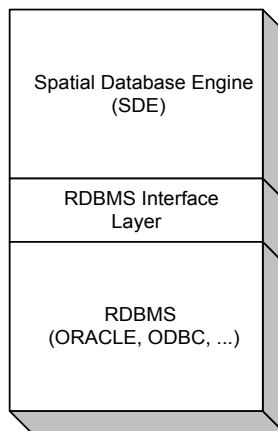


ESRI ArcSDE Reader/Writer

FORMAT NOTES:

- This format is not supported by FME Base Edition.
- This chapter describes FME's ArcSDE 3.x/8.x/9.x support, which is different from SDE 2.1 support. The SDE 2.1 format is deprecated in FME 2009.
- This chapter also applies to ArcGIS 9.x.
- This chapter includes **SDE30, SDERASTER (Reader), SDERASTERMAP (Writer), SDERASTERCATALOG (Writer)**.

ESRI's Spatial Database Engine (SDE) 3.x/ArcSDE 8.x/9.x¹ brings Geographical Information Systems (GIS) into the realm of Management Information Systems (MIS) by providing a spatial interface to industry-standard Relational Database Management Systems (RDBMS).



Spatial Database Engine

SDE enables a Relational Database Management System to store both spatial and non-spatial data by providing a new "Shape" column type to the underlying RDBMS. The Feature Manipulation Engine (FME) is an SDE client application capable of connecting to the SDE regardless of the platform on which it is located.

Overview

The SDE provides a seamless data model into which geographic data is stored. The SDE provides a relational data model organized around tables. In the FME, an SDE feature type² is equivalent to an RDBMS table. The FME can be used to read and write any RDBMS table whether or not it has a layer (vector spatial column) and whether or not it has a raster column. Spatial geometry in a table can be found in either a layer or a raster column.

A layer (vector spatial column) has the following properties:

1. Throughout this chapter, *SDE* refers to clients using SDE 3.x or ArcSDE 8.x or 9.x, unless otherwise stated. The same reader/writer is used to access all of these clients.
2. The terms "feature type" and "table" are used interchangeably throughout this chapter.

- Each layer has a spatial index that can be tuned specifically for it. The spatial index consists of between one and three two-dimensional (2D) grids. The sizes of the grid elements are ordered such that:

grid1 size < 4 X grid2 size (except if grid2 is set to zero)

grid2 size < 4 X grid3 size (except if grid3 is set to zero)

Tip: Since the SDE stores all coordinates as 32-bit (or 64-bit since ArcSDE 9.0) integer coordinates with an implied decimal position, it is possible for precision to be lost or for overflow to occur when features are stored in the SDE. Care must be taken to ensure that the SDE dataset system units preserve the data precision and the range.

- A single relational table can only have 1 layer.
- All features in a layer must be either two- or three-dimensional (2D or 3D). Mixed dimensionality is not allowed in a layer
- Each layer has its own coordinate system, false origin, and scaling factor.
- The layer in a raster catalog is referred to as the 'footprint' column, and stores the bounding box of each raster in the catalog.

A raster column has the following properties:

- A single relational table can only have 1 raster column.
- Each raster column has its own coordinate system. If the coordinate system is not specified, it will be stored as UNKNOWN.
- A raster column can either be a raster map storing all raster data in the table as a single raster in a single row, or as a raster catalog storing multiple rasters in multiple rows in the table.

The FME's SDE reader and writer modules take advantage of the unique capabilities of the SDE. The reader module retrieves features from the SDE by constructing queries consisting of both spatial and/or non-spatial components. An SDE database may have a very large number of features, therefore the query building capability is critical to ensure that the FME reader module is capable of satisfying highly focused data requests. The writer module takes advantage of the SDE's transaction model to ease the task of importing data into the SDE. The SDE writer is also able to operate in either an "Update" mode or an "Insert" mode, enabling the FME to be used as a key component in an SDE-based solution. The SDE reader is also capable of performing multi-table join queries, thereby exploiting the full power of the underlying RDBMS.

Note: Version support, WHERE clauses, spatial constraints, and multi-table joins are not supported for raster data at this time.

Tip: See the `SDE30QueryFactory` in the *FME Functions and Factories* manual. This factory also exploits the powerful query capabilities of the SDE 3.0/ArcSDE 8.x/ArcSDE 9.x.

See the `@SDEsql` function in the *FME Functions and Factories* manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against the SDE's underlying database.

ESRI ArcSDE Quick Facts

Format Type Identifier	SDE30 SDERASTER (Reader) SDERASTERMAP (Writer) SDERASTERCATALOG (Writer)
Reader/Writer	Both
Licensing Level	Professional for SDE30; ESRI for SDERASTER, SDERASTER- MAP, and SDERASTERCATALOG
Dependencies	None
Dataset Type	Database
Feature Type	table name
Typical File Extensions	Not applicable
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	No
Schema Required	Always
Transaction Support	Yes
Geometry Type	Yes
Encoding Support	Yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	yes
donut polygon	yes	solid	no
elliptical arc	no	surface	no
ellipses	no	text	yes
line	yes	z values	yes
none	yes		

Raster-specific Quick Facts

Band Interpretations	Red8, Red16, Green8, Green16, Blue8, Blue16, Gray8, Gray16, Int8, UInt8, Int16, UInt16, Int32, UInt32, Real32, Real64
Palette Key Interpretations	UInt8, UInt16
Palette Value Interpretations	RGB24, RGBA32
Interleave Type	BSQ
Nodata Value	any
Cell Origin	0.5
Multi-Band	Yes
Multi-Palette	No

Notes:

- Only rasters with color palettes can be written with the ArcSDE writer. Other types of palettes must either be converted to color palettes or be removed or resolved to continuous rasters in order to be written.
 - Raster catalogs must be registered with the Geodatabase via ArcCatalog, in order for them to be viewed properly. Unregistered raster catalogs will appear as tables.
-

FME SDE Highlights

The SDE reader and writer modules provide the FME with the ability to store data in and retrieve data from ESRI's SDE.

The SDE modules deliver the following capabilities.

- **Programmatic Table Creation:** Tables need not be created before a data import operation. All table creation details are handled by the FME.
- **Programmatic Index Creation:** Non-spatial column indices can be specified within FME mapping files. These indices are used to enhance the performance of the non-spatial component of searches.
- **Programmatic Spatial Column and Attribute Verification:** When loading data into an existing spatial database, the FME verifies that the table definitions specified in the mapping file match the existing RDBMS definitions.
- **Versioning Support:** FME enables data to be read from a particular version of an SDE database, and also allows data to be written to a specific version of an SDE database.
- **Transaction Support:** Transactions are fully supported enabling a partially complete load operation to be resumed later, without the loss of or duplication of data.
- **Attribute Query Support:** SQL `WHERE` clauses can be specified to limit the data being exported.
- **Multi-Table Query Support:** The queries may combine multiple RDBMS tables thereby exploiting the full SQL capabilities of the underlying RDBMS.
- **Multiple SDE Connections:** The FME is capable of having multiple SDE connections open to the same or different SDE databases. This may be used for selective repli-

cation between different SDE sites, for moving data from a preparation SDE to a production SDE, or for building result sets of features from multiple SDE databases.

- **Spatial Query Support:** FME exploits the spatial query capabilities of the SDE.
- **Non-Spatial Table Support:** Any table can be read or written to an SDE database with FME whether or not it is spatially enabled. For example, if your SDE uses Oracle then FME can be used to read, write, or create regular Oracle tables.
- **Rejected Features' Pipeline:** Features whose geometry is rejected by SDE can be sent through an FME Pipeline and modified so that they can be given a second opportunity to be inserted into SDE.
- **Geodatabase Reading/Writing Support:** Features can be read from or written to an existing feature class located on an Enterprise Geodatabase.
- **Unicode Support:** With SDE 9.2 and later, FME can read from and write to text fields encoded in UTF-16.
- **Fully Automatic Import and Export:** FME's SDE support provides fully automated import and export of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports or quick data exports.

Tip: The query support enables the data export operation to be highly focused, thereby reducing the amount of data that is exported.

- **Raster Support:** Raster reading and writing can be done through FME, and includes support for multiple bands, nodata values, mosaicking, compression and pyramiding as well as storage of data in either raster maps or raster catalogs.

Note: Nodata values in ArcSDE raster tables are stored as locations rather than explicit values. FME maps these nodata locations in the raster to a single, unused value within the data range using statistics calculated on the raster band(s). Therefore, in order to retrieve nodata values from a raster table, statistics must be calculated prior to reading from it. This can be done either through the FME ArcSDE raster writer by setting the statistics option to 'AUTO', or by manually calculating statistics on the raster after it has been written using ArcGIS applications or command line tools.

The SDE modules within FME are designed to work with other SDE products. For example, a user with a simple GUI search engine can easily identify all features satisfying a complicated query, then use FME with the SDE reader module to process these features.

Connecting to SDE

Connecting to SDE is the same for the reader and the writer. They both use the same directives in the same manner. There are two possible ways to connect to SDE:

- connecting to the ArcSDE service, which in turn communicates with the underlying database (a three-tier architecture), and
- connecting to the underlying database directly (a two-tier architecture).

With a direct connection (the two-tier architecture), ArcSDE does not need to be installed and an ArcSdeServer license is only needed if writing to the database; reading does not require a license.

Regular Connection

The connection parameters needed for a regular connection are as follows:

DATASET

Required/Optional: *Required*

This statement identifies the SDE database from which features are retrieved/written. In SDE, this dataset is referred to as the `DATABASE`. This is required no matter what the underlying RDBMS of the SDE. Some RDBMSes, such as Oracle, do not require a value, whereas others, such as SQLServer, do. For databases that do not require the value, the value `not_used` is specified by convention.

Example:

```
SDE30_DATASET testdset
```

[Workbench Parameter: <WorkbenchParameter>](#)

SERVER

Required/Optional: *Required*

This statement identifies the SDE server used to read data from the dataset.

Example:

```
SDE30_SERVER tuvok
```

[Workbench Parameter: <WorkbenchParameter>](#)

INSTANCE

Required/Optional: *Required*

The instance to which the FME is to connect. The usual value for systems with a single SDE instance is `esri_sde`. The instance can also be of the form `port:<port-number>`.

Example:

```
SDE30_INSTANCE esri_sde
```

[Workbench Parameter: <WorkbenchParameter>](#)

USERID

Required/Optional: *Optional if connecting in OSA mode*

The user name required to access the SDE database.

If the userid and/or password are missing or not set, then the reader will try and connect with Operating System Authentication.

Example:

```
SDE30_USERID ronny
```

[Workbench Parameter: <WorkbenchParameter>](#)

PASSWORD**Required/Optional:** *Optional if connecting in OSA mode*

The password associated with the specified user ID.

If the userid and/or password are missing or not set, then the reader will try and connect with Operating System Authentication.

Example:SDE30_PASSWORD ronpassword [Workbench Parameter: <WorkbenchParameter>](#)**VERSION_NAME****Required/Optional:** *Optional*

The SDE version to which FME connects. The version must already exist and the current user must have privileges set so that it can access the version. If the `VERSION_NAME` directive is not used, then the FME attempts to connect to `SDE.DEFAULT`. If there is no SDE schema, FME then attempts to connect to `dbo.DEFAULT`. If the name is not prefixed by the owner of the version, then it is assumed that the owner is the current user. This directive is only applicable when dealing with versioned tables. This directive is not available when reading/writing raster data, since versioning is not currently supported.

Default: *SDE.DEFAULT***Example:**

SDE30_VERSION_NAME ron.working-version

[Workbench Parameter: <WorkbenchParameter>](#)**Direct Connection**

The parameters needed to make a direct connection to SDE depend on the underlying database. In order to make a direct connection, the SDE must be of the same major & minor version as the client libraries with which the ArcSDE reader/writer was built.

For example, a direct connection to an ArcSDE 9.1 instance could only be made with a reader or writer built using 9.1 libraries.

Underlying Database	Mandatory Directive	Value
Oracle (option 1)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	INSTANCE	sde:oracle or sde:oracle9i (for 9i connections to use the right driver)
	USERID	<username>

Underlying Database	Mandatory Directive	Value
	PASSWORD	<password>@<Oracle Net Service Name>
Oracle (option 2)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	INSTANCE	sde:oracle://;local=<sqlnetalias>
	USERID	<username>
	PASSWORD	<password>
MS SQL Server	DATASET	<database_name>
	INSTANCE	sde:sqlserver:<SQL Server Instance Name> or sde:sqlserver:<SQL Server Instance Name>\<Named Instance> (for connecting to a named instance)
	USERID	<username>
	PASSWORD	<password>
DB2 (option 1)	DATASET	<db alias name specified through DB2 Configuration Assistant>
	SERVER	remote (if client application is remote, otherwise do not specify)
	INSTANCE	sde:db2
	USERID	<username>
	PASSWORD	<password>
DB2 (option 2)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	SERVER	remote (if client application is remote, otherwise do not specify)
	INSTANCE	sde:db2:<db alias name specified through DB2 Configuration Assistant>
	USERID	<username>
	PASSWORD	<password>
Informix	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	SERVER	remote (if client application is remote, otherwise do not specify)
	INSTANCE	sde:informix:<odbc data source name>

Underlying Database	Mandatory Directive	Value
	USERID	<username>
	PASSWORD	<password>

The directive `VERSION_NAME` can also be used to specify the version when making a direct connection.

Please refer to http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?Topic-Name=Properties_of_a_direct_connection_to_an_ArcSDE_geodatabase for more information on setting up the direct connect environment and tips on proper usage.

Reader Overview

The SDE reader begins by starting an SDE session with the server upon which the SDE dataset resides. Once connected, the SDE reader queries the SDE and passes the resulting features – that is, rows – on to the FME for processing.

When reading features from the SDE, the tables from which features are retrieved are specified in the mapping file using the `<ReaderKeyword>_IDs`. An optional spatial component and `WHERE` clause may also be specified. If no spatial or attribute constraints are specified, then all features from the identified table(s) are read.

The SDE reader requires that a `<ReaderKeyword>_IDs` statement be specified, identifying the tables from which data is to be retrieved. If no identifiers (IDs) are specified, then no features are read from the database.

The table below summarizes the different feature retrieval modes supported by the SDE reader module for a Spatial Retrieval search type. The next section contains an in-depth discussion of each of these keywords.

Search Keyword Suffix	Description
IDs	Specifies the tables from which features are to be retrieved. If no tables are specified, then no features are retrieved. Each ID specified may be simple or compound . A simple ID is an ID that specifies only one table. A compound ID is one in which several tables are specified – this is not supported when reading raster data. When a compound ID is specified, features are constructed by joining several tables together during the query. A single IDs statement consists of one or more IDs, each of which may be simple or compound .
SEARCH_ENVELOPE	Specifies the spatial extent of the feature retrieval. Only features that have the relationship specified by <code>SEARCH_METHOD</code> with the envelope are returned. The only valid value for <code>SEARCH_METHOD</code> for raster features is <code>SDE_ENVELOPE</code> .
SEARCH_FEATURE	Specifies a feature with an arbitrary number of coordinates as the search feature. Only features that have the relationship specified by <code>SEARCH_METHOD</code> with the search feature are returned. Currently only valid for vector features.

Search Keyword Suffix	Description
SEARCH_METHOD_FILTER	<p>Specifies if the features returned will or will not satisfy the spatial constraint. If <code>FALSE</code> is specified, then all features returned will not satisfy the spatial constraint. If <code>TRUE</code> is specified, then the features returned will satisfy the spatial constraint. Specifying <code>FALSE</code> enables you to select all features which are not contained by a feature, for example.</p> <p>Value: <code>FALSE</code> <code>TRUE</code> Default: <code>TRUE</code></p>
WHERE	<p>Specifies the attribute constraint that a feature must have to be retrieved. Any valid SQL <code>WHERE</code> clause may be entered here as this value is passed directly to the underlying RDBMS for processing.</p> <p>When compound IDs are used, the <code>WHERE</code> clause specifies how the tables are joined during the query.</p> <p>Currently only valid for vector features.</p>

Note: Search feature, multi-table joins, and `WHERE` clause constraints are not currently supported for raster tables.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for reading vector data is `SDE30`, the default for reading raster data is `SDERASTER`.

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

Specifies whether to retrieve schemas for all the tables in the database. This directive is only applicable when generating a workspace/mapping file or when calling `IFMEUniversalReader::readSchema()` through FME Objects. The type of tables returned depends on whether the `SDERASTER` or `SDE30` reader is being used. When using the raster reader then the raster tables are returned and when using the vector reader both vector and non-spatial tables are returned.

Value: `YES` | `NO`

Default Value: `NO`

Workbench Parameter: [<WorkbenchParameter>](#)

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

Specifies whether to retrieve all the table names in the database. This directive can only be used within FME Objects applications through `IFMEUniversalReader::readSchema()`. Unlike the `RETRIEVE_ALL_SCHEMAS` directive, the schema features contain only the feature type which represents the table name from the database. When using

the raster reader then the raster tables are returned and when using the vector reader both vector and non-spatial tables are returned.

Value: YES | NO

Default Value: NO

Workbench Parameter: [<WorkbenchParameter>](#)

PERSISTENT_CONNECTION

Required/Optional: *Optional*

Specifies whether to create a connection to SDE that persists and can be shared by other SDE Readers, Writers, and SDE30QueryFactories. When set to YES, the connection will remain open until FME shuts down, even if this reader is finished using it. Otherwise, the connection will be closed when the reader is shut down (unless another reader/writer/queryfactory is still using the connection).

Creating a new connection is an expensive operation. Depending on how FME is being used (that is, if there are multiple instances of the SDE Reader/Writer being used, or if the SDE30QueryFactory is being used to query/update the same SDE), the performance may improve by setting this directive to YES.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_PERSISTENT_CONNECTION YES
```

Workbench Parameter: [<WorkbenchParameter>](#)

WHERE

Required/Optional: *Optional*

Note: Currently only valid for vector features.

The specified WHERE clause is passed to the SDE for processing. WHERE clauses may be arbitrarily complex, limited only by the underlying RDBMS. The WHERE clause may specify join operations; for example, when a retrieval operation is from multiple tables.

When a join operation is being performed, the FME features have all of the attributes from the various tables combined into each feature. If more than one table has a column with the same name, then the feature attributes for those columns will be qualified with the table name to ensure that each attribute is unique. If a query operation is performed across multiple tables, the spatial component will only be taken from the primary table. In fact, none of the secondary tables should have a spatial column (layer or raster column). If the primary table does not have any spatial component, then the feature will not have any spatial component. See *IDs* on page 616 for a description of how compound IDs are specified.

If a table contains a layer (vector spatial column), then it is also possible to select one particular feature using the following WHERE clause syntax:

```
SE_ROW_ID = <integer>
```

where <integer> is some integer value and SE_ROW_ID is a virtual column provided by the SDE. Only the '=' operator can be used when the query involves SE_ROW_ID. A query involving SE_ROW_ID cannot be arbitrarily complex. It can ONLY be of the form SE_ROW_ID = <integer>.

In the event of an error, the FME will log the entire <WHERE clause>. This clause can then be debugged in the native RDBMS.

Value: <WHERE clause>

The attribute constraint used to limit the features which are retrieved based on attribution. When the WHERE clause references multiple tables, these tables must all be specified as a compound SDE30_ID value. See *IDs* on page 616 for a complete discussion.

Example 1:

The WHERE clause specified below instructs the FME to retrieve features from the database for the table(s) identified by <ReaderKeyword>_IDs, and for those rows in which the NUMLANES column has a value of 2 and the SURFACE column has a value of 'GRAVEL'. Notice the use of the double quotes around the compound WHERE clause.

```
SDE30_WHERE "NUMLANES=2 AND SURFACE='GRAVEL'"
```

Example 2:

The WHERE clause below illustrates how to perform a join on 3 different tables named REPLICATION, DIST_CENTER, and REPLICATION_LAYERS. Each of these tables must be specified in the associated SDE30_IDS clause for this example to work. Again, notice the use of the double quotes around the compound WHERE clause. Also note the use of the continuation characters when building a long WHERE clause.

```
SDE30_WHERE                                     \
"REPLICATION.REPLICATION_DATE is null AND"      \
"REPLICATION.AREA_CD = DIST_CENTER.AREA_CD AND" \
"REPLICATION.IDENT = REPLICATION_LAYERS.IDENT AND" \
"DIST_CENTER.TOWN_CD = '$(TOWN)'"
```

Workbench Parameter: <WorkbenchParameter>

REMOVE_TABLE_QUALIFIER

Required/Optional: *Optional*

Specifies whether to keep or remove the table name prefix. If the ArcSDE resides on a database (that is, MS SQL Server) where a specific value for database is set, then the full name for a table is <database_name>.<owner_name>.<table_name>. If the ArcSDE is located on a database (that is, Oracle) that does not require the database field, then the full name of a table is <owner_name>.<table_name>. Setting this keyword to YES indicates that the reader should return the table name without any prefixes. This is useful when:

- creating a workspace that will be passed on to another organization using the same table names,

- performing a translation to another database format but with a different user name, and
- when writing to a file-based format but not wanting the prefix in the name of the feature type.

When this keyword is set to `YES` during the generation of a mapping file or workspace, the source feature types will be the table names without any prefix; otherwise, they will contain the owner name as a prefix. It is recommended that this keyword not be changed in value after generating the mapping file/workspace as it is possible for no features to be successfully passed onto the writer (since the writer is expecting feature types with different names).

Note that even when `REMOVE_TABLE_QUALIFIER` is set to `YES`, if the table is owned by a user other than the current user, the `<owner_name>` prefix will **not** be dropped so that the reader will find the correct table; however, the `<database_name>` prefix will still be dropped.

Value: `YES` | `NO`

Default Value: `NO`

Example:

```
SDE30_REMOVE_TABLE_QUALIFIER YES
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE

Required/Optional: *Optional*

The `SEARCH_ENVELOPE` clause provides a mechanism for specifying a rectangular spatial constraint. The `SEARCH_ENVELOPE` clause works with the `SEARCH_METHOD` clause to define the spatial constraint.

Parameter	Contents
<code><min-x></code>	The minimum x coordinate in the coordinate system of the spatial column(s) being retrieved.
<code><min-y></code>	The minimum y coordinate in the coordinate system of the spatial column(s) being retrieved.
<code><max-x></code>	The maximum x coordinate in the coordinate system of the spatial column(s) being retrieved.
<code><max-y></code>	The maximum y coordinate in the coordinate system of the spatial column(s) being retrieved.

If all four coordinates of the search envelope are specified as zero, no search envelope is used.

Example:

```
SDERASTER_SEARCH_ENVELOPE 601190 5437839 611110 5447549
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE_COORDINATE_SYSTEM

Required/Optional: *Optional*

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data. The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

The syntax of the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is:

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

Workbench Parameter: [Search Envelope Coordinate System](#)

SEARCH_FEATURE

Required/Optional: *Optional*

Note: Currently only valid for vector features.

The SEARCH_FEATURE clause provides a mechanism for specifying an arbitrarily complex search feature. The SEARCH_FEATURE clause works with the SEARCH_METHOD clause to define the spatial constraint.

Parameter	Contents
[<xCoord> <yCoord>]+	A list of the coordinates defining the geometry of the query feature.

Example:

The example below defines an equivalent feature to the SDE30_SEARCH_ENVELOPE example shown above using the SDE30_SEARCH_FEATURE clause.

```
SDE30_SEARCH_FEATURE601190 5437839 601190 5447549 \
611110 5447549 611110 5437839 \
601190 5437839
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_METHOD

Required/Optional: *Required when SEARCH_FEATURE is used*

This statement specifies the type of spatial relationship the query features must have with the SEARCH_FEATURE or SEARCH_ENVELOPE, whichever is used, in order to be returned.

When SEARCH_ENVELOPE is used, the default value is SDE_ENVELOPE for vector mapping files/workspaces, SDE_AREA_INTERSECT for vector FME Objects applications, and

SDE_ENVELOPE for all raster related translations. When SEARCH_FEATURE is used, there is no default value and so this keyword must be specified.

When reading rasters, SDE_ENVELOPE is the only valid value and the SEARCH_FEATURE keyword is not supported.

Value:

The value of the search_method can be one of the following:

Parameter	Contents
SDE_ENVELOPE	Features must be within the envelope of the feature.
SDE_ENVELOPE_BY_GRID	Features must be within the envelope of the feature and are returned in grid order.
SDE_COMMON_POINT	Features must have a point in common with the query feature.
SDE_LINE_CROSS	Features must cross the query feature.
SDE_COMMON_LINE	Features must have a common line segment with the query feature.
SDE_CP_OR_LC	Features must have either a common point or a line crossing.
SDE_AI_OR_ET	Features must intersect or must share an edge.
SDE_AREA_INTERSECT	Features must intersect the query feature. This retrieves area, linear, and point features contained in or that intersect the area of the query feature.
SDE_AI_NO_ET	Features must intersect but must not have any edge touching with the query feature.
SDE_CONTAINED_IN	The returned features contain the query feature. A candidate feature that is an area will be returned when it encloses the query feature. If the candidate feature is a line, then it is returned if its path is coincident with the query feature. If the query feature is a point and the candidate feature is not an area, then the candidate feature will be returned if one of its vertices is the same as the query feature.
SDE_CONTAINS	The returned feature is contained by the query feature. If both the features are linear features, then the returned feature must lie on the search feature's path. Point features that lie on a search feature vertex are also returned.
SDE_CONTAINED_IN_NO_ET	Features must be contained within the query feature and not have any edge touching.
SDE_CONTAINS_NO_ET	Features must contain the query feature but must not share any edge.
SDE_POINT_IN_POLY	Returned feature must be an area feature that contains the first coordinate of the search feature.
SDE_IDENTICAL	The returned feature must be spatially identical to the query feature.

Example:

```
SDE30_SEARCH_METHOD SDE_AREA_INTERSECT
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_METHOD_FILTER

Required/Optional: *Optional*

Specifies if the features returned will or will not satisfy the spatial constraint. If FALSE is specified, then all features returned will not satisfy the spatial constraint. If TRUE is specified, then the features returned will satisfy the spatial constraint. Specifying FALSE enables you to select all features which are not contained by a feature.

Value: *TRUE | FALSE*

Default Value: *TRUE*

Example:

```
SDE30_SEARCH_METHOD_FILTER FALSE
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ORDER

Required/Optional: *Optional*

Note: Currently only valid for vector features.

Specifies the order that the underlying search is performed by the SDE.

Values:

OPTIMIZE – let SDE decide which to perform first

SPATIAL_FIRST – perform spatial query first

ATTRIBUTE_FIRST – perform tabular query first

Default Value: *OPTIMIZE*

Example:

```
SDE30_SEARCH_ORDER ATTRIBUTE_FIRST
```

Workbench Parameter: [<WorkbenchParameter>](#)

IDs

Required/Optional: *Required*

This statement specifies the tables from which features are to be retrieved. There may be multiple `SDE30_IDS` statements within a single FME mapping file, in which case the input set of tables comprises the union of all `SDE30_IDS` statements. The SDE reader module only extracts features from the identified tables.

If a read operation is performing a join operation then the `SDE30_IDS` for the join operation are specified using a compound ID.

Compound IDs have the following form:

```
SDE30_IDS A(B,C)
```

where the primary table in the query is A and the join operation during the read is joining with secondary tables B, and C. Secondary tables cannot contain layers (vector spatial columns) or raster columns.

The join operation that combines the tables is specified on the `SDE30_WHERE` clause. See the section *WHERE* on page 611, for a description of the `SDE30_WHERE` clause. The second example below illustrates how the `SDE30_IDS` are specified when the reader performs a multi-table join operation. Joins are not supported when reading rasters.

The general form of `SDE_IDS` is the name of the tables from which features are retrieved, separated with spaces:

```
<[table name[(sec_table[,sec_table]*)]]+>
```

Example 1:

For simple single table extracts, the `SDE30_IDS` is a list of table names as shown below, where features are read from the tables `roads`, then `streets`. No table combine operation occurs here. Each ID is treated as a separate query to the database.

```
SDE30_IDS roads streets
```

Example 2:

For more sophisticated queries in which join operations are desired, the `SDE30_IDS` has the following form. Note that there are no spaces between tables associated with a single query – spaces are used to separate different table identifiers. Each table identifier may be simple, as in Example 1, or compound, as in this example. The following ID statement is the counterpart to the second example for the `SDE30_WHERE` clause.

```
SDE30_IDS REPLICATION(DIST_CENTER, REPLICATION_LAYERS)
```

Note: Joins are not supported when reading rasters.

Workbench Parameter: [<WorkbenchParameter>](#)

ENVELOPE_QUERY_OPTIMIZATION

Required/Optional: *Optional*

Note: Valid only for vector features.

By default (that is, when this directive is set to `NO` or is not specified at all), the SDE Reader compares the envelope query used (if an envelope query is being used) to the largest possible extents for a given layer (vector spatial column). The largest possible extents are calculated using the X, Y origin of the layer and the X,Y scale of the layer. If the query envelope completely covers the largest possible extents for the current layer being read, then the envelope query is ignored for the current layer.

If this directive is specified with a value of `YES`, then the SDE Reader compares the envelope query used (if an envelope query is being used), to the extents of the layer as indicated by running the SDE administration command:

```
sdelayer -o describe_long -l <table name, layer name>
```

However, for this optimization to work, it is important that the extents of the layer are correct before running a translation. The extents can be updated using the SDE administration command:

```
sdelayer -o alter -l <table name, layer name> -E calc
```

If the extents are actually larger than currently set in SDE, running a translation may result in ignoring the query envelope used and returning erroneous features (i.e., features that would not be within the query envelope). However, if the extents are correct and if the query envelope completely covers the extents of the current layer, it is okay to ignore the envelope query for the current layer.

The purpose of this directive is to attempt to reduce the amount of time SDE spends executing a query on a table. One query is performed on each table being read (a multi-table join is considered one table). The greater the number of rows in a table, the longer a query may take.

If an envelope query is not being used, then setting this directive to either YES or NO will have no effect.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_ENVELOPE_QUERY_OPTIMIZATION YES
```

Workbench Parameter: [<WorkbenchParameter>](#)

CHILD_VERSION_NAME

Required/Optional: *Optional*

Note: This directive is only applicable on releases of ArcSDE that support versioning. It is also only valid for vector features.

Specifies the name of a child version to create using the version specified by `VERSION_NAME` as the parent version. All tables will then be read from this (child) version rather than from the parent version. All the tables read when this keyword is specified must be multiversioned and have read, insert, update, and delete permissions with the current user. This is because this keyword is designed to be used when checking out a copy of the data in the parent version, with the intention that the child version will be modified and possibly reconciled and posted back to the parent version. The version will be created as a public version and the description given to the version will be "Safe Software Created Version".

If `CHILD_VERSION_NAME` specifies a version that already exists, an error will be output. The child version will be owned by the user specified by `USERID`; therefore, if the owner is specified as part of the value for this directive, the owner must be the same as `USERID`. If the translation is aborted, then the child version created will be deleted. By default, this directive is left empty and therefore no child version is created.

Note: Version names are case-sensitive and hence the value supplied for this directive is also case-sensitive.

Example:

In the example below, `jim` is the name of the user who will own the child version when it gets created and `jim` is also the value that is currently used for the directive `USERID`. Specifying a different value for the user here than the one used in `USERID` will cause an error. Remember that it is not necessary to prefix the child version name with the user.

```
SDE30_CHILD_VERSION_NAME jim.field_edits
```

Workbench Parameter: [<WorkbenchParameter>](#)

MAX_FEATURES

Required/Optional: *Optional*

When specified, determines the maximum number of features the reader is allowed to read. Setting this keyword to zero means no limits are imposed on how many features can be read. This can be useful when it is uncertain how many features satisfy a given query, or when you want to read all the features in a given table unless there are many more than expected.

Value: *positive integer*

Default Value: *0*

Example:

```
SDE30_MAX_FEATURES 0
```

RASTER_PYRAMID_LEVEL_TO_READ

Required/Optional: *Optional*

When specified, determines the reduced resolution pyramid level to read raster data from. Zero denotes the base, full resolution pyramid. The raster must have had pyramids built when it was written to ArcSDE, and the pyramid level specified must be in the range of valid pyramid levels built.

Value: *positive integer*

Default Value: *0*

Example:

```
SDERASTER_RASTER_PYRAMID_LEVEL_TO_READ 2
```

USE_UNIFIED_DATE_ATTRS

Required/Optional: *Optional*

Specifies whether we want to use unified date attributes, where the date and time are read into one attribute, or whether we want to use split date attributes, where two attributes are produced, one with only the date and another with both the date and time.

The value of this keyword should not be changed. It is automatically set to YES in new mapping files and workspaces. To maintain backwards compability, if this keyword is not present, the reader will behave as though the keyword is set to NO.

Value: *YES | NO*

Default Value: YES (in new mapping files and workspaces), NO otherwise

Complete Reader Example

Simple Reader Example

The example below configures an SDE reader to extract features from the dataset `testdset` located on server `tuvok`. Only features located on the table named `roads`, that fall within the specified envelope and have an attribute `NUMLANES` with a value of 2 and an attribute `SURFACE` with a value of `GRAVEL`, are read from the SDE.

```
SDE30_DATASET testdset
SDE30_SERVER tuvok
SDE30_WHERE "NUMLANES=2 AND SURFACE='GRAVEL'"
SDE30_USERID joe
SDE30_PASSWORD bounce
SDE30_INSTANCE esri_sde
SDE30_VERSION_NAME joe-version
SDE30_SEARCH_ENVELOPE 601190 543783 611110 5447549
SDE30_IDS roads
SDE30_SEARCH_METHOD SDE_AREA_INTERSECT
```

Multi-table Join Example

The example below shows the specification of a multi-table join being performed during reading from the SDE database. The user specifies the tables upon which the query is to be performed using the `SDE30_IDS` clause, while the `SDE30_WHERE` clause specifies how the tables are to be combined to select the features to read. Once the compound ID is specified along with the `WHERE` clause, the features are processed as any other feature. The feature type assigned to the feature is the name of the primary table – in this example, this is `REPLICATION`.

```
SDE30_DATASET testdset
SDE30_SERVER tuvok
SDE30_USERID joe
SDE30_PASSWORD bouncy
SDE30_INSTANCE esri_sde
SDE30_VERSION_NAME joe-version

# Specify the tables upon which the table join is to be performed.
SDE30_IDS REPLICATION(DIST_CENTER,REPLICATION_LAYERS)

# Now specify the WHERE clause which specifies how the three
# tables above are to be combined
SDE30_WHERE \
"REPLICATION.REPLICATION_DATE is null AND " \
"REPLICATION.AREA_CD = DIST_CENTER.AREA_CD AND " \
"REPLICATION.IDENT = REPLICATION_LAYERS.IDENT AND " \
"DIST_CENTER.TOWN_CD = '$(TOWN)' "
```

Writer Overview

The SDE writer module stores FME features in an SDE database. The SDE writer module provides the following capabilities.

- **Transaction Support:** The SDE writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The SDE writer module uses the information within the FME mapping file/workspace to automatically create SDE tables as needed. If the tables will be storing geometry, then feature classes, raster maps, or raster catalogs can be created.
- **Table Validation:** When data is loaded into an existing table, the SDE writer module performs validation operations between the layer or raster column definition in the mapping file and that within the SDE. All discrepancies found are logged. All critical discrepancies result in the data load operation being halted.
- **Versioning Support:** The SDE writer provides versioning support that allows users to modify data to existing tables without affecting what other viewers see in their own versions of the SDE database. **Note:** Not currently supported for raster tables.
- **Geometry Demotion:** Demotes incorrect polygons to linear shapes if there are data problems, but the data is to be loaded "as is" into the SDE.
- **Non-Homogeneous Aggregate Loading:** Provides the ability to load non-homogeneous aggregates into SDE layers by splitting the aggregates into several homogeneous aggregates.
- **Non-spatial Index Creation:** The SDE writer module can also define non-spatial indices. Indices are specified to increase the performance of searches having a non-spatial component.
- **Update Capability:** The SDE writer module enables features to be updated in SDE through the use of specified keys and the `UPDATE` mode of operation. If area replacement of features is desired, this can also be accomplished by combining the capabilities of the SDE writer with the `SDE30QueryFactory`. **Note:** Updating is the only way to add data to a raster map table, and is referred to as mosaicking.
- **Rejected Features' Pipeline:** Features initially rejected by SDE can be sent through an FME Pipeline where changes can be made to the feature so that it can be accepted by SDE on its second attempt.
- **Geodatabase Writing Support:** Features can be written to an existing feature class located on an Enterprise Geodatabase.

Writer Directives

This section describes the keywords the SDE writer module recognizes. Each of the keywords is prefixed by the current `<WriterKeyword>_` when they are placed in a mapping file. By default, the `<WriterKeyword>` for writing vector data is `SDE30`, the default for writing rastermap data is `SDERASTERMAP`, and the default for writing raster catalog data is `SDERASTERCATALOG`.

RECONCILE_AND_POST

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This directive determines which changes to reconcile between the child version (i.e., the version specified by the connection-related directive `VERSION_NAME`) and its parent version. Conflicts must be resolved manually using ESRI ArcGIS. Valid values are `INSERTS`, `UPDATES`, `DELETES`, and `ALL`. More than one value can be specified as long as each value is separated by a space. A post of the child version to its parent will be automatically performed when `ALL` is specified, or when `INSERTS`, `UPDATES`, and `DELETES` are all specified. Upon successfully posting the child version to its parent, whether or not the child version is deleted is dependent upon the value of the `DELETE_CHILD_AFTER_RECONCILE_AND_POST` directive. The directive is also used to determine whether the child version is deleted when it is identical to its parent version, in which case no reconciliation or posting is needed.

The post will be performed automatically when `ALL` or `INSERTS`, `UPDATES`, and `DELETES` is specified. If an error occurs during the post phase, then all changes made during both the reconcile and post phases will be rolled back.

Value: *Any combination of `INSERTS`, `UPDATES`, `DELETES`, and `ALL`. Each value must be separated by a space.*

- `INSERTS` – features inserted in the parent version
- `UPDATES` – features updated in the parent version
- `DELETES` – features deleted in the parent version
- `ALL` – encompasses `INSERTS`, `UPDATES`, and `DELETES`.

Example:

In the example below, all the updates and deletes made to the child version will be reconciled with the parent version. If a conflict occurs, then none of the reconciled changes will be saved. If `INSERTS` was also specified, then a post back to the parent version would also occur, but since neither it nor `ALL` is specified, only a reconciliation will be performed.

```
SDE30_RECONCILE_AND_POST UPDATES DELETES
```

Workbench Parameter: [<WorkbenchParameter>](#)

TABLES_TO_RECONCILE

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This optional statement specifies a list of tables (separated by spaces) which should be reconciled. Using this directive, it is possible to use an ArcSDE writer just to reconcile and post changes (i.e., not to write features). It is also possible to specify additional tables to reconcile that were not written to during the current translation. This directive only gets used if the directive `RECONCILE_AND_POST` is specified. If no tables are specified (and `RECONCILE_AND_POST` is specified), then only the tables written to during the translation will be reconciled. If no tables were written to during the translation, then no tables will be reconciled.

Value: <[table name]*> separated by spaces. If a table is owned by a different user, then the table name must be prefixed by the owner.

Example:

```
SDE30_TABLES_TO_RECONCILE countries rivers cities
```

Workbench Parameter:

<WorkbenchParameter>**DELETE_CHILD_AFTER_RECONCILE_AND_POST**

Required/Optional: *Required*

Note: Valid only for vector features. Not supported for raster tables.

This directive determines whether to delete the child version following a reconcile and post, including the case where the child and parent version are identical. A value of 'YES' will delete the child version, while a value of 'NO' will leave it intact. The default value is 'YES'.

Value: *YES or NO.*

Example:

In the example below, the child version will not be deleted after the reconcile and post operation completes.

```
SDE30_DELETE_CHILD_AFTER_RECONCILE_AND_POST NO
```

TRANSACTION

Required/Optional: *Optional*

This statement instructs the SDE writer module to use transactions when loading data into the SDE. The SDE writer does not write any features to the SDE until the feature is reached that belongs to <last successful transaction> + 1. Specifying a value of 0 causes the SDE writer to use transactions and to write every feature to the SDE. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being rerun.

If the SDE30_TRANSACTION statement is not specified, then transactions are not used during the data load operation.

Value: <last successful transaction>

The transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Example:

```
SDE30_TRANSACTION 0
```

Workbench Parameter: <WorkbenchParameter>

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs FME about the number of features to be placed in each transaction before a transaction is committed to the database. When set to `VARIABLE` the SDE writer checks each feature for the `fme_db_transaction` attribute, for which there are 4 valid values:

- `COMMIT_BEFORE` - The current transaction is committed before writing the feature.
- `COMMIT_AFTER` - The current transaction is committed immediately after writing the feature.
- `ROLLBACK_AFTER` - The current transaction is rolled back immediately after writing the feature.
- `IGNORE` - The feature is written and no transaction handling occurs.

When the attribute is not found on the feature, then a value of `IGNORE` is assumed.

If the `SDE30_TRANSACTION_INTERVAL` statement is not specified, then a value of 100 is used as the transaction interval.

Value: `<transaction_interval>`

Either the number of features in a single transaction, or the value `VARIABLE`.

Default value: 100

Warning! If the `SDE30_TRANSACTION` statement is not specified, then transactions are not used during the data load operation, even if the `SDE30_TRANSACTION_INTERVAL` is specified.

Example:

```
SDE30_TRANSACTION_INTERVAL 50
```

Workbench Parameter: `<WorkbenchParameter>`

STRICT_LOAD

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This statement instructs FME to be very strict when loading spatial data from a feature into the SDE. When FME encounters a feature whose geometry cannot be converted into an SDE shape allowed by the layer (vector spatial column) for which it is destined, FME terminates the data load, logs the feature, and aborts the current transaction. In comparison to the `CONTINUE_TRANSLATION_BAD_DATA` directive, `STRICT_LOAD` is very limited in the type of errors that it can ignore. Use `CONTINUE_TRANSLATION_BAD_DATA` when it is desirable to ignore the majority of errors that may occur during a data load.

Value: `YES` | `NO`

Default Value: `NO`

Example:

```
SDE30_STRICT_LOAD YES
```

[Workbench Parameter: <WorkbenchParameter>](#)

FORCE_IN_AGGREGATES

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This statement instructs FME to make an extra effort to store multi-part polygon shapes (aggregates of polygons) into the SDE. When placed in this mode, the FME breaks apart aggregates that, according to the SDE, contain both polygons and lines, and attempt to store them as two feature aggregates. This is designed to assist with the loading of multi-part polygonal data in which some of the polygons are non-compliant with SDE's definition of a polygon.

To use this mode effectively, you must ensure that any polygonal layer (vector spatial column) for which this is applicable also allows for the storage of multi-part lines.

Value: YES | NO

Default Value: NO

[Workbench Parameter: <WorkbenchParameter>](#)

DEFAULT_Z_VALUE

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

The value to be used for the z coordinate when a 2D feature is forced to become 3D because the layer (vector spatial column) is defined as being 3D. The z value specified for this directive must be larger than the z origin.

Value: any real number

Default Value: 0

Example:

```
SDE30_DEFAULT_Z_VALUE 52.3
```

[Workbench Parameter: <WorkbenchParameter>](#)

LEAVE_LAYER_EXTENTS

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

By default, FME updates the layer (vector spatial column) extents when loading data into the SDE. This directive tells the FME not to perform this processing, thereby leaving the layer extent untouched.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_LEAVE_LAYER_EXTENTS NO
```

Workbench Parameter: [<WorkbenchParameter>](#)

SPLIT_DONUTS

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This directive is used when donut polygons are not to be stored as donuts, but rather simple polygons. When set to `YES` donut polygons are not stored in the SDE.

Value: `YES` | `NO`

Default Value: `NO`

Example:

```
SDE30_SPLIT_DONUTS NO
```

Workbench Parameter: [<WorkbenchParameter>](#)

CONTINUE_TRANSLATION_BAD_DATA

Required/Optional: *Optional*

This statement instructs the SDE writer to continue a translation even when an error occurs while attempting to load the data (the error may or may not be related to the data). A warning message will be output for each feature that could not be written to the SDE. The warning message will explain what went wrong. This directive is useful when trying to load bad data. Many more errors can be ignored using this directive than by using `STRICT_LOAD`. The `STRICT_LOAD` directive pertains only to converting geometry (from a feature) into an SDE shape to be written to a layer (vector spatial column).

When set to `ROLLBACK_THEN_CONTINUE`, if a feature fails to be written then the current transaction will be rolled back rather than committed when it comes time to commit the transaction. This means that none of the features in the rolled back transaction will be written to SDE. The translation will continue regardless of the error encountered. Transactions must be used when this value is specified. When used in conjunction with the `REJECTED_PIPELINE_DIRECTORY` keyword, if a feature returned from a pipeline fails to be written then the current transaction will be rolled back instead of committed.

If set to `YES` and transactions are being used, all transactions will be committed; however, failed features will not be written to SDE.

Value: `YES` | `NO` | `ROLLBACK_THEN_CONTINUE`

Default Value: `NO`

Example:

```
SDE30_CONTINUE_TRANSLATION_BAD_DATA ROLLBACK_THEN_CONTINUE
```

Workbench Parameter: [<WorkbenchParameter>](#)

REJECTED_PIPELINE_DIRECTORY

Required/Optional: *Optional*

This statement instructs the FME where to find the pipeline file(s) to be used. A pipeline is used when a failure occurs in writing a feature. When this statement is specified and there was an error in writing a feature, the writer first attempts to open a pipeline specific to the current table. The writer looks for a file called `<tableName>_pipeline.fmi` in the directory specified by this statement. If the file is not found, then the writer looks for a default pipeline called `default_pipeline.fmi` in the same directory. If neither of these files are found, then the translation is stopped.

If a pipeline file is found, then an FME pipeline is created using all the factories from the file. The pipeline can do almost anything a regular FME pipeline can do. However, only the first feature from the pipeline is retrieved. If the pipeline does not return any feature, then the writer does not insert into SDE the row that corresponds to the feature. At the present time, the feature is recorded as written in the statistics portion of the FME log, whether or not it was actually inserted into SDE.

If this directive is used with the `CONTINUE_TRANSLATION_BAD_DATA` directive set to `YES` or `ROLLBACK_THEN_CONTINUE`, and a feature is returned from the pipeline, then if the returned feature causes an error while being written to the SDE it will not cause the translation to stop. Rather a warning message, explaining why the feature couldn't be written, will be logged and the translation will continue. If `ROLLBACK_THEN_CONTINUE` was specified, the current transaction will be rolled back instead of committed when it comes time to commit the transaction.

If this statement is not specified, then no pipeline will be created by the writer for features rejected by SDE. A pipeline is only created if this statement is specified and a failure occurs in writing a feature

Value: *The absolute or relative path of a directory containing pipeline files. If a relative path and the command line FME are used, then the path is relative to the location where FME is called from. If a relative path and the Universal Translator are used, then the path is relative to the location of the mapping file. If the path contains spaces in it, the path should be double-quoted.*

Example:

```
SDE30_REJECTED_PIPELINE_DIRECTORY c:\sde\pipelines
```

Workbench Parameter: [<WorkbenchParameter>](#)

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This statement instructs the FME as to the type of mode in which it is to operate. If the value specified is `INSERT`, then the features being written to the database are not checked to see if duplicate key values are in the database. This is useful when loading new data into the SDE.

When the `WRITER_MODE` directive is set to `UPDATE`, the writer will check to see if the attribute `fme_db_operation` exists on the feature. If the attribute is set to `INSERT`, the feature will be inserted; if the value is `UPDATE`, the feature will be updated; and if the

value is `DELETE`, the feature will be deleted. If the attribute is set to any other value, the translation will be aborted and an error message logged. This functionality is designed to be used on tables that have an ArcSDE-maintained column, although it will also work on tables containing a spatial column.

If the writer mode is `UPDATE` and the attribute `fme_db_operation` is set to `UPDATE` or is not found on the feature, then the configuration parameter `SDE_UPDATE_FIELDS` (specified on a `SDE30_DEF` line) can be used to identify which features to update (it is never used for deleting features). It will only be needed if the table does not contain an ArcSDE-maintained (object ID) column and the table does not have a spatial column; however, if it is specified, then it will get used. If the configuration parameter `SDE_UPDATE_FIELDS` is specified, then the selected columns will make up a key. To avoid updating multiple features all at once, the user is responsible for ensuring that the specified key uniquely identifies a single feature within the SDE database. By using the RDBMS indices appropriately, the user should also ensure that a table scan of the underlying database will not result from each feature update operation.

If the `SDE30_WRITER_MODE` statement is not specified, then `INSERT` mode is used.

Writing a Raster Map

When writing a raster map to a table in SDE, the writer mode functions in a slightly different manner. `INSERT` mode inserts the raster into the table, overwriting any data that already pre-existed. `UPDATE` mode specifies that the writer is to mosaic the raster data to the pre-existing data, thereby updating the single row in the table. Raster catalogs may be inserted and updated to in a similar manner as vector data.

In order to perform a successful update, several conditions must be met by all the raster data that is to be mosaicked: the coordinate systems must be the same, the pixel depth must be the same, and the raster data itself must be either palette colored or continuous (they cannot be mixed). There is also a requirement for cell size and alignment to be the same, but the SDE writer will correct for these automatically, so they need not be altered. There is also no need to alter the writer mode when mosaicking, since the writer will automatically detect and correct this based on whether or not the data is pre-existing, in order not to overwrite it. The only way to overwrite existing raster data in a raster map is to set either the `SDE_DROP_TABLE` or the `SDE_TRUNCATE_TABLE` flag to `YES`.

Value: `INSERT` | `UPDATE` | `DELETE`

Default Value: `INSERT`

Example:

```
SDE30_WRITER_MODE INSERT
```

Workbench Parameter: [<WorkbenchParameter>](#)

BUFFERED_WRITES

Required/Optional: *Optional*

When specified, the buffered writing of the SDE is used which dramatically decreases the load time of data into the SDE.

Value: `YES` | `NO`

Default Value: *NO*

Example:

```
SDE30_BUFFERED_WRITES YES
```

Workbench Parameter: [<WorkbenchParameter>](#)

MAX_OPEN_TABLES

Required/Optional: *Optional*

Specifies the maximum number of streams that can be open simultaneously. Each stream writes to a particular table and so this directive determines the maximum number of tables that can be open and written to simultaneously. If this directive is not specified, or is given the value 0, then the SDE writer will set the maximum number of streams open simultaneously to 4 less than the number specified by `MAXSTREAMS` in `gi-omgr.defs`.

Value: *The maximum number of tables that can be open simultaneously*

Example:

```
SDE30_MAX_OPEN_TABLES 30
```

Workbench Parameter: [<WorkbenchParameter>](#)

ADD_LAYERS_TO_EXISTING_TABLES

Required/Optional: *Optional*

Specifies whether an existing business table within ArcSDE should have a layer (vector spatial column) added to it. To be eligible for this schema modification, the first feature written to the table must contain vector geometry.

Value: *YES | NO*

Default Value: *YES*

Example:

```
SDE30_ADD_LAYERS_TO_EXISTING_TABLES NO
```

Workbench Parameter: *Add Layers to Existing Tables:*

INTEGER_OVERRIDE_DEFINITION

Required/Optional: *Optional*

Specifies a definition to use for all integer column types when creating new tables. Any of the allowed FME attribute types for the ArcSDE Writer can be used as values for this directive. Additionally, the following can also be used: `number(<width>)` or `number(<width>, <decimal>)`. By default, this directive is not set, and so integer columns are stored using the C language long integer data type.

Values:

- *any allowed FME attribute types for the ArcSDE writer*
- *number(<width>) or number(<width>, <decimal>)*

Example:

In the example here, the ArcSDE database will use `char(30)` instead of `integer` as the type for all integer columns.

```
SDE30_INTEGER_OVERRIDE_DEFINITION char(30)
```

Workbench Parameter: [<WorkbenchParameter>](#)

Writing to an Enterprise Geodatabase

The SDE Writer is capable of writing features to an existing feature class that is located on an Enterprise Geodatabase. However, portions of the `SDE30_DEF` line must match the definition of the existing feature class. The following configuration parameters must match the existing definition:

1. `SDE_LAYER`
2. `SDE_DIMENSION`
3. `SDE_CAD`
4. `SDE_MEASURED`
5. `SDE_ANNOTATED`

All other configuration parameters do not need to be correct, and they will be ignored. Additionally, the non-spatial columns in the feature class do not need to be defined on the `SDE30_DEF` line.

SDE Table Representation

When reading from the SDE, it is not necessary that the source tables be defined. This is also true when writing to existing tables. However, if the SDE writer is going to create the tables, then definitions must be supplied. This is true even when the table already exists but the `SDE_DROP_TABLE` parameter has been set to `Y`. Within FME mapping files, SDE tables are defined using the `<WriterKeyword>_DEF` statement, whereas within Workbench they are defined by adding destination feature types.

When creating new tables, it is important to understand that the decision for which type of table to create is not based on the `DEF` line itself but rather the first feature written to the table. This means that if the first feature contains no geometry, then a business table (no spatial columns) will be created; if the first feature contains vector geometry, then a feature class (business table + layer) will be created; and if the first feature contains raster geometry, either a raster map or raster catalog will be created. The purpose of the `DEF` line is to specify what the table looks like.

If the table already exists in the database (and the user is not deleting it using the `SDE_DROP_TABLE` parameter) then its schema will not be altered. The only exception to this is when the first feature, within a translation, written to a pre-existing business table contains vector geometry and the directive `ADD_LAYERS_TO_EXISTING_TABLES` is set to `YES` or is not specified. In this case a layer will be added to the business table. The addition of the layer turns the business table into a feature class. If the `SDE_RASTERMAP` writer is used to write to an existing raster catalog, or the `SDE_RASTERCATALOG` writer is used to write to an existing rastermap, the translation will fail.

To define a simple table with no spatial or raster column using FME, the definition is in this form:

```
<WriterKeyword>_DEF <tableName> \
  [<columnName> <columndef>]*
```

A more general format of a table definition – in which a spatial column, along with attribute indices can be defined – is given here.

```
<WriterKeyword>_DEF <tableName> \
  [<columnName> <columndef>] * \
  [ [SDE_INDEX <indexName> \
SDE_INDEX_CONFIG <configKeyword> \
  SDE_COLUMN_NAME <columnName>[,<columnName>]* \
  SDE_UNIQUE <TRUE|FALSE|YES|NO> \
  SDE_SORT_ORDER ASCEND|DESCEND \
] * \
  [SDE_UPDATE_FIELDS <columnName> [,<columnName>]* \
  [SDE_STORAGE_TYPE <SDE_BINARY|WKB|SQL|NORMALIZED>] \
  SDE_LAYER <spatialColumnName> \
  [SDE_COORD_SYS_ID <coordSysID#>] \
  [SDE_COORD_SYS_DESCRIPTION <description>] \
  [SDE_PRECISION <32 | 64>] \
  SDE_GRID{0} <grid0size> \
  [SDE_GRID{1} <grid1size>] \
  [SDE_GRID{2} <grid2size>] \
  SDE_DIMENSION < 2 | 3 > \
  [SDE_CONFIG_KEYWORD <configKeyword>] \
  SDE_MEASURED < YES | NO > \
  SDE_ANNOTATED < YES | NO > \
  SDE_AREA < YES | NO > \
  SDE_LINE < YES | NO > \
  SDE_POINT < YES | NO > \
  SDE_SIMPLELINE < YES | NO > \
  SDE_NIL < YES | NO > \
  SDE_MULTIPART < YES | NO > \
  SDE_CAD < YES | NO > \
  SDE_XORIGIN <minimum_x> \
  SDE_YORIGIN <minimum_y> \
  SDE_SCALE <scale> \
  SDE_ZORIGIN <minimum_y> \
  SDE_ZSCALE <scale> \
  SDE_MEASURED_ORIGIN <minimum_y> \
  SDE_MEASURED_SCALE <scale> \
  SDE_TOLERANCE <tolerance> \
  SDE_MEASURED_TOLERANCE <tolerance> \
  SDE_ZTOLERANCE <tolerance> \
  [SDE_DESCRIPTION <layer description>] \
  [SDE_MINIMUM_FID <minimumFidNumber>] \
]
```

Another form of the general format of a table definition includes the definition of a raster column. An example is given here for a raster map.

Note: During creation of a new rastermap table, the writer will create a reserved ArcSDE column called NAME. This additional column will be created and populated with the value ESRI_SDERASTERDATASET. For raster catalogs, this column is optional, and populated by default with the value of the fme_basename attribute for each row in the table.

```

<WriterKeyword>_DEF <tableName> \
  [<columnName> <columndef>] * \
    [SDE_COORD_SYS_ID <coordSysID#>] \
    [SDE_COORD_SYS_DESCRIPTION <description>] \
    [SDE_CONFIG_KEYWORD <configKeyword>] \
    SDE_RASTER <rasterColumnName> \
    [SDE_COMPRESS_TYPE < NONE | LZ77 | JPEG | JPEG2000>] \
    [SDE_PYRAMID_INTERPOLATION< NONE | NEAREST_NEIGHBOR | \
      BILINEAR | BICUBIC >] \
    [SDE_PYRAMID_LEVEL_TYPE < NONE | AUTO | CUSTOM >] \
    [SDE_PYRAMID_MAX_LEVEL <maxLevel>] \
    [SDE_RASTER_STATS_TYPE < NONE | AUTO >] \
    [SDE_DESCRIPTION <raster description>] \
    [SDE_DROP_TABLE <YES | NO>] \
    [SDE_TRUNCATE_TABLE <YES | NO>] \
    [SDE_COMPRESS_COLORMAP <YES | NO>] \
    [SDE_RASTER_MOSAIC_MODE <NONE | MERGE | DELETE>]

```

The table definition for a raster catalog includes additional parameters for creating the spatial (footprint) column.

Note: If a table definition is given for a table that does not yet exist and the only column defined in the definition is the spatial or raster column, then the writer will create an object ID column maintained by ArcSDE called OBJECTID. This additional column will be created because ArcSDE does not allow tables to contain only a spatial/raster column.

```

<WriterKeyword>_DEF <tableName> \
  [<columnName> <columndef>] * \
    [SDE_COORD_SYS_ID <coordSysID#>] \
    [SDE_COORD_SYS_DESCRIPTION <description>] \
    [SDE_CONFIG_KEYWORD <configKeyword>] \
    SDE_RASTER <rasterColumnName> \
    [SDE_COMPRESS_TYPE < NONE | LZ77 | JPEG | JPEG2000>] \
    [SDE_PYRAMID_INTERPOLATION< NONE | NEAREST_NEIGHBOR | \
      BILINEAR | BICUBIC >] \
    [SDE_PYRAMID_LEVEL_TYPE < NONE | AUTO | CUSTOM >] \
    [SDE_PYRAMID_MAX_LEVEL <maxLevel>] \
    [SDE_RASTER_STATS_TYPE < NONE | AUTO >] \
    [SDE_DESCRIPTION <raster description>] \
    [SDE_MINIMUM_FID <minimumFidNumber>] \
    [SDE_DROP_TABLE <YES | NO>] \
    [SDE_TRUNCATE_TABLE <YES | NO>] \
    [SDE_XORIGIN <minimum_x>] \
    [SDE_YORIGIN <minimum_y>] \
    [SDE_SCALE <scale>]

```

<tableName>

This specifies the name of the SDE table being defined by the `SDE30_DEF` statement. The name must conform to the conventions and restrictions of the underlying RDBMS database.

The following example shows the first portion of the definition for a table named roads.

```
SDE30_DEF roads . . .
```

Attribute Definitions

This section of the `SDE30_DEF` statement defines the attributes for a table. A table must have at least one attribute.

- The <attribute name> specified within the FME mapping file must obey the following rules:
 - Attribute Names must be in uppercase.
 - Attribute Names must obey all length and character restrictions of the SDE.
- The <attribute definition> defines the type and optionality of the attribute, and has the following form:


```
<attribute type>, (optional|required)
```
- The supported attribute types are listed in the following table.

FME Attribute Type
smallint
integer
float
double
char(n)
blob
date
guid

The directive `optional` or `required` immediately follows the attribute type and indicates if the attribute is required. If nothing is specified, then the value defaults to `optional`.

The following example creates a required attribute called `NUMOFLANES` which is an `integer` type.

```
NUMOFLANES integer,required
```

smallint

This type is used to represent 16-bit integer values.

integer

This type is used to represent 32-bit integer values.

float

This type is used to represent 32-bit float values.

double

This type is used to represent 64-bit integer values

char(n)

This type is used to represent character values with a length not exceeding *n* characters. With SDE 9.2 and later, the `char(n)` column maps to a unicode column encoded in UTF-16 if the DBTUNE parameter `UNICODE_STRING` is set to `TRUE` or is not present.

blob

This is used to store arbitrary binary data in the SDE. See *@Reformat* and *@File* in the *FME Functions, Factories and Transformers* manual for a description of the `@Reformat` and `@File` functions, and for information on how to load and retrieve data into a blob attribute.

With the use of blob types coupled with `@System` and `@File` it is possible to store any arbitrary data with an SDE feature. If a feature has sound, video, images, or documents, or all of the above, they can be zipped up to form a compact package using `@System`. Next, `@File` can be used to load the zip file into an attribute of the SDE. The contents are then loaded directly in the database for later retrieval.

date

This is used to store and retrieve date information to the SDE.

When a date field is read by the SDE, two attributes are set in the FME feature. The first attribute has the name of the database column, and its value is of the form `YYYYMMDD`. This is compatible with all other FME dates.

The second attribute has a suffix of `.full` and is of the form `YYYYMMDDHHMMSS`. It specifies the date and the time, with the time portion specified using the 24-hour clock.

For example, if a date field called `UPDATE_DATE` is read, the following attributes will be set in the retrieved FME feature:

```
UPDATE_DATE = '19980820'  
UPDATE_DATE.full = '19980820201543'
```

When writing to the SDE, the writer looks for both attributes. Either may be in the form `YYYYMMDD` or `YYYYMMDDHHMMSS`. If both attributes are specified, then the value specified in `UPDATE_DATE.full` is used.

guid

This type is used to represent Globally Unique Identifiers (GUIDs), which are stored as text strings of length 36 within FME. The format of a GUID is 8 hexadecimal digits followed by a hyphen, then three groups of 4 hexadecimal digits, each followed by a hyphen, and then 12 hexadecimal digits. Note that { and } braces found at the beginning and end of the GUID are removed by the Reader and added on by the Writer if not present.

When writing to a required GUID field, the Writer will automatically generate a GUID if no value is supplied for it on the feature.

Example:

414EF035-DCDF-4DAD-96DA-E86C0DA661B2

SDE_INDEX <INDEXNAME>

This section of the `SDE30_DEF` line defines one or more non-spatial index columns. Non-spatial column indices are used to increase the performance of the non-spatial component of queries.

Each index definition is identified by a unique name and has the components shown in the following table.

Parameter	Contents
<code>SDE_INDEX_CONFIG</code>	The configuration keyword that describes the storage characteristics of the index tables. Example value is <code>DEFAULTS</code> .
<code>SDE_COLUMN_NAME</code>	A comma-separated list of the columns that make up the index.
<code>SDE_UNIQUE</code>	A flag to indicate if the index values are unique or not. Acceptable values are <code>N</code> or <code>Y</code> .
<code>SDE_SORT_ORDER</code>	The sort order of the index. Indicates whether the index returns the records in ascending or descending order. Acceptable values are <code>ASCEND</code> or <code>DESCEND</code> .

The following example defines an index called `countryCapital`. The index is ascending and is not unique. The index is built on the columns `COUNTRY` and `CAPITAL`. The index table storage characteristics are taken from the `dbtune.sde` file entries defined by `DEFAULTS`.

```
SDE_INDEX countryCapital \
SDE_INDEX_CONFIG DEFAULTS \
SDE_COLUMN_NAME COUNTRY,CAPITAL \
SDE_UNIQUE N \
SDE_SORT_ORDER ASCEND \
```

Configuration Parameters

There are a number of configuration parameters in the `SDE30_DEF` line that are used to define spatial column characteristics. They are described in the following table.

Note: The values populated in the settings box set values for configuration parameters.

Parameter	Contents
<code>SDE_LAYER</code>	<p>This defines the name of the spatial column within the table being defined. The spatial column is the column that contains the geometry of the feature.</p> <p>The following example gives the spatial column a name of <code>SHAPE</code>.</p> <pre>SDE_LAYER SHAPE</pre> <p>It is recommended that <code>SHAPE</code> be used as the name.</p>
<code>SDE_PRECISION</code>	<p>This optional field specifies whether to set the precision to 32-bit or 64-bit. If not specified in a workspace/mapping file, then it will be set to 32-bit; however, all newly created workspaces/mapping files specify this field and set it to 64-bit. When writing to ArcSDE 8.x or older, 32-bit precision will automatically get used since 64-bit support was added in ArcSDE 9.0.</p>
<code>SDE_COORD_SYS_ID</code>	<p>This optional field specifies the coordinate system of the spatial column. This is only used during the initial creation of a spatial column. The value is an integer value that corresponds to one of the predefined coordinate systems specified in ESRI's <i>Projection Engine</i> documentation which is shipped with every SDE 30.</p> <p>Either <code>SDE_COORD_SYS_ID</code> or <code>SDE_COORD_SYS_DESCRIPTION</code> can be specified, but not both.</p> <p>If it is not specified, then the coordinate system will taken from the first feature written to each table.</p>
<code>SDE_COORD_SYS_DESCRIPTION</code>	<p>This optional field specifies the coordinate system of the spatial column. This approach enables the entire projection to be specified using a description as defined in the ESRI Projection Engine documentation that is shipped with every SDE 30.</p> <p>As mentioned above, you specify either <code>SDE_COORD_SYS_ID</code> or <code>SDE_COORD_SYS_DESCRIPTION</code>, but not both.</p> <p>If it is not specified, then the coordinate system will taken from the first feature written to each table.</p>
<code>SDE_GRID{0}</code>	<p>This is specified as part of a spatial column definition. It gives the size of the spatial index in the coordinate system of the layer (vector spatial column).</p> <p>When set to -1, a spatial index will not be created. This is useful when a valid spatial index is not known. It also improves the speed of writing features.</p> <p>After the translation, the "Calculate Default Spatial Grid Index" tool (from ArcToolbox > Data Management Tools > Feature Class) can be used to calculate a valid spatial index. When <code>Grid{0}</code> is set to -1, level 2 & 3 grids do not get built even though the values for these levels get stored with the layer information.</p> <p>The following example defines the grid size of 200:</p> <pre>SDE_GRID{0} 200</pre>

Parameter	Contents
SDE_GRID{1}	<p>This optional parameter defines the level 2 grid element size. This is not needed for the majority of spatial columns. If specified, this must be at least 3 times the size of SDE_Grid{0}. If it is not desired, then either the value should not be specified or it should be given a value of 0.</p> <p>The following example defines a grid size of 600 for level 1 grid:</p> <pre>SDE_GRID{1} 600</pre>
SDE_GRID{2}	<p>This optional parameter defines the level 3 grid element size. This level grid is rarely required. If specified, this must be at least 3 times the SDE_GRID{1}. If it is not desired, then either the value should not be specified or it should be given a value of 0.</p> <p>The following example defines a grid size of 4000 for the level 2 grid:</p> <pre>SDE_GRID{2} 4000</pre>
SDE_DIMENSION	<p>The SDE requires that all features within a feature class have the same dimension. This parameter defines the dimension of the layer (vector spatial column). Currently, the dimension can be either 2 or 3.</p> <p>The example below defines the layer to have a dimension of 2:</p> <pre>SDE_DIMENSION 2</pre>
SDE_UPDATE_FIELDS	<p>The list of field names that are used by the SDE Writer when it is operating in UPDATE mode. If the table is either registered as multi-versioned or contains a spatial column, then this configuration parameter is optional. In general, this should identify a unique feature but can also be used to update multiple features if desired.</p> <p>The following example sets the update fields to be country and capital:</p> <pre>SDE_UPDATE_FIELDS COUNTRY,CAPITAL</pre>
SDE_XORIGIN	<p>The minimum x value of the spatial column being defined. No coordinate values can be less than the value specified here. For raster catalogs, this value should be calculated from the lower left corner of the lower-leftmost raster to be added to the catalog. If the value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the lower extent of a spatial column to be -180:</p> <pre>SDE_XORIGIN -180</pre>
SDE_YORIGIN	<p>The minimum y value of the spatial column being defined. No coordinate values can be less than the value specified here. For raster catalogs, this value should be calculated from the lower left corner of the lower-leftmost raster to be added to the catalog. If the value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the lower extent of a spatial column to be -90:</p> <pre>SDE_YORIGIN -90</pre>

Parameter	Contents
SDE_SCALE	<p>The scale of the spatial column. This defines the number of units per user coordinate stored within the spatial column. For raster catalogs, if this value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the scale to be 100: <code>SDE_SCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>
SDE_ZORIGIN	<p>The minimum z value stored within the spatial column. The example below defines the minimum z value to be 0: <code>SDE_ZORIGIN 0</code></p>
SDE_ZSCALE	<p>The scale of the spatial column z coordinate. This defines the number of units per user coordinate stored within the spatial column.</p> <p>The example below defines the z scale to be 100: <code>SDE_ZSCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>
SDE_MEASURED_ORIGIN	<p>The minimum measure value that is stored within the spatial column. The example below defines the minimum measure value to be 0: <code>SDE_MEASURED_ORIGIN 0</code></p>
SDE_MEASURED_SCALE	<p>The scale of the spatial column measured value. This defines the number of units per user coordinate that are stored within the spatial column.</p> <p>The example below defines the measured scale to be 100: <code>SDE_MEASURED_SCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>
SDE_DESCRIPTION	<p>The description of the spatial column, which is just free text. <code>SDE_DESCRIPTION RoadWork</code></p>
SDE_MINIMUM_FID	<p>The minimum feature ID assigned to shapes stored in the layer (vector spatial column). When the SDE stores shapes in a table, each shape is given an ID number that is unique throughout the table. If not specified, then the Feature ID starts at 1 for each spatial column.</p> <p>The only time this value needs to be specified is when tricks are being performed using the underlying RDBMS in which you want the Feature ID to be unique through a set of tables rather than throughout a single table.</p> <p>The example below results in the feature IDs starting at 100000 for the table upon which the statement is specified: <code>SDE_MINIMUM_FID 100000</code></p>

Parameter	Contents
SDE_CONFIG_KEYWORD	<p>The SDE configuration keyword specifies the storage parameters for the layer (vector spatial column) or raster column. Note that in releases before ArcGIS 9.3, the configuration keyword specified must be present in the \$SDEHOME/etc/db-tune.sde file. If not specified, the keyword DEFAULTS will be used.</p> <p>For more information, search <i>parameter name-configuration string pairs</i> in ESRI ArcGIS Server help files.</p> <p>The example below uses a configuration keyword of TEST: SDE_CONFIG_KEYWORD TEST</p>
SDE_MEASURED	<p>Y – The spatial column allows measures to be specified on each coordinate of the features. N – The spatial column does not allow measures.</p>
SDE_ANNOTATED	<p>Y – The spatial column allows annotation to be specified. N – The spatial column does not allow annotations.</p>
SDE_AREA	<p>Y – The spatial column allows area features to be stored. N – The spatial column does not allow area features to be stored.</p>
SDE_LINE	<p>Y – The spatial column allows linear features to be stored. Line features are those linear features that may touch or cross over themselves. N – The spatial column does not allow linear features to be stored.</p>
SDE_POINT	<p>Y – The spatial column allows point features to be stored. N – The spatial column does not allow points.</p>
SDE_SIMPLELINE	<p>Y – The spatial column allows simple lines to be stored. Simple lines are lines that do not touch or cross over themselves. N – The spatial column does not allow simple lines.</p>
SDE_NIL	<p>Y – The spatial column allows NIL features to be stored. NIL features are features that have a shape object with no coordinates. N – The spatial column does not allow NIL features.</p>
SDE_MULTIPART	<p>Y – The spatial column allows features that have multiple parts. Multi-part features must be homogeneous. That is, all parts must be either area, linear, or point within a single feature. N – The spatial column does not allow features which have multiple parts.</p>
SDE_CAD	<p>Y – The layer (vector spatial column) allows CAD data to be stored with it. This is for CAD client layers. FME is not capable of storing data in the CAD blob associated with the layer. N – The layer does not allow CAD data.</p>

Parameter	Contents
SDE_STORAGE_TYPE	<p>SDE_BINARY – the feature geometry for the layer (vector spatial column) is stored in SDE binary mode. This is the default and the only type that is supported for SDE 3.x.</p> <p>WKB – the feature geometry for the layer is stored in SDE using the OGC Well Known Binary form. ArcSDE 8.x only.</p> <p>SQL – Stored as SQL or well known text format. ArcSDE 8.x only.</p> <p>NORMALIZED – Normalized format (used for Oracle Spatial Only). ArcSDE8.x only.</p>
SDE_DROP_TABLE	<p>Specifies that the SDE writer drop the table before writing, and create a new one. For raster tables, the associated raster column and band information tables will be dropped as well. If the table does not exist, it will be created when the data is written. The writer expects that the general table type (i.e. raster, feature class/vector, business/non-spatial) of the new table will be the same as the table being deleted, with the exception of business tables where it is possible to delete a business table but create a feature class.</p> <p>The following example sets the drop table flag to false.</p> <pre>SDE_DROP_TABLE NO</pre> <p>Default: NO Values: YES NO</p>
SDE_TRUNCATE_TABLE	<p>Specifies that the SDE writer truncate the table before writing. For raster tables, the associated raster column and band information tables will be truncated as well. If the table does not exist, it will be created when the data is written.</p> <p>The following example sets the truncate table flag to false.</p> <pre>SDE_TRUNCATE_TABLE NO</pre> <p>Default: NO Values: YES NO</p>
SDE_TOLERANCE	<p>The cluster tolerance of the XY values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The XY tolerance is the minimum distance allowed between XY coordinates before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p>The example below defines the xy tolerance to be 0.001:</p> <pre>SDE_TOLERANCE 0.001</pre>

Parameter	Contents
SDE_MEASURED_TOLERANCE	<p>The cluster tolerance of the measured values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The measured tolerance is the minimum distance allowed between M values before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p>The example below defines the measured tolerance to be 0.001: SDE_MEASURED_TOLERANCE 0.001</p>
SDE_ZTOLERANCE	<p>The cluster tolerance of the Z values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The Z tolerance is the minimum distance allowed between Z values before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p>The example below defines the Z tolerance to be 0.001: SDE_ZTOLERANCE 0.001</p>
SDE_RASTER	<p>This defines the name of the raster column within the table being defined. The raster column is the column that defines the geometry of the table as raster and contains the geometry of the features in the table. If a table definition has both a spatial and a raster column, the spatial column will be ignored. The following example gives the raster column a name of RASTER.</p> <p>SDE_RASTER RASTER</p> <p>It is recommended that RASTER be used as the name.</p>
SDE_COMPRESS_TYPE	<p>This defines the type of compression to for the raster table being defined. The following example gives the raster column a compression type of LZ77.</p> <p>SDE_COMPRESS_TYPE LZ77</p> <p>Default: NONE</p> <p>Values: NONE LZ77 JPEG JPEG2000</p> <p>Note: LZ77 is the only valid compression option for images with a colormap. Also, JPEG2000 compression is only available for servers running SDE version 9.0 or later, and on rasters with an 8-bit pixel depth and no colormap.</p>

Parameter	Contents
SDE_PYRAMID_INTERPOLATION	<p>This defines the interpolation type of pyramid creation for the table being defined.</p> <p>The following example gives the raster column a pyramid interpolation type NEAREST_NEIGHBOR.</p> <pre>SDE_PYRAMID_INTERPOLATION NEAREST_NEIGHBOR</pre> <p>Default: NONE</p> <p>Values: NONE NEAREST_NEIGHBOR BILINEAR BICUBIC</p> <p>Note: A value of NONE disables pyramid creation and disregards the other pyramid settings. Also note that nearest neighbor is the only valid pyramid setting for classified raster data.</p>
SDE_PYRAMID_LEVEL_TYPE	<p>This defines the way the maximum pyramid level is set for the raster table being defined.</p> <p>The following example gives the raster column sets a maximum pyramid level to be automatically calculated.</p> <pre>SDE_PYRAMID_LEVEL_TYPE AUTO</pre> <p>Default: NONE</p> <p>Values: NONE AUTO CUSTOM</p> <p>Note: A value of NONE disables pyramid creation and disregards the other pyramid settings.</p>
SDE_PYRAMID_MAX_LEVEL	<p>This defines the maximum pyramid level to be created for the table being defined.</p> <p>The following example gives the raster column sets a maximum pyramid level to be automatically calculated.</p> <pre>SDE_PYRAMID_MAX_LEVEL AUTO</pre> <p>Note: This setting is only used if the pyramid level type is set to the value CUSTOM.</p>
SDE_RASTER_STATS_TYPE	<p>This defines the type of statistics calculation for the raster table being defined.</p> <p>The following example gives the user automatic determination of a statistics calculation function.</p> <pre>SDE_RASTER_STATS_TYPE AUTO</pre> <p>Default: NONE</p> <p>Values: NONE AUTO</p> <p>Note: A value of NONE turns statistics calculation off for this table.</p>

Parameter	Contents
SDE_RASTER_MOSAIC_MODE	<p>Specifies the mosaic mode that will be used when mosaicking data to an SDE rastermap. The default is MERGE. A value of NONE means that new data will completely replace existing raster data, and no mosaic is applied. MERGE causes the data to be mosaicked, replacing existing pixel values with new pixel values where they overlap, and leaving all other data untouched. The nodata values in the existing raster are not altered. DELETE mode does not mosaic any new data, but rather has the sole purpose of deleting data in the existing raster. This is accomplished by deleting pixel data where the new raster overlaps the existing raster and the value for that pixel location in the new raster is nodata. MERGE and DELETE modes are ignored on insert, and are only valid for a mosaic operation.</p> <p>The following example sets the mosaic mode to merge.</p> <pre>SDE_RASTER_MOSAIC_MODE MERGE</pre> <p>Default: MERGE Values: NONE MERGE DELETE</p> <p>Note: The DELETE mode is only supported by ArcSDE version 9.0 and later.</p>
SDE_RASTER_COMPRESS_COLORMAP	<p>This is used with palette colored rasters written to a raster map in SDE. It specifies whether the colormap is to be compressed or untouched. Compressing a colormap will remove any invalid entries, and possibly make future mosaic operations to the same table faster and less likely to approximate colors.</p> <p>The following example sets the compress colormap flag to true:</p> <pre>SDE_RASTER_COMPRESS_COLORMAP YES</pre> <p>Default: YES Values: YES NO</p>

Example of a Feature Class Definition

A typical SDE30 definition looks like this:

```
SDE30_DEF WORLD \
  SDE_LAYER WORLD_GEOM \
  SDE_GRID{0} 1000000 \
  SDE_DIMENSION 2 \
  SDE_CONFIG_KEYWORDDEFAULTS \
  SDE_MEASURED Y \
  SDE_ANNOTATED Y \
  SDE_AREA Y \
  SDE_LINE Y \
  SDE_POINT Y \
  SDE_SIMPLE_LINE Y \
  SDE_NIL Y \
  SDE_MULTIPART Y \
  SDE_XORIGIN -19000000 \
  SDE_YORIGIN -19000000 \
  SDE_SCALE 10 \
  SDE_ZORIGIN 0 \
  SDE_ZSCALE 1 \
  SDE_MEASURED_ORIGIN 0 \
```

```

SDE_MEASURED_SCALE1 \
SDE_TOLERANCE 0.0004 \
SDE_MEASURED_TOLERANCE \
SDE_ZTOLERANCE \
SDE_INDEX countryCapital \
SDE_INDEX_CONFIG DEFAULTS \
SDE_COLUMN_NAME COUNTRY,CAPITAL \
SDE_UNIQUE N \
SDE_SORT_ORDER ASCEND \
COUNTRY char(3) \
CAPITAL char(30) \
SDE_UPDATE_FIELDSCOUNTRY,CAPITAL
    
```

Example of a Raster Column Definition

A typical SDERASTERMAP definition looks like this:

```

SDERASTERMAP_DEF WORLD \
SDE_DROP_TABLE Y \
SDE_TRUNCATE_TABLE N \
SDE_CONFIG_KEYWORDDEFAULTS \ \
SDE_RASTER RASTER \
SDE_COMPRESS_TYPELZ77 \
SDE_PYRAMID_INTERPOLATION NONE \
SDE_PYRAMID_LEVEL_TYPERNONE \
SDE_PYRAMID_MAX_LEVEL0 \
SDE_RASTER_STATS_TYPEAUTO \
SDE_RASTER_COMPRESS_COLORMAP Y \
SDE_RASTER_MOSAIC_MODE MERGE
    
```

And an SDERASTERCATALOG definition might look like this:

```

SDERASTERCATALOG_DEF WORLD \
SDE_DROP_TABLE N \
SDE_TRUNCATE_TABLE N \
SDE_CONFIG_KEYWORDDEFAULTS \
SDE_RASTER RASTER \
SDE_COMPRESS_TYPELZ77 \
SDE_PYRAMID_INTERPOLATION NONE \
SDE_PYRAMID_LEVEL_TYPERNONE \
SDE_PYRAMID_MAX_LEVEL0 \
SDE_RASTER_STATS_TYPEAUTO \
SDE_RASTER_COMPRESS_COLORMAP Y \
SDE_XORIGIN -400000 \
SDE_YORIGIN -400000 \
SDE_SCALE 1000000
    
```

Using Versioning with the SDE Reader, Writer, and QueryFactory

Database states will be created by FME only when updating/inserting/deleting from a versioned table/feature class. Therefore, only the SDE writer or an SDE30QueryFactory in DELETE or UPDATE mode is able to create a database state.

The SDE reader and the `SDE30QueryFactory` in `QUERY` mode will never create a database state. All changes made during a single translation to a specific version (on the same SDE), even if they were made by different `SDE30QueryFactories` or different SDE writers, are placed into one (and the same) child state. Versioning must be used when updating/inserting/deleting from a versioned table/feature class. If versioning is not used in these cases, than an "Insufficient permissions" error will be generated and the translation stopped.

Note: Versioning is not supported at this time for raster data.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes* on page 7), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
<code>sde30_type</code>	The type of geometric entity stored within the feature. The valid values are listed below: <code>sde30_point</code> <code>sde30_nil</code> <code>sde30_line</code> <code>sde30_area</code> <code>sde30_circle</code> <code>sde30_ellipse</code> <code>sde30_simple_line</code> <code>sde30_raster</code>
<code>sde30_measures</code>	This is present for features that have measures when reading. To write measures, you simply build this list with one value for each vertex in the feature being written. This is a comma separated list of floating values which correspond to the vertex measures. The first value is for the first vertex, second for the second and so on.
<code>SE_ROW_ID</code>	For tables that are spatially enabled with vector data, this is the value for the internal SDE row number as defined by SDE.

Features read from, or written to, the SDE also have an attribute for each column in the database table.

Points

sde30_type: `sde30_point`

Features with this value are point features or a multi-part feature consisting of points. This is used by both the reader and the writer.

NIL Coordinates

sde30_type: `sde30_nil`

Features with this value are features or multi-part features consisting of no coordinates. This is used by both the reader and the writer.

Lines

sde30_type: sde30_line

Features with this value are features or multi-part features consisting of linear features. This type of linear feature is allowed to touch or cross over itself. This is used by both the reader and the writer.

Simple Lines

sde30_type: sde30_simple_line

Features with this value are features or multi-part features consisting of linear shapes. This type of linear feature is not allowed to touch or cross over itself. This is used by both the reader and the writer.

Areas

sde30_type: sde30_area

Features with this value are features or multi-part features consisting of area shapes. An area shape is a shape that forms either a polygon or a donut polygon.

Circles

sde30_type: sde30_circle

Features with this value are point features with the point specifying the centre of the circle. The rest of the circle is described using the following attributes.

Attribute Name	Contents
sde30_radius	The radius of the circle.
sde30_num_points	The number of points to use when creating the circle.

Features of this type are only used by the writer for reasons of convenience and are stored in the SDE as polygons. When read, they come out as area features tagged with sde30_type of sde30_area.

Ellipse

sde30_type: sde30_ellipse

Features with this value are point features with the point specifying the centre of the ellipse. An area shape is a shape that forms either a polygon or a donut polygon.

Attribute Name	Contents
sde30_major_axis	The major axis of the ellipse.

Attribute Name	Contents
sde30_minor_axis	The minor axis of the ellipse.
sde30_num_points	The number of points to use when creating the ellipse.
sde30_rotation	The angle of the ellipse given in degrees measured counter-clockwise from horizontal.

Features of this type are only used by the writer for reasons of convenience and are stored in the SDE as polygons. When read, they come out as area features tagged with sde30_type of sde30_area.

Rasters

sde30_type: sde30_raster

Raster features are stored in SDE as either raster maps or rows in raster catalogs. Raster maps are always read as one feature, whereas each row in a raster catalog is read as a separate raster feature.

Attribute Name	Contents
sde30_raster_compression	Overrides the value of the table level parameter SDE_COMPRESS_TYPE, on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_pyramid_interp_type	Overrides the value of the table level parameter SDE_PYRAMID_INTERPOLATION, on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_pyramid_level_type	Overrides the value of the table level parameter SDE_PYRAMID_LEVEL_TYPE, on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_pyramid_max_level	Overrides the value of the table level parameter SDE_PYRAMID_MAX_LEVEL, on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_stats_type	Overrides the value of the table level parameter SDE_RASTER_STATS_TYPE, on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_mosaic_mode	Overrides the value of the table level parameter SDE_RASTER_MOSAIC_MODE, on a feature-by-feature basis. This attribute is only used for raster maps, and will be ignored if specified on features to be added to a raster catalog.

Annotation

The SDE30 enables annotation information to be attached to any feature within its database. Unlike other systems where text or annotations are standalone features, in the SDE30 annotation is an optional part of any feature. It should be noted that annotations can only be stored in spatial columns where annotation is permitted.

The following attributes are used to store the annotation information within an FME feature. If the `sde30_text_string` is specified and no location or position information is stipulated, then the `text_string` is placed at the first coordinate of the associated feature and given a rotation of 0.

Attribute Name	Contents
<code>sde30_text_string</code>	The annotation string.
<code>sde30_text_size</code>	The size of the text in user units. Default: 1.0
<code>sde30_text_gap_ratio</code>	The gap between the characters in the text. Default: 0.0
<code>sde30_text_level</code>	The annotation level. Default: 1
<code>sde30_text_x_offset</code>	The x coordinate of the first point of the annotation offset. Default: 0.0
<code>sde30_text_y_offset</code>	The y coordinate of the first point of the annotation offset. Default: 0.0
<code>sde30_text_symbol</code>	The annotation symbol number. Default: 1
<code>sde30_text_x</code>	The location of the text when the text is placed with a single point. This is used in conjunction with <code>sde30_rotation</code> . If not specified, then the first point of the associated shape is used.
<code>sde30_text_y</code>	The location of the text when the text is placed with a single point. This is used in conjunction with <code>sde30_rotation</code> . If not specified, then the first point of the associated shape is used.
<code>sde30_text_z</code>	The location of the text when it is placed with a single point. This is used in conjunction with <code>sde30_rotation</code> . If not specified, then the first point of the associated shape is used.
<code>sde30_text_x_location</code>	An array of x coordinates that define the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: <code>sde30_text_x_location</code> , <code>sde30_text_y_location</code> and <code>sde30_text_z_location</code> (optional)
<code>sde30_text_y_location</code>	An array of y coordinates that defines the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: <code>sde30_text_x_location</code> , <code>sde30_text_y_location</code> and <code>sde30_text_z_location</code> (optional)

Attribute Name	Contents
sde30_text_z_location	An array of z coordinates that defines the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_location, sde30_text_y_location and sde30_text_z_location (optional)
sde30_text_x_leader	An array of x coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_leader, sde30_text_y_leader and sde30_text_z_leader (optional)
sde30_text_y_leader	An array of y coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_leader, sde30_text_y_leader and sde30_text_z_leader (optional)
sde30_text_z_leader	An array of z coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_leader, sde30_text_y_leader and sde30_text_z_leader (optional)
sde30_rotation	The rotation of the annotation measured from the horizontal in a counterclockwise direction. This is only used when the annotation location is specified using sde30_text_x, sde30_text_y, and sde30_text_z. Default: 0
sde30_justification	The justification of the text relative to its offset point. Range: sde30_upper_left sde30_upper_center sde30_upper_right sde30_center_left sde30_center_center sde30_center_right sde30_lower_left sde30_lower_center sde30_lower_right Default: sde30_lower_left

Troubleshooting

Connecting to ArcSDE

Problems sometimes arise when attempting to connect to an ArcSDE database. This is almost always due to a misconfiguration in the user's environment. The following suggestions can often help detect and overcome such problems.

- Ensure you have configured the services file so that the port number specified for the SDE instance you are connecting to matches that of the server.

- To test connectivity to ArcSDE, get the `sde3list.zip` file from `ftp://ftp.safe.com/fme/misc` and unzip `sde3list.exe` into your FME installation directory.

Change to the FME installation directory and execute the following command.

```
sde3list <server> <instance> <database> <username> <password> <outputfile>
```

For example:

When querying an SDE on a DBMS that doesn't have a notion of multiple databases per instance (for example, Oracle), the command would look as follows:

```
sde3list KHAN sde_esri not_used chris password tables.txt
```

Note that the a value must be specified for the database. Here, we specified `not_used`.

When querying an SDE on a DBMS that does support multiple databases per instance, the command would be as follows:

```
sde3list KHAN sde_esri myDb chris password READ tables.txt
```

The `sde3list` program is a simple program that connects to the database and retrieves all available tables. If this program cannot be made to work, then FME will not be able to connect to the database either.

Miscellaneous

- It is not possible to write to an ArcSDE business table with one column when it is a blob column. The table must have an additional column as well.