 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 specified path.

You do not have to set the SDEHOME environment variable if the following are true:

- Your users are using ESRI client applications built with the ArcSDE 9 C API (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 used to set environment variables for direct connect use. It may be more convenient to set environment variables for direct connect here than via system tools.

See Chapter 3 in *Managing ArcSDE Application Servers* for more information on the dbinit.sde file.

Client/database compatibility

Direct connect drivers are only compatible with a same-vintage database configured for ArcSDE. For example, you cannot direct connect from ArcMap 9.0 to 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.x. 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.x authorization file.

Setting up clients for SQL Server direct connect

Set up the database

You must set up and configure each database that users will be direct connecting to. Use standard Microsoft SQL Server tools and ArcSDE tools and documentation to

1. install the application server software
2. perform the post installation

When your database is configured for ArcSDE, you are ready to set up your client machines.

Setting up the client machines

On the client machines for which you want to set up a direct connection, you

1 install MDAC

2 set environment variables

3 test the direct connection

If your client machines do not already have MDAC installed, consult Microsoft documentation for MDAC installation instructions.

Set environment variables

Set the SDEHOME and SDE_DATABASE environment variables. Set SDEHOME to point to the directory the client application's dynamically linked library files are stored. Optionally, set the SDE_DATABASE variable to the name of the SQL Server database you will be connecting to.

Connection syntax

There is a particular syntax to use when connecting with direct connect. For the Service (or instance) value,

```
sde:sqlserver:<sqlserver instance name>
```

where

`sde:sqlserver`

is a required part of the syntax and

`<sqlserver instance name>`

is the SQL Server instance you are connecting to. To connect to a specific "named instance", use the <server\instance name> convention. For example, to connect to a named instance called "gis" on a server machine named "geo", enter

`sde:sqlserver:geo\gis.`

Test the connection from the client application

Test the connection.

APPENDIX F

Storing XML Data

This appendix discusses the storage of XML data in an ArcSDE for SQL Server database. An ArcSDE XML column is used to store metadata documents that are published to a Metadata Service. ArcSDE XML columns require that full-text indexing functionality be installed and configured in the databases storing those columns. This appendix will help you understand storing and managing ArcSDE XML data.

Configuring ArcSDE for SQL Server

ArcSDE XML columns require SQL Server's full-text search engine to be installed; the full-text search engine is not supported with Microsoft SQL Server Desktop Engine (MSDE). Once this is accomplished, the database and ArcSDE must be properly configured for the XML column to operate effectively.

Installing SQL Server's text component

The database administrator must accomplish steps 1 through 3 below before an XML column, and therefore an ArcIMS Metadata Service, can be created.

Step 1: Install the full-text search engine. The full-text search engine is installed by default with a typical installation of SQL Server 2000. However, you should be sure the full-text search option is selected when you install the software. If you upgraded SQL Server and full-text search was not installed in SQL Server 7.0, full-text search will not be installed as part of the upgrade; it must be installed as an additional component after the upgrade is completed.

Step 2: Verify the full-text search engine was installed properly. Once SQL Server has been installed, you must verify the full-text search engine was installed properly by running the following query using the Query Analyzer:

```
SELECT fulltextserviceproperty('IsFulltextInstalled')
```

If the value 1 is returned, the full-text search engine has been installed properly. If the value 0 or an error is returned by this query, you must install or reinstall the full-text search engine. You will not be able to successfully create a full-text catalog or an ArcSDE XML column if this query does not return the value 1.

Step 3: Create a full-text catalog in the database. Once the full-text search engine has been correctly installed, you can use either the Enterprise Manager or the Query Analyzer to create a full-text catalog in the database in which the Metadata Service's objects will be created.

The steps for creating a full-text catalog using Enterprise Manager are provided below. If you prefer to use Query Analyzer, see the sp_fulltext_database and sp_fulltext_catalog topics in SQL Server Books Online.

1. In the tree, click the plus sign (+) next to Microsoft SQL Servers, expand SQL Server Groups, expand the appropriate SQL Server instance, then expand the Databases folder.
2. Right-click the database in which the Metadata Service's tables will be created.
3. Click New and click New Full-Text Catalog.
4. For Name, type **SDE_DEFAULT_CAT**. You can provide a different name for the full-text catalog, but you must then modify the value of the XML_IDX_FULLTEXT_CAT keyword in the SDE_dbtune table. This is discussed in more detail below.
5. By default, the full-text catalog is stored in the FTDATA folder of your SQL Server install location. To change this, click the ellipsis (...), and navigate to a new location, then click OK.
6. Click OK in the New Full-Text Catalog Properties dialog box. Once a full-text catalog has been added to the database, review the ArcSDE XML SDE_dbtune parameters before creating an XML column or an ArcIMS Metadata Service.

Setting ArcSDE XML SDE_dbtune parameters

The XML parameters set in the SDE_dbtune table affect how documents published to an ArcSDE XML column are indexed by the database. Information about the following XML parameters should be reviewed before creating an XML column: XML_IDX_FULLTEXT_CAT and XML_IDX_FULLTEXT_UPDATE_METHOD.

If your XML column will store documents written in a language other than English, you will also want to review information about the XML_IDX_FULLTEXT_LANGUAGE parameter in addition to verifying the language settings in the database. If you need to change the default values for these SDE_dbtune parameters and your XML column is associated with an ArcIMS Metadata Service, the changes need to be made for the DEFAULTS keyword. This is discussed in more detail later in this Appendix.

For information on how to change or update these or any other SDE_dbtune keywords, refer to Editing the SDE_dbtune table in Chapter 3 "Configuring Dbtune Storage Parameters."

XML_IDX_FULLTEXT_CAT

The value of the XML_IDX_FULLTEXT_CAT parameter must match the name of the full-text catalog you created in the database. If the full-text catalog's name is different, the SDE_dbtune value must be updated to match it.

Multiple spatial database model: A multiple spatial database instance of ArcSDE has one ArcSDE instance, one database named 'sde,' and one or more databases containing spatial data are dependent upon the sde database. If you have a multiple spatial database instance of ArcSDE, all spatial databases that will contain an ArcSDE XML column must have their own full-text catalog. In this configuration, as each spatial database references the central SDE_dbtune settings, each full-text catalog's name must match the value of the SDE_dbtune XML_IDX_FULLTEXT_CAT parameter.

Single spatial database model: If you have a single spatial database instance of ArcSDE (where there is one SQL Server database per ArcSDE instance), the name of the full-text catalog in each database must match the value of the XML_IDX_FULLTEXT_CAT parameter in that database. Each spatial database may have a different name for its full-text catalog.

XML_IDX_FULLTEXT_UPDATE_METHOD

The frequency and method with which the full-text catalog will be updated in the database needs to be considered based on the amount and rate of change of the documents published to the XML column, the hardware resources available to the database, and whether or not it is important for a newly published document to be immediately available for a search.

Documents can only be found by a search once they have been indexed by the database; that is, once the full-text catalog has been updated. The XML_IDX_FULLTEXT_UPDATE_METHOD parameter in the SDE_dbtune table determines how and when the full-text catalog will be updated.

By default, the XML_IDX_FULLTEXT_UPDATE_METHOD parameter is set to CHANGE_TRACKING BACKGROUND. Change tracking means that when the full-text catalog is updated, only documents added since the last update will be indexed. ESRI recommends that you always use change tracking with your full-text catalog. Background instructs the database to manage when the full-text catalog is updated and to do so using a background process. If this parameter is changed to CHANGE_TRACKING MANUAL, the full-text catalog must be updated manually.

Typically, CHANGE_TRACKING BACKGROUND will provide better overall performance for your database. Background updates will be affected by the resources available on the database server, the priority of the background process, and the load of other operations on the database. Manual updates will have a higher priority in the database and, thus, will probably be faster, but they will affect the performance of

searches and other operations in the database. Updates to the full-text catalog should be scheduled during windows for database maintenance (e.g., nightly); you may want to use the SQL Server database's tools to schedule when the full-text catalog will be updated.

Review the SQL Server Books Online topics 'Full-Text Indexing Support', 'Maintaining Full-text Indexes', 'Full-text Search Recommendations', and 'Performing Infrequent Tasks' for more information about updating full-text catalogs.

XML_IDX_FULLTEXT_LANGUAGE

If your Metadata Service will store documents written in a language other than English, you may need to set the XML_IDX_FULLTEXT_LANGUAGE parameter in the SDE_dbtune table. The XML_IDX_FULLTEXT_LANGUAGE parameter is used to set the language of the column that is indexed in the full-text catalog. This setting affects the linguistic analysis used to index the text in the published documents—if the published documents are written in Japanese but the language of the column is set to English, the text will not be indexed properly, and you will have trouble finding documents with a search.

If you are using a localized version of SQL Server, the default full-text language option will be set to the language of the server. For example, if you are using a Japanese version of SQL Server, the language of the database, the full-text catalog, and the column to be indexed should all be set to Japanese because the default full-text language option will be set to Japanese. You should not have to change the value of the XML_IDX_FULLTEXT_LANGUAGE parameter in the SDE_dbtune table.

If you are using an international version of SQL Server, the default full-text language option will be set to U.S. English. Even if you specifically set the language of the database and the full-text catalog to Japanese, the language of the column that will be indexed will default to U.S. English. Set the XML_IDX_FULLTEXT_LANGUAGE parameter in the SDE_dbtune table to the appropriate value for the language of the published documents; this value is used to set the language of the column. See the SQL Server Books Online help topic 'sp_fulltext_column' for the appropriate value; for example, for Japanese you would set the value of the XML_IDX_FULLTEXT_LANGUAGE parameter in the SDE_dbtune table to be 0x0411.

When a column is set to a specific language, such as Japanese, you will be able to publish and search documents in both Japanese and English. If the XML column will contain documents in many languages or in unsupported language, the language of the column to be indexed should be set to neutral; you may have trouble searching with some text.

See the SQL Server Books Online help topics 'Default full-text language Option', 'Column-Level Linguistic Analysis', and 'sp_fulltext_column' for more information.

Modifying the ArcSDE MAXBLOBSIZE parameter for ArcIMS Metadata Services

XML documents that contain metadata describing GIS resources may contain an image or a reference to an image that illustrates the contents of the resource. When these XML

documents are published to an ArcIMS Metadata Service, a copy of the image is stored in a BLOB column in the feature class associated with the service.

For metadata XML documents created using ArcCatalog, a small thumbnail image may be embedded within the XML document. There should be no trouble publishing XML documents containing these thumbnail images to a Metadata Service with the default MAXBLOBSIZE setting for ArcSDE.

When a metadata XML document is published to a Metadata Service using ArcCatalog and a thumbnail is not embedded within the document, the document's content is checked to see if there is a reference to an external image file. Specifically, the FGDC metadata element Browse Graphic File Name is checked. If a file path or HTTP address is found that references a JPEG image and ArcCatalog can access the image, the file will be loaded into the Metadata Service as the document's thumbnail. ArcCatalog will only attempt to access the first JPEG image referenced in the metadata.

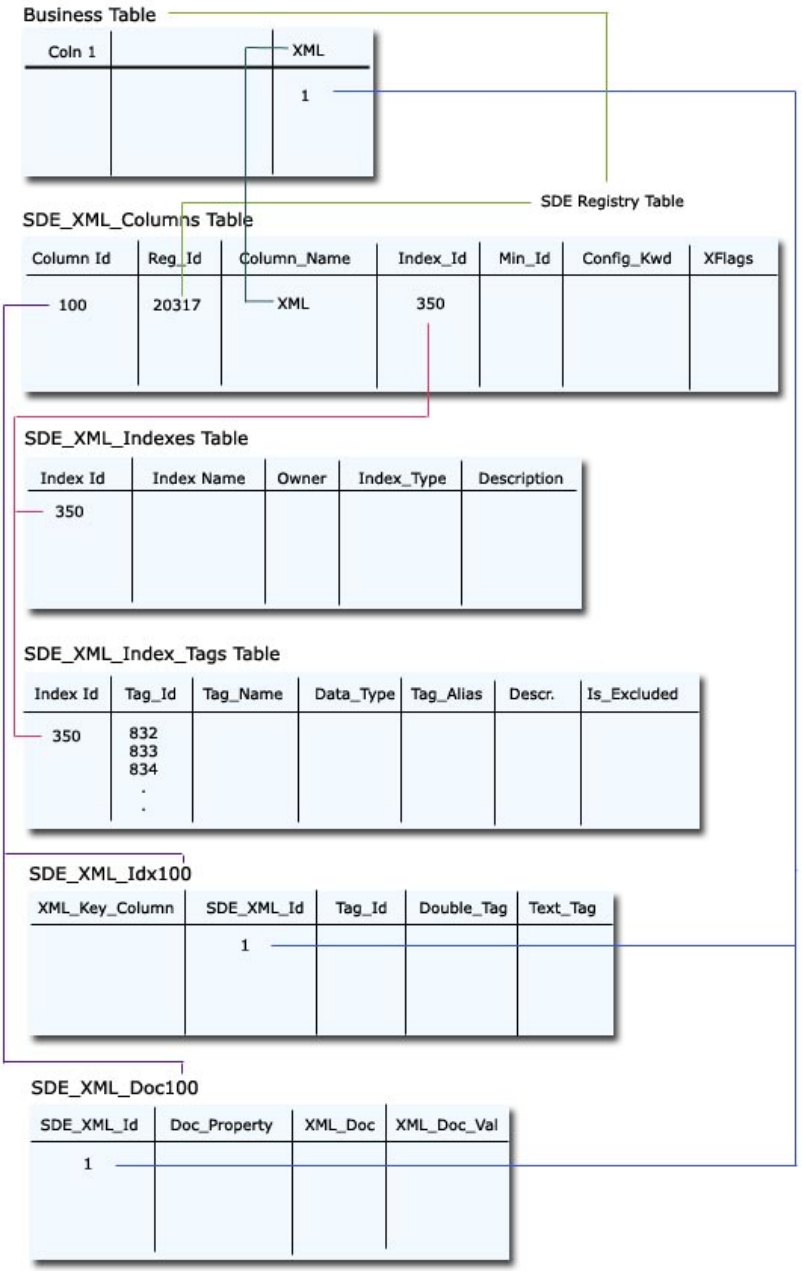
External image files referenced by a metadata document may be large. If the file is larger than approximately one megabyte in size, publishing the document to the Metadata Service may fail. You must adjust the default value of the MAXBLOBSIZE parameter in ArcSDE if you want to support publishing larger images as thumbnails.

The MAXBLOBSIZE parameter is stored in the SDE_server_config table. Use the sdeconfig command to modify SDE_server_config parameter values. The full syntax for this command can be found in the *ArcSDE Administration Command Reference*. For information on setting this parameter appropriately, see the section on managing BLOB data in Chapter 3, 'Configuring ArcSDE Application Servers,' of *Managing ArcSDE Application Servers*. You may also want to refer to Appendix D, 'ArcSDE Initialization Parameters,' in the same guide.

ArcSDE XML columns database schema

Three ArcSDE system tables are used to manage XML columns: SDE_xml_columns, SDE_xml_index_tags, and SDE_xml_indexes; these tables are owned by the SDE user. ArcSDE also creates two additional tables for each XML column that are used to store and index XML documents: the SDE_xml_doc and SDE_xml_idx tables; these tables are owned by the user who owns the business table containing the XML column.

An example of the ArcSDE XML columns database schema is shown on the next page.



When an XML column is added to an object, the column is recorded in an ArcSDE XML columns table. Information in this table is used to create the XML document and index tables and to record information about their text indexes in the ArcSDE XML indexes and index tags tables.

XML columns table (SDE_xml_columns)

When you add an XML column to a business table, a row is added to the XML columns table.

The table's columns are described below. The ArcSDE data type follows the column's name.

Column_id (SE_INT32)—The XML column's identifier. The table's primary key. This value is assigned by ArcSDE at the time the XML column is created.

Column_name (SE_STRING)—The name of the column in the business table that is the XML column.

Config_keyword (SE_STRING)—The SDE_DBTUNE configuration keyword whose parameters determine how the XML document and the XML XPath index tables and text indexes created on those tables are defined in the database. For more information on SDE_dbtune keywords and their parameters, see Chapter 3, "Configuring dbtune storage parameters" in this document.

Index_id (SE_INT32)—The identifier of the XPath index associated with the XML column, if one exists. A foreign key to the XML indexes table.

Minimum_id (SE_INT32)—The value of the initial number used in the business table's XML column to identify individual XML documents.

Registration_id (SE_INT32)—The identifier of the business table containing the XML column. A foreign key to the ArcSDE table registry.

Xflags (SE_INT32)—A value indicating if the original documents in the XML document table are stored compressed or uncompressed. By default, documents are compressed; compressed documents provide better performance.

XML index tags table (SDE_xml_index_tags)

An XML column may optionally have an XPath index, which lets people search the content of a specific XML element or attribute in each document. The definition of which elements and attributes are included in or excluded from each XPath index is recorded in this table.

This table occurs once in each ArcSDE database. It contains one row for each XPath associated with an XML column's XPath index.

Description (SE_STRING)—Text identifying the content that should be contained in the XML element or attribute.

Index_id (SE_INT32)—The identifier of the XPath index associated with an XML column, if one exists. A foreign key to the XML indexes table.

Is_excluded (SE_INT32)—A value indicating if the XML element is included in or excluded from the XPath index. A zero indicates the XPath is included; a one indicates the XPath is excluded.

Tag_alias (SE_INT32)—A number that may be used to identify an XPath. For example, the Z39.50 communication protocol uses numeric codes to refer to content that may be searched. This column is not used by the ArcIMS Z39.50 Connector.

Tag_id (SE_INT32)—The identifier of an XPath or tag.

Tag_name (SE_STRING)—An absolute XPath identifying an XML element or attribute that may occur in an XML document. For example, /metadata/mdDateSt identifies an XML element, and /metadata/dataIdInfo/tpCat/TopicCatCd/@value identifies an XML attribute. These XPaths must not contain asterisks (*) to refer to a group of XML elements or attributes—each elements or attribute is matched exactly using the XPaths specified in this table.

XML indexes table (SDE_xml_indexes)

This table occurs once in each ArcSDE database. It contains one row for each XML column that has an XPath index.

Description (SE_STRING)—Text identifying the XPath index. If an index definition file was used to create the index, the index's description may be specified at the top of the file.

Index_id (SE_INT32)—The identifier of the XPath index. The table's primary key.

Index_name (SE_STRING)—The name of the XPath index. For XPath indexes associated with an ArcIMS Metadata Service, the name will be "ims_xml#" where # is the identifier of the XML column in the Metadata Service's business table.

Index_type (SE_INT32)—A value indicating the type of XPath index. With ArcSDE 9.1 the value will always be 2; only XPath indexes of the type SE_XML_INDEX_DEFINITION are supported.

Owner (SE_STRING)—The database user who owns the XML column. For ArcIMS Metadata Services, this is the user specified in the service's ArcXML configuration file.

XML document table (sde_xml_doc<column_id>)

The ArcSDE database contains one of these tables for each XML column. The number in the table name is the XML column's identifier. This table contains one row for each XML document stored in the XML column.

Doc_property (SE_INT32)—A value indicating if any conflicts were found when adding the content of an XML document to the XPath index. A one indicates a conflict was found, for example, when an element is supposed to be indexed numerically but the document contains a string in that element instead. A NULL value indicates there were no problems indexing the document.

Sde_xml_id (SE_INT32)—The identifier for an XML document stored in the XML column. The primary key for the table.

Xml_doc (SE_BLOB)—The XML document.

Xml_doc_val (SE_BLOB)—The content of the entire XML document, with all XML tags removed. A text index is built on this column by default; this index is used to respond to full-text queries. For ArcIMS Metadata Services, this index is used to respond to FULLTEXT requests.

XML XPath index table (sde_xml_idx<column_id>)

The ArcSDE database contains one of these tables for each XML column that has an XPath index. The number in the table name is the XML column's identifier. This table contains one row for each XML element or attribute in each document that is included in the XPath index.

Double_tag (SE_FLOAT64)—For XPaths that are indexed numerically, the number contained in the element or attribute. For ArcIMS Metadata Services, this column is used to respond to TAGVALUE requests.

Sde_xml_id (SE_INT32)—The XML document's identifier. A foreign key to the XML document table.

Tag_id (SE_INT32)—The identifier for the XML element or attribute that is indexed. This number corresponds to the value in the tag_id column in the XML index tags table.

Text_tag (SE_CLOB)—For XML elements or attributes that are indexed as string, the text contained in the element or attribute. A text index may optionally be built on this column; this index is used to respond to XPath queries. For ArcIMS Metadata Services, this index is used to respond to TAGTEXT requests.

Xml_key_column (SE_INT32)—The primary key for the table.