

ESRI Geodatabase Reader/Writer

FORMAT NOTES:

- To use FME's ESRI Geodatabase Reader/Writer, you must also install ArcGIS® Desktop.

The Geodatabase reader and writer modules allow FME to store data in and retrieve data from ESRI's Geodatabase. Support is provided for translating several aspects of a Geodatabase, and with the size of ESRI's ArcObjects, further expansion of the reader/writer will almost certainly continue to cover more diverse aspects of the format.

Geodatabase Quick Facts

Format Type Identifier	GEODATABASE_SDE (ArcSDE) and GEODATABASE_MDB (Access) and GEODATABASE_FILE (File-based)
Reader/Writer	Both
Licensing Level	Various
Dependencies	ArcGIS Desktop
Dataset Type	Database (ArcSDE) File (Access) Directory (File-based)
Feature Type	Feature Class name
Typical File Extensions	for Personal Geodatabase: .mdb for File-based Geodatabase: .gdb
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Always
Schema Required	Yes
Transaction Support	Yes
Rich Geometry	Yes
Geometry Type	geodb_type
Encoding Support	Yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	yes
donut polygon	yes	solid	no
elliptical arc	yes	surface	yes
ellipses	yes	text	yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
line	yes	z values	yes
none	yes		

Overview

The Geodatabase reader and writer translates several different types of features:

- table-level metadata
- reading and writing of geometric features such as points, multipoints, polylines, and polygons
- reading and writing of non-spatial table data
- reading and writing of annotations, including leader lines and feature-linked annotations
- reading and writing of dimensions
- reading and writing of geometric network features, including simple junctions, complex junctions (reading only), simple edges and complex edges

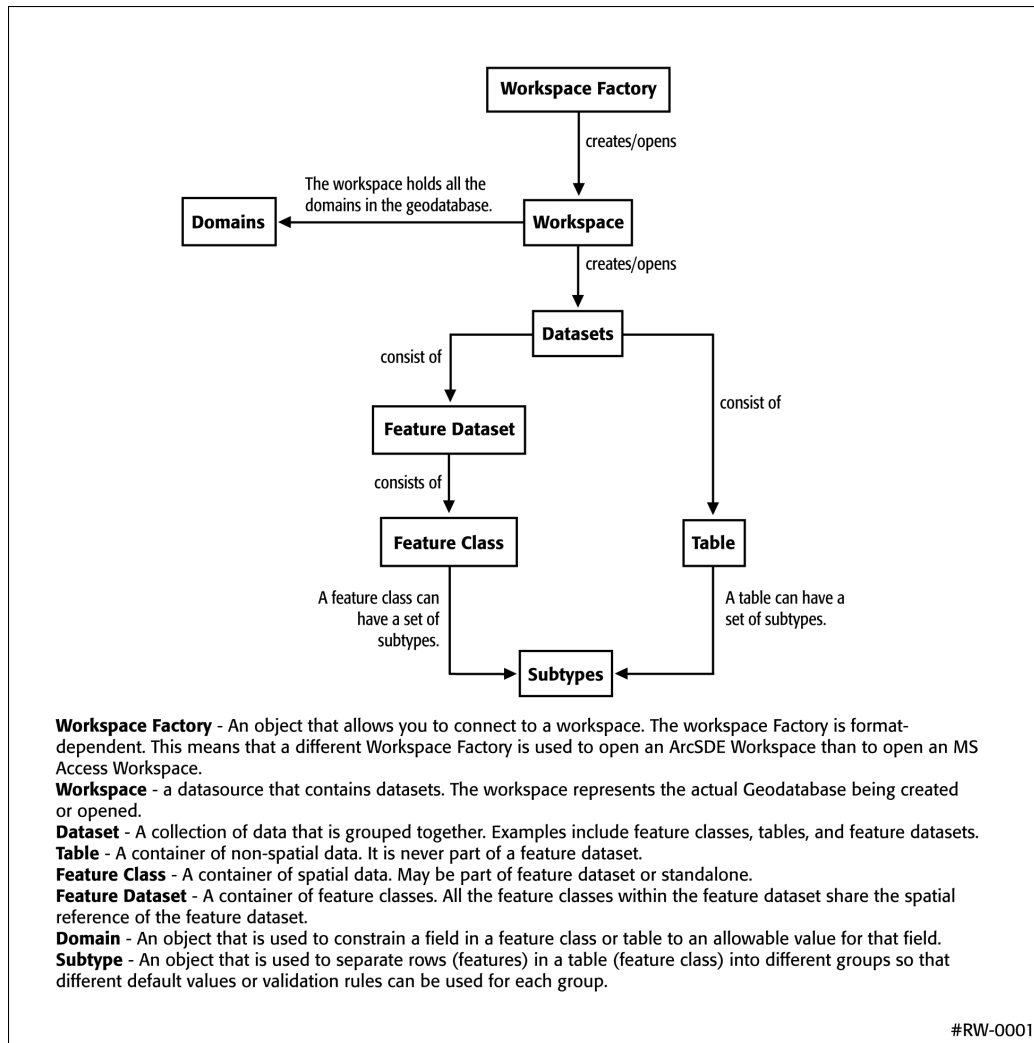
The Geodatabase modules also provide the following capabilities:

- **Programmatic Table Creation:** Tables need not be created before a data import operation. All table creation details are handled by the FME.
- **Transaction Support:** Transactions are fully supported, enabling a partially complete load of operation to be resumed later, without the loss or duplication of data.
- **Update/Delete Support:** In addition to appending features, the Geodatabase writer provides the ability to update and delete existing features in the Geodatabase.
- **Attribute Query Support:** SQL WHERE clauses can be specified to limit the data being exported.
- **Spatial Query Support:** FME exploits the spatial query capabilities of Geodatabase using both search envelopes and search features with a multitude of filtering options. This allows greater control to translate only the relevant spatial data.
- **Non-Spatial Table Support:** Any table can be read or written to a Geodatabase with FME, whether or not it contains spatial data. The FME can read, write, and create regular RDBMS tables (such as those in Oracle) in Geodatabase.
- **Versioning Support:** FME enables data to be read from a particular version of an Enterprise Geodatabase, and also allows data to be written to a specific version of an Enterprise Geodatabase.
- **Rich Geometry Model Support:** Both the Geodatabase Reader and Writer support the rich geometry model. The addition of rich geometry model support allows lines and polygons containing arcs to be maintained, rather than stroked.
- **Fully Automatic Import and Export:** The FME's Geodatabase support provides fully automated import and export of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports and exports.
- **Mapping File Customization:** The FME's ability to generate mapping files for user customization allows greater and more precise control over Geodatabase translations.
- **Unicode Support:** Geodatabase text columns are stored in the UTF-16 encoding. FME can read and write this data.

- Archiving Support: The Geodatabase reader can retrieve archived data from a table that has archiving enabled, through the use of a WHERE clause constrained by the `gdb_from_date` and `gdb_to_date` attributes.

Conceptual Diagram of Geodatabase

Below is a simplified diagram of some of the objects within Geodatabase. Labels describe the relationship between objects.



Reader Overview

The Geodatabase reader begins by opening the Geodatabase dataset that resides within a server/file system. Once opened, the Geodatabase reader queries the Geodatabase and passes the resulting features – that is, rows – on to the FME for processing. Every feature that is read is tagged with its original integer object ID (this is the object ID Geodatabase assigns it) under the attribute name `geodb_oid`.

Unlike other formats supported by FME, the Geodatabase reader has three different reader types. When reading from an Enterprise Geodatabase, the <ReaderType> is GEODATABASE_SDE, when reading from a Personal Geodatabase (an MS Access file), the <ReaderType> is GEODATABASE_MDB, and when reading from a File-based Geodatabase (a directory ending in .gdb), the <ReaderType> is GEODATABASE_FILE. By default, the <ReaderKeyword> is the same as the <ReaderType>.

When reading features from the Geodatabase, the tables from which features are retrieved are specified in the mapping file using the <ReaderKeyword>_IDs.

The Geodatabase reader uses the <ReaderKeyword>_IDs statement to identify the tables from which data is to be retrieved. If no identifiers (IDs) are specified and no DEF lines are specified and the Enterprise Geodatabase reader is used, then no features are read from the database. However, if no identifiers (IDs) are specified and no DEF lines are specified and the Personal Geodatabase or File-based Geodatabase reader is used, then all features are read from the database.

The table below summarizes the different feature retrieval modes supported by the Geodatabase reader module. The word *table* refers to both non-spatial tables and feature classes. However, *feature class* applies only to feature classes and not tables. The next section contains a detailed description of each directive.

Search Type	Search Directive Suffix	Description
Non-Spatial and Spatial Retrieval	IDs	Specifies the tables from which features are to be retrieved. If no tables are specified and the Personal Geodatabase or File-based Geodatabase reader is being used then all features are retrieved. If no tables are specified and the Enterprise Geodatabase reader is being used then no features are retrieved.
	WHERE	Specifies the attribute constraint that a feature must have to be retrieved. The where clause follows the SQL syntax of the underlying database, except that ORDER BY, GROUP BY, nested queries, and aggregate functions (i.e. MAX, COUNT) cannot be used.
Spatial Retrieval	SEARCH_ENVELOPE	Specifies the spatial extent of the feature retrieval. Only features that have the relationship specified by SEARCH_METHOD with the envelope are returned. This cannot be specified at the same time as a SEARCH_FEATURE is specified.
	SEARCH_FEATURE	Specifies a feature with an arbitrary number of coordinates as the search feature. Only features that have the relationship specified by SEARCH_METHOD with the search feature are returned. This cannot be specified at the same time as a SEARCH_ENVELOPE is specified. Also, when specifying the search_feature, make sure it has a simple geometry. If it does not have a simple geometry, then its geometry is always simplified by the Geodatabase reader.

Reader Directives – all Geodatabase Types

This section describes the directives that are recognized by the Geodatabase reader module. Each directive is prefixed by the current `<ReaderKeyword>_` when placed in a mapping file. Unless otherwise specified, the `<ReaderKeyword>` for the Geodatabase reader is the same as the `<ReaderType>`.

The following directives are used by all Geodatabase types.

FEATURE_READ_MODE

Required/Optional: *Optional*

This directive provides the ability to read table-level metadata when set to `Metadata`. In this mode, the reader outputs one feature per feature type. The `geodb_type` of the feature is `geodb_metadata` and the entire XML metadata document belonging to the Geodatabase table is found in the attribute `geodb_metadata_string`. Where applicable, the following attributes are also supplied: `fme_feature_identifier` which indicates the name of the object ID field, `fme_num_entries` (personal geodb only) which indicates the number of features in the table, `fme_contains_spatial_column` which indicates whether the table has a geometry column (i.e. in ESRI ArcGIS terms, whether the table is a feature class), `fme_geometry{0}` which indicates the types of geometry the feature class contains, `fme_dimension` which indicates whether the feature class is 2D or 3D. If the table is a feature class, the geometry of the metadata feature returned is a polygon, representing the extents of the feature class and the coordinate system of the feature class also gets set on the feature. When reading metadata, the `IDS` and `DEF` keywords are used to determine which feature types should have metadata read from them.

When set to `Features`, the reader outputs features stored within tables.

Parameter: `<feature_read_mode>`

Values: `Features` | `Metadata`

Default Value: `Features`

Example:

```
GEODATABASE_SDE_FEATURE_READ_MODE Metadata
```

Workbench Parameter: `Feature Read Mode`

WHERE

Required/Optional: *Optional*

An SQL-like (determined by the underlying database) WHERE clause that selects only certain records for extraction from the Geodatabase.

The specified `WHERE` clause is passed to the Geodatabase for processing. The `WHERE` clause can be almost like an SQL clause (using the syntax supported by the underlying database) except that `ORDER BY`, `GROUP BY`, nested queries, and aggregate functions (i.e. `MAX`, `COUNT`) cannot be used.

This `WHERE` clause applies to all tables retrieved. For more specific queries, see the Reader directive `DEF`.

Example:

The `WHERE` clause specified below instructs the FME to retrieve features from the Geodatabase for the tables that are listed on the `<ReaderKeyword>_IDs` lines (unless no `_IDs` lines are specified in which case all tables are examined). The features retrieved must have for their `ObjectID` a value greater than 10 and their `City` attribute must be Vancouver.

```
GEODATABASE_SDE_WHERE ObjectID > 10 AND \
                        City = 'Vancouver'
```

DEF

Required/Optional: *Optional*

Describes tables. Normally these lines are automatically generated within a mapping file using FME. When reading from an Enterprise Geodatabase, the table names on the `DEF` lines may be prefixed by the user ID of the person who created the table, followed by a period (for example, `<userid>.<tablename>`). The only table names that must be prefixed by a user ID are those tables that were not created using your own user ID. However, you may still not be able to access another person's tables if your user ID doesn't have the correct privileges.

This directive is usually automatically generated while generating a mapping file for a specific Geodatabase, but there is one way it can be customized. An automatically generated `DEF` might look like this:

```
GEODATABASE_MDB_DEF IndexGrid \
  geodb_type      geodb_polyline \
  GEODB_OID      integer \
  OBJECTID       integer \
  Entity         char(254) \
  Handle         char(254) \
  Layer          char(254) \
  Color          integer \
  Linetype       char(254) \
  Elevation      double \
  Thickness      double \
  SHAPE_Length  double
```

Note: The returned feature types will match the table names on the `DEF`s exactly (including the character case). As a result, if the `DEF`'s table name was `IndexGrid` (with no owner prefix) then the feature type of the returned features would also be `IndexGrid`. If the `DEF`'s table name was `sde.IndexGrid` then the feature type would be `sde.IndexGrid`.

Customizing Reader DEF Lines

With already generated and working `DEF` lines, a `WHERE` clause can be added just like a normal attribute on a `DEF` line above, except that instead of the type (such as `double`), the value will be an SQL `WHERE` clause. This clause **MUST** be in double quotation marks ("") and must conform to the same restrictions as the `WHERE` clause directive listed

above. In addition, the clause cannot be continued on to the next line so a continuation character (\) cannot appear in the middle of the clause.

Example:

The `WHERE` clause specified below instructs the FME to retrieve features from the feature class `IndexGrid` with the constraint that the `Color` value must be 5. Note that the `WHERE` clause is case-sensitive, and can appear on any line.

```

GEODATABASE_MDB_DEF IndexGrid \
    geodb_type      geodb_polyline \
    GEODB_OID       integer \
    OBJECTID        integer \
    Entity          char(254) \
    Layer           char(254) \
    Color           integer \
    Linetype        char(254) \
    Elevation       double \
    Thickness       double \
    WHERE           "Color = 5" \
    SHAPE_Length    double

```

IDs

Required/Optional: *Optional*

This statement specifies the tables from which features are to be retrieved. There may be multiple `GEODATABASE_<SDE|MDB|FILE>_IDS` statements within a single FME mapping file, in which case the input set of tables comprises the union of all `GEODATABASE_<SDE|MDB|FILE>_IDS` statements. The Geodatabase reader module only extracts features from the identified tables. If no `GEODATABASE_<SDE|MDB|FILE>_IDS` lines appear in the mapping file, then all tables with DEF lines will be used as the input set. If the Personal Geodatabase or File-based Geodatabase reader is being used and there are no DEF lines and no IDs, then all the tables will be read. If the Enterprise Geodatabase reader is being used and there are no DEF lines and no IDs, then no tables will be read. (This behavior is different from reading Personal and File-based Geodatabases.)

The returned feature types will match the IDs exactly (including the character case). As a result, if the IDs was `tableOne` (with no owner prefix) then the feature type would also be `tableOne`. If the IDs was `sde.tableOne` then the feature type would be `sde.tableOne`.

Parameter: `<[table name]+>`

Note: The table name must be exactly the same as it appears on the DEF line. For Enterprise Geodatabases, the tables on the DEF line may be prefixed by the user ID and a period (for example, `<userid>.`). If this is the case, then corresponding tables on the IDs line must also be prefixed by the username and a period.

Enterprise Geodatabase Example:

As shown below, the `GEODATABASE_SDE_IDS` is a list of table names. In the example, features are read from the table `roads`, and then from the table `streets`. Both tables are owned by a user named `jacob`. Each ID is treated as a separate query to the database. In this example, the assumption is made that the DEF lines for these tables also

contain the user ID and a period. If this was not the case, then no tables would be found because the table names on the IDs line would not match up with the table names on the DEF lines, even though they may be referring to the same table.

```
GEODATABASE_SDE_IDS jacob.roads jacob.streets
```

Personal and File-based Geodatabase Example:

As shown below, the `GEODATABASE_<MDB|FILE>_IDS` is a list of table names. In the example, features are read from the table `roads`, and then from the table `streets`. Each ID is treated as a separate query to the database. Reading from a File-based Geodatabase would look exactly the same except that the reader directive `GEODATABASE_MDB` would be replaced with `GEODATABASE_FILE`.

```
GEODATABASE_MDB_IDS roads streets
```

SEARCH_ENVELOPE

Required/Optional: *Optional*

Specifies a rectangular area to be used in conjunction with the `SEARCH_METHOD` directive for extraction of spatial features. This cannot be specified at the same time as a `SEARCH_FEATURE` is specified.

Parameters:

<min-x>

The minimum x coordinate in the coordinate system of the feature(s) being retrieved.

<min-y>

The minimum y coordinate in the coordinate system of the feature(s) being retrieved.

<max-x>

The maximum x coordinate in the coordinate system of the feature(s) being retrieved.

<max-y>

The maximum y coordinate in the coordinate system of the feature(s) being retrieved.

Example:

```
GEODATABASE_MDB_SEARCH_ENVELOPE 6190 57239 6310 57549
```

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>
```

SEARCH_FEATURE

Required/Optional: *Optional*

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, but cannot be specified at the same time as a SEARCH_ENVELOPE is specified.

Note: It is recommended you use a simplified search feature. If you don't, the reader will simplify the search feature and this could produce unexpected results.

Parameter: [*<xCoord> <yCoord>*]+ (A list of the coordinates defining the geometry of the query geometry.)

Example:

The example below defines an equivalent geometry to the GEODATABASE_MDB_SEARCH_ENVELOPE example shown above using the GEODATABASE_<SDE|MDB|FILE>_SEARCH_FEATURE clause.

```
GEODATABASE_SDE_SEARCH_FEATURE 6190 57239 6190 57549      \
6310 57549 6310 57239                                     \
6190 57239
```

SEARCH_ORDER

Required/Optional: *Optional*

Specifies the order that the underlying search is performed on the Geodatabase. This directive determines whether the spatial component or the attribute component of a query is performed first. The benefit of using this directive is for efficiency. For example, if the attribute component would filter the data more than would the spatial component, then it would be desirable to use the SEARCH_ORDER directive to force the attribute component to be used first. If the directive is not used, then by default the spatial component is used first.

Parameter: *<search_order>*

Value: *SPATIAL_FIRST | ATTRIBUTE_FIRST*

Example:

```
GEODATABASE_SDE_SEARCH_ORDER SPATIAL_FIRST
```

SEARCH_METHOD

Required/Optional: *Optional*

This directive specifies the type of spatial relationship the queried spatial features must have with either the `SEARCH_ENVELOPE` or the `SEARCH_FEATURE` in order to be returned. (Note that only one of `SEARCH_ENVELOPE` and `SEARCH_FEATURE` may be specified at a time.)

Parameter: `<search_method>`

Values: The values for the search method mostly follow the basic Clementini relationships that are used by ESRI. For further information on these relationships, see *Exploring ArcObjects Vol. II: Geographic Data Management Chapter 8*.

The value of the `SEARCH_METHOD` can be one of the following:

- `GEODB_INTERSECTS`: Features must intersect with the query geometry.
- `GEODB_ENVELOPE_INTERSECTS`: The envelopes of the features must intersect with the envelope of the query geometry.
- `GEODB_TOUCHES`: Features must touch the query geometry.
- `GEODB_OVERLAPS`: The query geometry overlaps the features returned.
- `GEODB_CROSSES`: The query geometry crosses the feature returned.
- `GEODB_WITHIN`: The query geometry is within the features returned.
- `GEODB_CONTAINS`: The query geometry contains the features returned.

Default value: `GEODB_INTERSECTS`

Example:

```
GEODATABASE_MDB_SEARCH_METHOD GEODB_CONTAINS
```

SPLIT_AT_ARCS (only applicable with classic geometry)

Required/Optional: *Optional*

This directive specifies whether or not to vectorize arcs. When set to `NO` arcs that are a piece of paths or polygons are vectorized - arcs not part of a larger geometry will remain as arcs. When set to `YES` during workspace/mapping file generation, all polygon feature classes will indicate that they contain polyline geometry rather than polygon geometry because it is assumed that the value for this directive will remain as `YES` for the translation. Changing the value to `NO` for the translation may produce unexpected results.

When this directive is set to `YES` for the translation, arcs are not vectorized and lines/polygons containing arcs are split up into arcs and lines. Each piece receives all the user-defined attributes, and gets tagged with an additional two attributes:

`geodb_segment_index` and `geodb_original_geometry`. The attribute `geodb_segment_index` is zero-based (i.e., the first piece has an index of 0) and can be used to piece back together the original geometry. The second attribute, `geodb_original_geometry`, indicates whether the original geometry was a `line`, `polygon`, or a `donut`. A piece will only be tagged as a `donut` if it was a hole in a polygon or if it was a shell that contained holes. As a result, an island that does not contain any holes will have `geodb_original_geometry` set to `polygon`. If the piece is an arc or an ellipse, it will contain additional attributes describing its characteristics.

When this directive is set to `NO` for the translation, arcs are vectorized. As a result, polygon/polyline features with arc segments retain their original geometry, rather than be-

ing split up into individual pieces. When 3D arcs are vectorized, the z values of the arc are linearly interpolated from the start point to the end point.

When features are read using rich geometry, this directive will be ignored. To split rich geometry paths, use the PathSplitter transformer.

Parameter: *<split_at_arcs>*

Values: YES | NO

Default: NO

Example:

```
GEODATABASE_SDE_SPLIT_AT_ARCS YES
```

TRANSLATE_SPATIAL_DATA_ONLY

Required/Optional: *Optional*

This directive is used for translating spatial data only. When set to YES, non-spatial tables, relationships, domains, and subtypes will not be translated. If this directive is specified when generating a workspace or mapping file, then no schemas will be returned for non-spatial tables.

Parameter: *<translate_spatial_data_only>*

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_MDB_TRANSLATE_SPATIAL_DATA_ONLY YES
```

RESOLVE_DOMAINS

Required/Optional: *Optional*

This directive specifies whether to resolve attributes that have a default coded value domain (i.e., the domain was not set up through a subtype) associated with them. This means that when an attribute of a feature has a coded value domain associated with it, another attribute will also be added that represents the textual description of the coded attribute. The new attribute will be *<attribute-name>_resolved*, where *<attribute-name>* is the name of the attribute containing the code. This attribute will only be added when *<attribute-name>* contains a non-NULL value.

Parameter: *<resolve_domains>*

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_MDB_RESOLVE_DOMAINS YES
```

RESOLVE_SUBTYPE_NAMES

Required/Optional: *Optional*

This directive specifies whether to resolve the subtype field of a feature. A feature that exists in a table that has subtypes will have an attribute that is the subtype field. The subtype field will hold an integer value that specifies which subtype the feature belongs to, and this integer value also has a string name equivalent called the description. If `YES` is specified for this directive, the corresponding description will be added as an attribute on the feature, and the attribute will be `geodb_subtype_name`. When set to `YES` during the generation of a mapping file/workspace, the schema for a table with subtypes will contain the attribute `geodb_subtype_name`.

Parameter: `<resolve_subtype_names>`

Values: `YES` | `NO`

Example:

```
GEODATABASE_SDE_RESOLVE_SUBTYPE_NAMES YES
```

IGNORE_NETWORK_INFO

Required/Optional: *Optional*

This directive determines whether to read the network information belonging to a network feature. When set to `YES`, junctions will be treated as point features, and edges will be treated as polyline features, with the `geodb_type` being set to `geodb_point` and `geodb_polyline`, respectively. When set to `NO`, Geodatabase specific attributes describing network information such as network connectivity will be inserted on the feature. The geometry of the feature remains the same regardless of the value given to this directive. The speed of reading network features is vastly improved if the network info is ignored.

Parameter: `<ignore_network_info>`

Values: `YES` | `NO`

Default Value: `NO`

Example:

```
GEODATABASE_SDE_IGNORE_NETWORK_INFO YES
```

OMIT_GENERIC_OBJECTID_ATTRIBUTE

This directive is only used when generating a workspace or reading schema features using FME Objects.

This directive determines whether each schema should add an additional integer attribute called `geodb_oid`, a generic attribute representing the object id field. Valid values are `YES` and `NO`. This attribute can be useful when the real object id field is unknown because the value for this attribute will be the object id's of the features read. Regardless of the value for this directive, the real object id field will always be supplied on the schema. This directive affects schema generation only. This means that the attribute will still be on the feature during reading, even if the attribute wasn't on the schema.

Parameter: `<omit_generic_objectid_attr>`

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_SDE_OMIT_GENERIC_OBJECTID_ATTRIBUTE YES
```

SPLIT_COMPLEX_EDGES

This directive determines whether complex edge features should be split. When split, complex edge features are read at the **element** level rather than the **feature** level. The element level represents the logical view of the geometric network. As a result, no network connectivity information is lost. When split, each FME feature stores the following attributes:

Attribute Name	Contents
<code>geodb_element_id</code>	The element ID of the logical edge element.
<code>geodb_element_index</code>	An attribute created and assigned by FME. It is used to order the edge elements within a complex feature. The index begins at zero, not one.
<code>geodb_from_junction_element_id</code>	The junction element ID that corresponds to the <i>from endpoint</i> . Note: This is the <i>from endpoint</i> of the edge element, not the edge feature.
<code>geodb_to_junction_element_id</code>	The junction element ID that corresponds to the <i>to endpoint</i> . Note: This is the <i>to endpoint</i> of the edge element, not the edge feature.

The following complex edge attributes are not present on the FME feature: `geodb_junction_feature_count` and `geodb_edge_element_count`. Even though elements are being read, the `geodb_type` of each feature is still `geodb_complex_edge`.

If an error occurs when retrieving the geometry for an edge element, then the geometry is skipped but the network attributes are still read.

Parameter: `<split_complex_edges>`

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_MDB_SPLIT_COMPLEX_EDGES YES
```

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to 'Yes', indicates to the reader to return all the schemas of the tables in the database.

If this specification is missing then it is assumed to be 'No'.

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to `RETRIEVE_ALL_SCHEMAS`; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If `RETRIEVE_ALL_SCHEMAS` is also set to "Yes", then `RETRIEVE_ALL_SCHEMAS` will take precedence. If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

CHECK_SIMPLE_GEOM

Required/Optional: *Required*

This directive specifies whether a check should be performed on features read from geodatabase to determine whether they are simple. This is an expensive check and impacts reader performance.

Parameter: `<check_simple_geom>`

Range: YES | NO

Default: NO

Example:

```
GEODATABASE_MDB_CHECK_SIMPLE_GEOM YES
```

READ_FEAT_LINKED_ANNOS

Required/Optional: *Required*

This directive specifies whether feature-linked annotations should have their text, angle and position properties merged as attributes onto the main feature that they are linked to, when reading. If set to yes, this will produce a list attribute as detailed in the section on Annotations with the `geodb_anno_class_id`, `geodb_h_align`, `geodb_v_align`, `geodb_text_string`, `geodb_text_angle`, `geodb_text_word_spacing`, `geodb_text_character_width`, `geodb_text_x_coord` and `geodb_text_y_coord` set and the annotation table(s) need not be read explicitly. If set to no, feature-linked annotations will be read normally as annotations when encountered.

Parameter: <read_feat_linked_annos>

Range: YES | NO

Default: NO

Example:

```
GEODATABASE_MDB_READ_FEAT_LINKED_ANNOS YES
```

SPLIT_MULTI_PART_ANNOS

Required/Optional: *Optional*

This directive specifies whether or not to split multi-part annotations into separate features for each 'element' when reading. If set to yes, a single feature for each element (usually a word) in a multi-part annotation will be produced on reading, resulting in feature-specific attributes such as angle and text position being stored according to the location of each element. If set to no, multi-part annotations will be read normally, as a single feature storing a single set of attributes describing the positioning of the text.

Parameter: <split_multi_part_annos>

Values: YES | NO

Default: NO

Example:

```
GEODATABASE_SDE_SPLIT_MULTI_PART_ANNOS YES
```

VALIDATE_FEATURES

Required/Optional: *Optional*

This directive specifies whether or not to validate features passed to the geodatabase writer. When set to 'yes', validation is performed on the subtype, attribute rules, relationship rules, network connectivity rules and any custom rules present on the feature class. Failed features will be logged with an extended error message describing the reason for the failure. When set to 'no', validation is not performed.

Parameter: <validate_features>

Values: YES | NO

Default: NO

Example:

```
GEODATABASE_SDE_VALIDATE_FEATURES YES
```

Reader Directives – Enterprise Geodatabase

These directives are used when connecting to an Enterprise Geodatabase.

CONNECTION_FILE

Required/Optional: *Optional*

This statement identifies the pathname of a connection file to be used to connect to an Enterprise Geodatabase. A connection file provides the necessary information to connect to the SDE server, such as the server name or the username. The connection file must be a *.sde file and have the proper format for a connection file as defined by ESRI. If you specify a connection file, you do not need to specify the directives SERVER, INSTANCE, USER, PASSWORD, and VERSION. (If, however, a password was not specified in the connection file, you will be prompted for a password.)

The pathname of the connection file to be used to connect to an Enterprise Geodatabase.

Parameter: *<connection_file>*

Example:

```
GEODATABASE_SDE_CONNECTION_FILE C:\GeoDB\connect.sde
```

DATASET

Required/Optional: *Required*

For Enterprise Geodatabases, this is the name of the dataset from which features are retrieved. In an Enterprise Geodatabase, this dataset is referred to as the DATABASE. Some RDBMS's, 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:

```
GEODATABASE_SDE_DATASET testdset
```

SERVER

Required/Optional: *Required*

The name of the Geodatabase server used to read data from the dataset.

Parameter: *<server>*

Example:

```
GEODATABASE_SDE_SERVER dax
```

INSTANCE

Required/Optional: *Required*

The Enterprise Geodatabase instance to which FME connects. The usual value for systems with a single ArcSDE instance is esri_sde8.

Parameter: *<instance>*

Example:

```
GEODATABASE_SDE_INSTANCE esri_sde8
```

USERID

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

User ID of the Enterprise Geodatabase user.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Parameter: *<userid>*

Example:

```
GEODATABASE_SDE_USERID jacob
```

PASSWORD

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

Password for the user account.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Parameter: *<password>*

Example:

```
GEODATABASE_SDE_PASSWORD jacobpassword
```

VERSION

Required/Optional: *Optional*

The version of the dataset to be read (used in multi-versioned databases). The version name must be prefixed by the owner of the version and a period. Note that the version name is case-sensitive.

Parameter: *<version>*

Example:

```
GEODATABASE_SDE_VERSION jim.versionone
```

CHILD_VERSION_NAME

Required/Optional: *Optional*

This optional directive specifies the name of a child version to create. The new version will be the child of the version specified by the VERSION directive. If CHILD_VERSION_NAME specifies a version that already exists, an error will be output. After the child version is created, data is read from the Geodatabase using this version instead of from VERSION. No default is provided for this directive, so no child version created in the default case.

Parameter: *<child_version_name>*

Example:

```
GEODATABASE_SDE_CHILD_VERSION_NAME check_out
```

ARCHIVE_WHERE**Required/Optional:** *Optional*

This optional directive specifies the where clause used to constrain features read from an archived table. The dates must be in FME date format, and will be converted to the format expected by the underlying database. One or both of the `GDB_FROM_DATE` or `GDB_TO_DATE` column names must be specified in order to form a valid where clause. If the `GDB_FROM_DATE` is not specified, the creation date of the archive will be assumed. If the `GDB_TO_DATE` is not specified, the current date will be assumed. ArcGIS uses transaction time to record changes to the archive, not valid time.

Workbench Parameter: *<archive_where>***Example of a 'moment' query:**

The following demonstrates an example of using the archive where clause to perform a 'moment' query, which will return all features existing in the database as of 9:00 am on May 1st, 2007

```
GEODATABASE_SDE_ARCHIVE_WHERE GDB_FROM_DATE <= 20070501090000 AND  
GDB_TO_DATE > 20070501090000
```

Example of a 'range' query:

The following demonstrates an example of using the archive where clause to perform a 'range' query, which will return all features inserted after 9:00 am on January 1st, 2007 and updated or deleted before 11:59 pm on December 31st, 2007

```
GEODATABASE_SDE_ARCHIVE_WHERE GDB_FROM_DATE > 20070101090000 AND  
GDB_TO_DATE < 20071231235959
```

Reader Directives – Personal and File-based Geodatabase

The directive listed below is used when connecting to a Personal Geodatabase (*.mdb file) or a File-based Geodatabase (directory ending in .gdb).

DATASET**Required/Optional:** *Required*

The file (Personal Geodatabase) or directory (File-based Geodatabase) from which data is to be read.

For Personal Geodatabases, an *.mdb file is specified when connecting to a Personal Geodatabase.

For File-based Geodatabases, a directory ending in .gdb is specified.

Personal Geodatabases Example:

```
GEODATABASE_MDB_DATASET "C:\FME\personalGeodb.mdb"
```

File-based Geodatabases Example:

```
GEODATABASE_FILE_DATASET "C:\FME\montgomery.gdb"
```

Improving the Speed of Translations Using the Geodatabase Reader

You can speed up translations involving the Geodatabase Reader by not resolving subtypes and coded value domains.

These operations add extra processing to each row of tables that contain subtypes or coded value domains.

Simple Reader Example

The example below configures a Geodatabase reader to extract features from the dataset `testdset` located on server `tuvok`. Only features located on the layer named `roads` that fall within the specified envelope are read from the Geodatabase.

```
GEODATABASE_SDE_DATASET testdset
GEODATABASE_SDE_SERVER tuvok
GEODATABASE_SDE_USERID joe
GEODATABASE_SDE_PASSWORD bounce
GEODATABASE_SDE_INSTANCE esri_sde8
GEODATABASE_SDE_VERSION sde.DEFAULT
GEODATABASE_SDE_SEARCH_ENVELOPE601190 543783 611110 5447549
GEODATABASE_SDE_SEARCH_METHOD GEODB_CONTAINS
GEODATABASE_SDE_IDS roads
```

Writer Overview

The underlying format that the Geodatabase writer module uses to store FME features depends on the `<WriterType>`. The `<WriterType>` is `GEODATABASE_MDB` when writing to Personal Geodatabases, `GEODATABASE_SDE` when writing to Enterprise Geodatabases, and `GEODATABASE_FILE` when writing to File-based Geodatabases. Note that use of the word *table* is meant to refer to both non-spatial tables and feature classes.

The Geodatabase writer module provides the following capabilities:

- **Versioning Support:** The Geodatabase writer provides the ability to write to a specific version of an Enterprise Geodatabase.
- **Update/Delete Support:** The Geodatabase writer provides the ability to update and delete existing features in the Geodatabase.
- **Transaction Support:** The Geodatabase 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 Geodatabase writer module uses the information within the FME mapping file to automatically create tables or feature classes as needed. If a

feature class is to be created, then certain parameters, such as the grid size and whether or not the features will contain Z values, can be specified. Personal Geodatabase table names are limited to a maximum length of 52 characters.

- **Automated Calculation of Extents and Grid 1 Size:** When writing to a Personal Geodatabase it is possible to set the writer in a mode whereby it will determine what the extents and grid 1 size of the dataset are and will use these values when creating new feature classes. This is not available with the Enterprise Geodatabase or File-based Geodatabase writer; however, valid grid sizes can be calculated and set when writing to Enterprise Geodatabase and File-based Geodatabase.
- **Ignoring Failed Features:** The Geodatabase writer allows the user to continue with a translation, even when a feature would normally cause the translation to fail. The user is given the ability to choose how many failed features are allowed to fail before deciding to halt the translation altogether. Additionally, the user can specify a directory in which to store failed features, which will be stored in the FME Feature Store format.

Writer Directives – all Geodatabase Types

This section describes the directives the Geodatabase writer module recognizes. Each of the directives is prefixed by the current `<WriterKeyword>_` when they are placed in a mapping file. By default, the `<WriterKeyword>` for the Geodatabase writer is the same as the `<WriterType>`.

Tip:

Due to the complexity and capabilities of a Geodatabase, writing features to one can be difficult, particularly when trying to create feature classes. The Personal Geodatabase writer has a mode whereby it will calculate the extents, scales, and grid sizes on your behalf.

To take advantage of this ability, set the `XY_SCALE` directive to zero. If a non-zero value for the directive `GRID_1` is specified, then the specified value will be used instead of using the grid 1 size calculated by the Personal Geodatabase writer. If no z values are found while calculating the extents, the minimum Z value will be `default_z - 20,000` and the Z scale will be `10,000`. The value for `default_z` will be taken from the directive `DEFAULT_Z_VALUE`. Please note that feature classes that exist before the translation will not have their extents and grid 1 size changed.

Even when the writer calculates the x,y,z origins, scales and grid 1 size, the DEF lines can override the calculated values by setting the configuration parameter `GEODB_XYSCALE` to a non-zero value. See `GEODB_XYSCALE` on page 725 for more information. (This functionality is not available when using the Enterprise Geodatabase or File-based Geodatabase writer.)

Note: It is important to understand that the initial values assigned to the writer directives, where possible, come from values in the settings box. The DEF line configuration parameters that correspond to the directives will not be assigned values. This prevents the problem whereby the values for the directives are changed, but never get used because they are overridden by the corresponding DEF line configuration parameters. The DEF line configuration parameters should only be used when the feature classes have differing dimensions (2d or 3d), origins, scales, grid sizes, and measure support.

The directives listed below are used by all Geodatabases.

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode* on page 19.

This statement instructs the Geodatabase writer on the type of mode in which it is to operate. When the `WRITER_MODE` directive is set to `UPDATE` or `DELETE`, the writer will check to see if the attribute `fme_db_operation` exists on the feature. A value of `INSERT` for this attribute means the feature will be inserted with no extra update processing; a value of `UPDATE` means the feature will be updated; and a value of `DELETE` means the feature will be deleted. If the attribute is set to any other value, the translation will be aborted and an error message logged. If the attribute is not set, the mode will be that indicated by the writer directive `WRITER_MODE`.

To update or delete a feature, the object ID must be on the feature passed into the Geodatabase writer. The object ID must be stored in an attribute with the same name as the object id field in the destination table. For example, if the destination table has an object ID field called `O_ID`, then this attribute must exist and be populated with the correct value on the FME feature.

As with inserting features, updating and deleting features on versioned tables with an ArcSDE requires that the `TRANSACTION_TYPE` directive be set to `VERSIONING`.

Parameter: `<writer_mode>`

Values: `UPDATE` | `DELETE` | `INSERT`

Default Value: `INSERT`

- `INSERT` – All features are inserted;
- `UPDATE` – By default, the features will be updated;
- `DELETE` – By default, the features will be deleted.

Example:

```
GEODATABASE_SDE_WRITER_MODE INSERT
```

TRANSACTION_TYPE

Required/Optional: *Optional*

This statement indicates which transaction mechanism the Geodatabase writer should use. Within ArcGIS, there are currently two transaction mechanisms: edit sessions and (regular) transactions. An edit session corresponds to a long transaction. During an edit session, edits made by other users do not become visible until the edit session is ended. If a translation does not complete successfully and the Geodatabase writer is using an edit session, then all the edits will be discarded.

Values: `VERSIONING` | `EDIT_SESSION` | `TRANSACTIONS` | `NONE`

- `VERSIONING`: Starts an edit session and then ends it when the translation is finished. This value should be used when writing to a versioned table in an Enterprise Geodatabase.

- **EDIT_SESSION:** Starts an edit session and then ends it when the translation is finished. This value should be used when edits are made to tables that have custom behavior associated with them.
- **TRANSACTIONS:** Starts the (regular) transaction mechanism. This can be used only when writing to non-versioned tables that do not have custom behavior.
- **NONE:** No transaction mechanism is used. This can be used only when writing to non-versioned tables that do not have custom behavior.

Default Value: *TRANSACTIONS*

Note: Currently there is no difference between choosing VERSIONING and EDIT_SESSION.

Example:

```
GEODATABASE_MDB_Transaction_TYPE EDIT_SESSION
```

TRANSACTION

Required/Optional: *Optional*

Transactions do not get used unless the `TRANSACTION_TYPE` directive is set to `TRANSACTIONS`. This statement instructs the Geodatabase writer when to begin to write features to the Geodatabase. The writer does not write any features to the Geodatabase until a feature is reached that belongs to `<last successful transaction> + 1`. Specifying a value of 0 causes the Geodatabase writer to use transactions and to write every feature to the Geodatabase. Normally, the value specified is zero – a positive non-zero value is only specified when a data load operation is being rerun.

If the `GEODATABASE_<SDE | MDB | FILE>_TRANSACTION` statement is not specified and transactions are being used (`TRANSACTION_TYPE` is set to `TRANSACTIONS`) then a default value of 0 is used.

Parameter: `<transaction #>` This is the transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Example:

```
GEODATABASE_MDB_TRANSACTION 0
```

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement tells FME the number of features to be placed in each transaction before a transaction is committed to the database.

If the `GEODATABASE_<SDE | MDB | FILE>_TRANSACTION_INTERVAL` statement is not specified, then a default value of 1000 is used as the transaction interval.

Parameter: `<transaction_interval>` The number of features in each transaction.

Default Value: *1000*

Example:

GEODATABASE_MDB_TRANSACTION_INTERVAL 50

Note: The current transaction is committed and a new transaction is started whenever a new table is created or opened, even if the transaction interval was not reached.

HAS_Z_VALUES

Required/Optional: *Optional*

This directive determines whether or not the dataset contains z coordinates. The value of this directive may be overridden by the DEF line parameter of GEODB_HAS_Z_VALUES if a value is specified for it. Valid values for this directive are YES, NO, or AUTO_DETECT. When set to AUTO_DETECT, the writer determines the dimension of the feature class by checking the dimension of the first feature headed for that feature class.

Parameter: *<has_z_values>*

Values: YES | NO | AUTO_DETECT

Default Value: AUTO_DETECT

Example:

GEODATABASE_MDB_HAS_Z_VALUES yes

DEFAULT_Z_VALUE

Required/Optional: *Optional*

This directive determines the z value to use when writing a 2D feature to a 3D feature class.

If the GEODATABASE_<SDE | MDB | FILE>_DEFAULT_Z_VALUE statement is not specified, then a default value of 0 is used.

Parameter: *<default_z_value>* The value to use for the Z coordinate(s) when writing a 2D feature to a 3D feature class.

Values: *real numbers*

Default Value: 0

Example:

GEODATABASE_SDE_DEFAULT_Z_VALUE -11.5

X_ORIGIN

Required/Optional: *Optional*

The X-coordinate of the origin for all feature classes (individual origins can be set – see *Geodatabase Table Representation* on page 713) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the X origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <x_origin>

Value: *real number*

Default Value: 0

Example:

```
GEODATABASE_MDB_X_ORIGIN -120.29
```

Y_ORIGIN

Required/Optional: *Optional*

The Y-coordinate of the origin for all feature classes (individual origins can be set – see *Geodatabase Table Representation* on page 713) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the Y origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <y_origin>

Value: *real number*

Default Value: 0

Example:

```
GEODATABASE_MDB_Y_ORIGIN -32.55
```

Z_ORIGIN

Required/Optional: *Optional*

The Z-coordinate of the origin for all feature classes (individual origins can be set – see *Geodatabase Table Representation* on page 713) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the Z origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <z_origin>

Value: *real number*

Default Value: *0*

Example:

```
GEODATABASE_SDE_Z_ORIGIN 120
```

XY_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes (individual scales can be set – see *Geodatabase Table Representation* on page 713) and all feature datasets. This is used to specify the level of precision to keep when storing XY coordinates, since all coordinates are stored as integers. Depending on the scale, it changes the precision of the coordinates stored. For example, if you have the coordinate (5.354, 566.35) and you set the XY_SCALE to be 100, then the coordinate stored will be (5.35, 566.35).

When writing to a Personal Geodatabase, if this value is set to 0 then the x,y,z origins and scales will automatically be calculated. These calculated values will be used instead of the values supplied by the writer directives for the x,y,z origins and scales. The grid 1 size will also be calculated, but will only be used if the value for the GRID_1 directive is 0. Even when the writer calculates the x,y,z origins, scales and grid 1 size, the DEF lines can override the calculated values by setting the configuration parameter GEODB_XYSCALE to a non-zero value. See *GEODB_XYSCALE* on page 725 for more information.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<xy_scale>*

Value: *real number greater than 0.*

When writing to a Personal Geodatabase, zero can be specified, in which case the writer will calculate the extents and scales itself. Only used when creating new feature classes.

Default Value: *100 when writing to an Enterprise Geodatabase; 0 when writing to a Personal Geodatabase*

Example:

```
GEODATABASE_MDB_XY_SCALE 1000
```

Z_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes (individual scales can be set – see *Geodatabase Table Representation* on page 713) and all feature datasets. This is used to specify the level of precision to keep when storing Z coordinates, since all coordinates are stored as integers. Depending on the

scale, it changes the precision of the coordinates stored. For example, if you have the z coordinate 5.354 and you set the Z_SCALE to be 100, then the coordinate stored will be 5.35.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <z_scale>

Value: *real number greater than 0.*

Default Value: *Enterprise Geodatabase writer: 100; Personal Geodatabase writer: 0*

The default value for a Personal Geodatabase is 0 because by default the writer calculates the extents and scales, and therefore ignores the value assigned to this directive.

Example:

```
GEODATABASE_SDE_Z_SCALE 10
```

HAS_MEASURES

Required/Optional: *Optional*

This directive determines whether or not the dataset contains measures. The value of this directive will be overridden by the DEF line parameter GEODB_HAS_MEASURES if a value is specified for it.

If the GEODATABASE_<SDE|MDB|FILE>_HAS_MEASURES statement is not specified, then a default value of NO is used.

This directive is used only when creating new feature classes.

Parameter: <has_measures>

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_MDB_HAS_MEASURES yes
```

MEASURES_ORIGIN

Required/Optional: *Optional*

The minimum measures value possible. This is used as an offset because measure data is stored as positive integers (0 to 2147483647) relative to the measures origin. This value is applied to all feature classes and feature datasets, although individual feature classes can override this value using the DEF line parameter GEODB_MEASURES_ORIGIN.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<measures_origin>*

Values: *real number*

Default Value: *0*

Example:

```
GEODATABASE_MDB_MEASURES_ORIGIN -412.98
```

MEASURES_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes. Individual feature classes can override this value using the DEF line parameter GEODB_MEASURES_SCALE. This is used to specify the level of precision to keep when storing measures, since all measures are stored as integers. Depending on the scale, it changes the precision of the measures stored. For example, if you have the measure 566.354 and you set the MEASURES_SCALE to be 100, then the measures stored will be 566.35. The value is only used when creating a new feature class.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<measures_scale>*

Values: *real number greater than 0*

Default Value: *100*

Example:

```
GEODATABASE_MDB_MEASURES_SCALE 10000
```

GRID_1

Required/Optional: *Optional*

This sets the global grid 1 size for the whole translation. It may be overridden if the DEF line has the setting for GRID{1} parameter. Valid values are real numbers greater than zero.

When using the Enterprise or File-based Geodatabase writer, if the value is 0 and no value is specified for the GRID{1} DEF line parameter (or it is 0) then the grid size will be automatically calculated. The File-based Geodatabase writer will also automatically calculate sizes for grids 2 & 3. This directive is only used when creating new feature classes.

Parameter: *<grid_1_size>*

Values: *real number greater than 0*

Default Value: *Personal or File-based Geodatabase writer: 0;
Enterprise Geodatabase writer: 1000*

Example:

```
GEODATABASE_MDB_GRID_1 45.6
```

IGNORE_FAILED_FEATURE_ENTRY

Required/Optional: *Optional*

This directive tells the Geodatabase writer whether or not it should ignore features that would normally cause the translation to fail. It allows you to ignore features that are topologically incorrect, are not supported by the Geodatabase writer, or conflict with the definition of the table to which it is to be inserted (that is, they are outside of the geometry envelope specified by the feature class). Additionally, polygons, donuts, or aggregates of polygons/donuts that cannot be reoriented will be ignored.

If the `GEODATABASE_<SDE|MDB|FILE>_IGNORE_FAILED_FEATURE_ENTRY` statement is not specified or given the value NO, then features are not ignored and will cause the translation to fail when encountered. Additionally, the three other associated directives described below will now be put into effect.

Parameter: *<ignore_failed_features>*

Values: *YES | NO*

Default Value: *NO*

Example:

```
GEODATABASE_SDE_IGNORE_FAILED_FEATURE_ENTRY YES
```

MAX_NUMBER_FAILED_FEATURES

Required/Optional: *Optional*

This directive informs the Geodatabase writer of the number of features to ignore before causing a translation to fail due to a problematic feature. (However, the translation may still fail for other reasons.)

This directive is only applicable if `IGNORE_FAILED_FEATURE_ENTRY` is set to YES.

Parameter: *<max_number_failed_features>*

Values: *To ignore all failed features: -1; otherwise 0 or a positive integer.*

Example:

```
GEODATABASE_SDE_MAX_NUMBER_FAILED_FEATURES 100
```

DUMP_FAILED_FEATURES

Required/Optional: *Optional*

This directive gives the user the option of storing the failed features to an FFS file so that they can be viewed at a later time. For this statement to be used,

GEODATABASE_<SDE|MDB|FILE>_IGNORE_FAILED_FEATURE_ENTRY must be specified and have the value YES.

Parameter: <dump_failed_features>

Values: YES | NO

Default: NO

FFS_DUMP_FILE

Required/Optional: *Optional*

This directive allows you to choose where the file containing failed features should be stored. The failed features will be stored in the FME Feature Store format. The file will be created automatically, but will only get created if there is a failed feature. If this directive is specified and a failed feature is encountered, then if a file with the same name as given to this directive already exists, it will be overwritten. This directive must be specified if

GEODATABASE_<SDE|MDB|FILE>_DUMP_FAILED_FEATURES

is specified and has the value YES.

Parameter: <ffs_dump_file>

Values: *path and filename*

If either the path or the filename contains a space, the value must be enclosed in double quotation marks. The filename must end in the extension .ffs.

Example:

```
GEODATABASE_MDB_FFS_DUMP_FILE "c:\user temp\bad features.ffs"
```

ANNOTATION_UNITS

Required/Optional: *Optional*

This directive allows you to specify which map units should be used when creating a new annotation feature class. Its value will be applied to all annotation feature classes created by the writer identified by <WriterKeyword>. A Multi-Writer should be used when annotation feature classes with different map units need to be created. This directive is not used when opening an existing annotation feature class. If the writer creates an annotation feature class, and the directive ANNOTATION_UNITS is set to `unknown_units` (the default value), then the writer tries to determine what type of unit the spatial reference uses and sets ANNOTATION_UNITS to the closest unit that is greater than or equal to it (with respect to its meters per unit value). If a local/unknown coordinate system is used, the units are set to meters.

Parameter: <annotation_units>

Values: *unknown_units, decimal_degrees, inches, points, feet, yards, miles, nautical_miles, millimeters, centimeters, meters, kilometers, and decimeters*

Default Value: *unknown_units*

Example:

GEODATABASE_MDB_ANNOTATION_UNITS nautical_miles

SIMPLIFY_GEOM

Required/Optional: *Optional*

Note: You can use this directive only if you have installed ArcGIS 9. It will not work with ArcGIS 8.

If it is set to YES, then the geometry being written out is simplified (if it is currently a non-simple geometry).

Value: YES | NO

Default Value: NO

Writer Directives – Enterprise Geodatabase

The directives listed below are used when connecting to an Enterprise Geodatabase. The CONNECTION_FILE, SERVER, USERID, PASSWORD, and INSTANCE directives operate in the same manner as they do for the Geodatabase reader.

The other writer-specific directives are discussed in the following sections.

CONNECTION_FILE

Required/Optional: *Optional*

The pathname of a connection file for an Enterprise Geodatabase. Having a connection file means that it is not necessary to specify the other directives.

DATASET

Required/Optional: *Required*

The Enterprise Geodatabase dataset from which data is to be written.

SERVER

Required/Optional: *Optional*

The server machine where the dataset resides.

INSTANCE

Required/Optional: *Optional*

The Enterprise Geodatabase instance to connect to.

USERID

Required/Optional: *Optional*

User ID of the Enterprise Geodatabase user.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

PASSWORD

Required/Optional: *Optional*

Password for the user account.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

VERSION

Required/Optional: *Optional*

The name of the version to which features should be written (only applicable on multi-versioned databases).

Parameter: *<version>*

Example:

```
GEODATABASE_SDE_VERSION jim.testversion
```

RECONCILE_AND_POST

Required/Optional: *Optional*

This optional directive reconciles all the changes between the version specified by the writer directive `VERSION` and its parent version, even those edits made outside of the current translation, and then posts the version to its parent. The reconcile and posting is done at the end of the translation when the Geodatabase writer is being shut down. As a result, if an error occurs during the reconcile or post, then only the reconcile and post changes are undone; all the features that were inserted/updated/deleted prior are still saved. The reconcile and post will only be successful if no conflicts are found. Conflicts must be resolved manually using ESRI's ArcGIS. Upon a successful post, the child version will be deleted or left intact depending on the value of the `DELETE_CHILD_AFTER_RECONCILE_AND_POST` directive.

Since reconciling is done for all features including those inserted/updated/deleted outside the current translation, then this directive can be used in an empty workspace to simply reconcile and post a child version to its parent. Since this directive reconciles and posts the version specified by `VERSION`, the value for `VERSION` must not be `SDE.DEFAULT` because `SDE.DEFAULT` has no parent version. When reconciling and posting, the `TRANSACTION_TYPE` directive must be set to `VERSIONING`.

Parameter: *<reconcile_and_post>*

Values: *YES | NO*

Default Value: *NO*

Example:

```
GEODATABASE_SDE_RECONCILE_AND_POST YES
```

DELETE_CHILD_AFTER_RECONCILE_AND_POST

Required/Optional: *Required*

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'.

Parameter: `<delete_child_after_reconcile_and_post>`

Values: YES | NO

Default Value: YES

Example:

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

```
GEODATABASE_SDE__DELETE_CHILD_AFTER_RECONCILE_AND_POST NO
```

Writer Directives – Personal and File-based Geodatabase

DATASET

Required/Optional: *Required*

For Personal Geodatabase, this is the MS Access file to which data is to be written.

For File-based Geodatabase, this is the directory ending in .gdb.

COMPRESS_AT_END

Required/Optional: *Optional*

This directive determines whether the dataset should be compressed after the writer has finished writing features to it. This directive applies to both Personal and File Geodatabases.

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_MDB_COMPRESS_AT_END YES
```

OVERWRITE_GEODB

Required/Optional: *Optional*

If set to YES, deletes the existing database. This directive applies to both Personal and File Geodatabases.

Values: YES | NO

Default Value: *NO*

Writing Subtypes and Domains

When writing subtypes, the user has two choices: use the integer code or use the code's description. If the integer code is used, then the code is set on an attribute of the same name as the subtype field. For example, if the subtype field is called `road_type`, then an attribute called `road_type` must be populated on the FME feature with the appropriate integer code. If the code's description is used instead, then the description must be supplied on a special attribute called `geodb_subtype_name`. The code corresponding to the description will then be looked up and, once found, will be inserted onto the subtype field.

Similarly, when writing domains, either the integer code may be specified or the code's description. If the integer code is used, then the code is set on an attribute of the same name as the domain field. For example, if the domain field is called `road_type`, then an attribute called `road_type` must be populated on the FME feature with the appropriate integer code. If the code's description is used instead, then the description must be supplied on an attribute called `road_type_resolved`. The code corresponding to the description will be looked up and inserted onto the domain field.

The Geodatabase writer goes through the following steps when writing a subtype:

1. Retrieve the subtype (code) attribute from the feature.
 - a. If the attribute is found but does not contain a valid subtype code (i.e. the code is not one of the possible subtypes or the code is not an integer) then an error is returned and the feature is not written.
 - b. If the attribute was not supplied on the feature, or was supplied but set to the empty string (i.e. it was set to ""), then go to step 2.
2. Retrieve the `geodb_subtype_name` attribute.
 - a. If this attribute is found and not set to the empty string, then the writer attempts to look up the code corresponding to the supplied description. If no code is found, then the description used is not valid, resulting in an error being returned and the feature not being written.
 - b. If this attribute is not supplied or was supplied but set to the empty string (i.e., set to ""), and we're inserting a new feature (not updating an existing feature), then the default code gets written to the subtype field.

Geodatabase Table Representation

The Geodatabase writer requires that every Geodatabase table to which a feature is written be defined within the FME mapping file if the table does not yet exist. If the table exists when writing, then only the table name needs to be specified on the DEF line. When reading from the Geodatabase, it is not necessary that the source tables be defined. Geodatabase tables are specified within the FME mapping file using the `<WriterKeyword>_DEF` statement. It is important to note that when using FME to define a simple table with no spatial column, the definition is merely in this form:

```
<WriterKeyword>_DEF <tableName> \
[geodb_type geodb_table] \
```

```

[<attribute name> <attribute type>]* \
[GEODB_OBJECT_ID_NAME <column_name>] \
[GEODB_OBJECT_ID_ALIAS <column_name>]

```

The more general format of a table definition – in which a spatial column can be defined (a feature class in ESRI terms) – is given here.

```

<WriterKeyword>_DEF <tableName> \
[geodb_type <geodb_type>] \
[<attribute name> <attribute type>] * \
[GEODB_UPDATE_KEY_COLUMNS <list of column names>] \
[GEODB_DROP_TABLE < YES | NO >] \
[GEODB_TRUNCATE_TABLE < YES | NO >] \
[GEODB_OBJECT_ID_NAME <column_name>] \
[GEODB_OBJECT_ID_ALIAS <column_name>] \
[ GEODB_SHAPE_NAME <column_name> \
  GEODB_SHAPE_ALIAS <column_name> \
  GEODB_FEATURE_DATASET <feature_dataset_name>] \
GEODB_GRID{1} <grid1size> \
[GEODB_GRID{2} <grid2size>] \
[GEODB_GRID{3} <grid3size>] \
[GEODB_GRID{n} <gridnsize>] \
[GEODB_AVG_NUM_POINTS <num_points>] \
[GEODB_XORIGIN <minimum_x>] \
[GEODB_YORIGIN <minimum_y>] \
[GEODB_XYSCALE <scale>] \
[GEODB_HAS_Z_VALUES < YES | NO >] \
[GEODB_ZORIGIN <minimum_z>] \
[GEODB_ZSCALE <scale>] \
[GEODB_HAS_MEASURES < YES | NO >] \
[GEODB_MEASURES_ORIGIN <measures_origin>] \
[GEODB_MEASURES_SCALE <measures_scale>] \
[GEODB_ANNO_REFERENCE_SCALE <anno_ref_scale>]
]

```

Note that the OBJECTID column and SHAPE column precede all user-defined columns in a new table or feature class created by the Geodatabase writer.

<tableName>

This specifies the name of the Geodatabase table being defined by the <WriterKeyword>_DEF statement. The name must conform to the conventions and restrictions of the underlying RDBMS database. For example, the name cannot have a hyphen (-) in it. Table name case is always preserved.

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

```

GEODATABASE_SDE_DEF roads . . .

```

geodb_type <geodb_type>

When the Geodatabase writer creates a new table, it looks for the geodb_type specified on the DEF line. For a list of all the valid geodb_type's possible, see *Feature Representation* on page 729. If the geodb_type is not valid or is not found on the DEF line, then

the `geodb_type` attribute will be retrieved from the first feature headed for that table. In the unlikely occurrence that the feature does not have a valid `geodb_type` or any `geodb_type`, then a warning will be logged, the writer will assume that the `geodb_type` is `geodb_table`, and a non-spatial table will be created. As a result, all features of that feature type will have their geometry ignored.

Attribute Definitions

This section of the `<WriterKeyword>_DEF` statement defines the attributes for a table.

- The `<attribute name>` specified within the FME mapping file must obey the following rules:
 - Attribute Name case must conform to the conventions and restrictions of the underlying RDBMS database. When writing to a Personal or File Geodatabase, the case is preserved, but when writing to an Enterprise Geodatabase whether or not the case is preserved depends on the underlying database (i.e., in Oracle, attribute names are made uppercase, whereas in Microsoft SQL Server at ArcSDE 9.2 or later, case is preserved).
 - Attribute Names must obey all length and character restrictions of the Geodatabase. There is a limit of 30 characters when writing to Geodatabase.

Note: If a table is being created and one of the attribute names conflicts with a Geodatabase system field (i.e., the object ID or one of the shape fields), then a different name will be selected for the system field and a warning message logged.

- The `<attribute name>` definition defines the type and has the form `<attribute name> <attribute type>`

The supported attribute types are listed in the following table.

FME Attribute Type
smallint
smallint(n)
integer
integer(n)
float
float(n,m)
double
double(n,m)
boolean
char(n)
date
subtype
subtype_codes
range_domain

FME Attribute Type

coded_domain

smallint

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

smallint(n)

This type is used to represent small integer values with less than or equal to n digits.

If an invalid width is specified for this data type then FME will correct the value and output a warning saying that it has done so. A valid value for the width lies between 1 and 5 inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular smallint in the other Geodatabases.

integer

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

integer(n)

This type is used to represent integer values with less than or equal to n digits.

If an invalid width is specified for this data type then FME will correct the value and output a warning saying that it has done so. A valid value for the width lies between 1 and 10 inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular integer in the other Geodatabases.

float

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

float(n,m)

This type is used to represent float values with a precision not exceeding n and a scale not exceeding m.

If an invalid width and/or decimal is specified for this data type then FME will correct the value(s) and output a warning saying that it has done so. A valid value for the width lies between 1 and 6 inclusive. A valid value for the decimal lies between 0 and the value of the width inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular float in the other Geodatabases.

double

This type is used to represent 64-bit float values.

double(n,m)

This type is used to represent double values with a precision not exceeding *n* and a scale not exceeding *m*.

If an invalid width and/or decimal is specified for this data type then FME will correct the value(s) and output a warning saying that it has done so. A valid value for the width lies between 1 and 38 inclusive. A valid value for the decimal lies between 0 and the value of the width inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular float in the other Geodatabases.

boolean

This type is used to represent Boolean values. The possible values are `true` and `false`.

char(n)

This type is used to represent character values with a length not exceeding *n* characters. The FME will read from and write to geodatabases using the UTF-16 encoding.

date

This is used to store and retrieve date information within the Geodatabase.

When a date field is read by the Geodatabase, it is placed in the FME feature with the form `HHMMSS`, `YYYYMMDD`, or `YYYYMMDDHHMMSS`. The time portion is specified using the 24-hour clock. When writing, the date attribute must also be in one of these three forms. These forms are compatible with all other FME dates.

Note: When the Geodatabase writer creates a new table, all the fields, except for the object ID field, will be defined as allowing NULL values.

subtypes

Subtypes allow you to define a subclassification of a table based on a field. For instance, a table named `road` may have a field called `condition` which can have values `good`, `bad` and `miserable`. In the Geodatabase, the field must be an integer type of some sort in order to be able to create subtypes on it.

For the Geodatabase writer, subtypes can be created when creating a new table using the following syntax on `DEF` lines

```
<attribute name> subtype(value1:value2:value3:.....valuen)
```

or

```
<attribute name> subtype_codes(code1:val1:code2:val2:....coden:valn)
```

Note: The argument list (i.e., everything between the parentheses) must be colon-separated and must be encoded using a special XML-like encoding. Workbench automatically encodes the list properly. To encode the argument list manually within a mapping file or FME Objects, see *Substituting Strings in Mapping Files* in the FME Fundamentals on-line help file. You can also contact Safe Software for assistance.

In the first case, descriptions can be supplied as strings, in which case codes are generated by the Geodatabase writer starting at zero. In the second case, the input list consists of pairs of codes and corresponding descriptions.

The first code in the list will be used as the default subtype code. In the first case where only descriptions are specified, the code created for `value1` will be used as the default subtype code. For instance, if the `DEF` line is specified as follows

```
subtype_codes(1:a:3:b:4:c:5:d)
```

then 1 will be used as the default code (which maps to a).

The following restrictions are applied when subtypes are created:

- Each table can have only one subtype.
- All the codes have to be unique and valid integers.
- All the `value,description` pairs have to be unique.
- You cannot add subtypes to an existing table. If you do, the `DEF` line definition will be ignored and the table will use the existing subtype, if one exists. If a subtype does exist on the table, then a comparison will be made between the `DEF` line definition and the existing subtype's definition. A warning message will be logged if they are different.

When writing features, the subtype attribute must contain the code (which is stored as an integer by Geodatabase). To supply the description instead of the code for a feature, use the special attribute `geodb_subtype_name`. If the subtype attribute is not specified, then the writer will look for the `geodb_subtype_name` attribute and will convert the description to its corresponding code. See the section *Writing Subtypes and Domains* on page 713 for more information.

Example:

```
<writerKeyword>_DEF road \
  type geodb_polyline \
  condition subtype_codes(0:good:1:bad:2:miserable) \
  ..... \
  .....
```

domains

The domains can be specified at the `DEF` lines using the following syntax:

```
<attribute name> range_domain(domain_name:field_type:min_val: max_val)
<attribute name> coded_domain(domain_name:field_type:name_1:value_1: ....
                             name_n:value_n)
```

This syntax is used to create new domains along with the new field. The domains defined as above are added to both the workspace and the field of the table. If the domain

name already exists in the workspace, a comparison is made against the existing definition. A warning is issued if they differ, and the existing domain is used.

Note: The argument list (i.e., everything between the parentheses) must be colon-separated and must be encoded using a special XML-like encoding. Workbench automatically encodes the list properly. To encode the argument list manually within a mapping file or FME Objects, see *Substituting Strings in Mapping Files* in the FME Fundamentals on-line help file. You can also contact Safe Software for assistance.

Example 1:

To define a domain with the following code and values for a float field type:

Code	Values
1.2	val1
3.6	val2,val3
4.5	test "quotes" here

you would use the following format:

```
coded_domain(floatCodedDom:float:1.2:val1:3.6:val2<comma>val3:4.5:
<quote>test<space><quote><quote>quotes<quote><quote><space>here<quote>)
```

To use an existing domain, you can use the following short syntax:

```
range_domain(domain_name)
coded_domain(domain_name)
```

An error will be generated if the domain does not exist in the workspace or is not defined elsewhere using the long syntax (i.e., the syntax specified at the beginning of this section) for domains.

Note: If the same new domain is referenced multiple times in the translation, the long syntax form only needs to be specified on one attribute; all other instances can use the short form. It does not matter on which attribute the long form is specified within the mapping file or workspace.

To specify the length of a text field while specifying an existing domain, you can use the following variant of the short syntax:

```
coded_domain(domain_name:char<openparen><textsize><closeparen>)
```

Note: The <openparen> and <closeparen> are the result of applying FME's XML-like way of escaping certain characters. Contact Safe Software if more information is needed, keeping in mind that Workbench automatically performs this encoding when creating new domains and subtypes.

For example,

```
coded_domain(textDomain:char<openparen>50<closeparen>)
```

It is important to note:

- The variant of this short syntax is allowed for text fields only.
- Since ArcGIS allows text fields to have only coded value domains, using this field type with `range_domain` will result in an error.
- If no text length is specified for a text field, that is, if the syntax used is

```
coded_domain(domain_name)
```

then FME will insert a default length of 254 characters.

Example 2:

A complete DEF line example is shown below:

```

GEODATABASE_MDB_OUT_DEF Q1_LINE \
    geodb_type          geodb_polyline \
    GEODB_DROP_TABLE YES \
    GEODE_TRUNCATE_TABLE NO \
    GEODE_OBJECT_ID_NAME Object_ID \
    GEODB_OBJECT_ID_NAME Object_ID \
    GEODE_OBJECT_ID_ALIAS "Object ID" \
    GEODE_SHAPE_NAME Shape \
    GEODB_SHAPE_ALIAS Shape \
    GEODE_CONFIG_KEYWORD DEFAULTS \
    GEODE_FEATURE_DATASET "" \
    GEODB_GRID{1}      $_GEODBOutGrid1 \
    GEODE_AVG_NUM_POINTS "" \
    GEODE_HAS_MEASURES NO \
    GEODB_HAS_Z_VALUES $_GEODBOutDimension \
    GEODE_XORIGIN "" \
    GEODE_YORIGIN "" \
    GEODE_ZORIGIN "" \
    GEODE_XYSCALE "" \
    GEODE_ZSCALE "" \
    GEODB_ANNO_REFERENCE_SCALE "" \
    geodb_oid          integer \
    igds_class          double \
    igds_color          double \
    igds_graphic_group double \
    igds_style          double \
    igds_weight         double \
    Object_ID          integer \
    Shape_Length       double \
    property            range_domain(intDomain:integer:0:100) \
    testVal            range_domain(realDomain:double:0.5:25.5) \
    textVal            coded_domain(textVal:char<openparen>50<closeparen>:a:all:b:bad) \
    floatVal           coded_domain(floatCodedDom:float:1.2:val1:3.6:val2<comma>val3:
    4.5:<quote>test<space><quote><quote>quotes<quote><quote><space>here<quote>)

```

Configuration Parameters

There are a number of configuration parameters in the GEODATABASE_<SDE|MDB|FILE>_DEF line that are used to define characteristics of a feature class. They are described in the following table. **The configuration parameters are only needed when defining a new table.** If the table already exists, it is sufficient to have the following line:

```
<WriterKeyword>_DEF <tableName>
```

When creating a new table, configuration parameters found on the table definition will override the equivalent writer directives. However, the parameters related to setting grid sizes and x,y,z origins and scales will override the writer direc-

tives only if the configuration parameter `GEODB_XYSCALE` is specified and not equal to zero.

Note: It is important to understand that the values from the settings box are assigned to the writer directives, not to the configuration parameters described in this section. This was done to prevent the problem whereby the values for the writer directives get changed in the workspace/mapping file, but never get used because they are overridden by the corresponding configuration parameters.

Parameter	Contents
<code>geodb_type</code>	A valid <code>geodb_type</code> must be specified in order to determine what type of table to create. Please check the section <i>Feature Representation</i> on page 729 for valid values of <code>geodb_type</code> . On DEF lines, setting the <code>geodb_type</code> to <code>geodb_arc</code> is equivalent to <code>geodb_polyline</code> and setting it to <code>geodb_ellipse</code> is equivalent to <code>geodb_polygon</code> .
<code>GEODB_UPDATE_KEY_COLUMNS</code>	<p>The list of field names that are used by the writer when it is updating or deleting a feature. The value is a comma-separated list of the fields which are matched against the corresponding FME attributes' values to specify which rows are to be updated using the other attribute values. If a corresponding attribute is not on the FME feature, then the value NULL will be used in the query for that particular column.</p> <p>In general, this should identify a unique feature but can also be used to update/delete multiple features if desired. If an update/delete is being performed and no value is assigned to this parameter, then the writer will use the object ID column to perform the update and the corresponding object ID attribute must be present on the FME feature.</p> <p>The following example sets the update fields to be <code>COUNTRY</code> and <code>CAPITAL</code>:</p> <pre>GEODB_UPDATE_KEY_COLUMNS COUNTRY,CAPITAL</pre>
<code>GEODB_DROP_TABLE</code>	<p>Specifies that the writer drop the table before writing, and create a new one. If the table does not exist, it will be created when the data is written.</p> <p>The following example sets the drop table flag to false.</p> <pre>GEODB_DROP_TABLE NO</pre> <p>Default: NO Values: YES NO</p>
<code>GEODB_TRUNCATE_TABLE</code>	<p>Specifies that the writer truncate the table before writing. If the table does not exist, it will be created when the data is written. The following example sets the truncate table flag to false.</p> <pre>GEODB_TRUNCATE_TABLE NO</pre> <p>Default: NO Values: YES NO</p>

Parameter	Contents
GEODB_OBJECT_ID_NAME	<p>The name of the column containing object IDs for the current table. The name of the column must not contain any spaces. If this parameter is not found on the DEF line, then the column will be given the name <code>Object_ID</code> (Universal Translator) or <code>OBJECTID</code> (Workbench). If the value conflicts with a user attribute, then the writer will change the value for this field (by appending a numeric suffix) and log a warning.</p> <p>Note: Due to the way annotation feature classes are created since the inception of ArcGIS 9.0, this parameter has no effect when run on computers with ArcGIS 9.0 or newer.</p> <p>The following example defines the column name to hold object IDs to be <code>OBJECT_IDENT</code>:</p> <pre>GEODB_OBJECT_ID_NAME OBJECT_IDENT</pre>
GEODB_OBJECT_ID_ALIAS	<p>The alias for the object IDs column for the current table. The alias is used in ArcMap (and possibly in other ArcGIS products) when viewing data, the object ID column will be labeled by its alias. If this parameter is not found on the DEF line, then the alias will be given the value <code>Object ID</code> (Universal Translator) or <code>OBJECTID</code> (Workbench).</p> <p>The following example defines the alias for the object ID column name to be <code>Primary ID</code>. Notice the alias name must be surrounded by quotation marks ("").</p> <pre>GEODB_OBJECT_ID_ALIAS "Primary ID"</pre>
GEODB_SHAPE_NAME	<p>The name of the column containing the shape data for features in the current feature class. This applies only to feature classes. The name of the column must not contain any spaces. If this parameter is not found on the DEF line, then the column will be given the name <code>Shape</code> (Universal Translator) or <code>SHAPE</code> (Workbench). If the value, or one of its corresponding <code>LENGTH</code> or <code>AREA</code> fields, conflicts with a user attribute, then the writer will change the value for this field (by appending a numeric suffix) and log a warning.</p> <p>Note: Due to the way annotation feature classes are created since the inception of ArcGIS 9.0, this parameter has no effect when run on computers with ArcGIS 9.0 or newer.</p> <p>The following example defines the shape data column name to be <code>Geometry</code>:</p> <pre>GEODB_SHAPE_NAME Geometry</pre>
GEODB_SHAPE_ALIAS	<p>The alias for the shape data column. When viewing data in ArcMap (and possibly in other ArcGIS products), the shape data column will be labeled by its alias. If this parameter is not found on the DEF line, then the alias will be given the value <code>Shape</code> (Universal Translator) or <code>SHAPE</code> (Workbench).</p> <p>The following example defines the alias for the shape data column name to be <code>"Shape Geometry"</code>. (Note that the alias name must be enclosed in quotation marks.)</p> <pre>GEODB_SHAPE_ALIAS "Shape Geometry"</pre>

Parameter	Contents
GEODB_FEATURE_DATASET	<p>The name of the feature dataset to which a feature class belongs. If this parameter is not specified on the DEF line, then the feature class is created as a standalone feature class. If this parameter is specified, then the feature class will be made part of the specified feature dataset, and if that dataset does not exist then it will be created. If the feature dataset is created by FME, then the values for the false origin and scale are taken from the respective writer directives (that is, the X_ORIGIN, Y_ORIGIN, Z_ORIGIN, XY_SCALE, and Z_SCALE). If a directive is not specified, then the default value for that directive is used. However, when writing to a File-based Geodatabase, these directives do not get used. Instead, default values, based on the coordinate system set, are used for the origin and resolution. The following example specifies that the feature class belongs to a feature dataset named <code>Town</code>:</p> <pre data-bbox="699 661 1114 682">GEODB_FEATURE_DATASET Town</pre>
GEODB_GRID{1}	<p>This parameter specifies the size of the spatial index in the coordinate system of the layer. It is only applicable if a feature class is being created and is only used if the <code>GEODB_XYSCALE</code> parameter contains a non-zero value. The value must be a real number greater than zero.</p> <p>When using the Enterprise Geodatabase or File-based Geodatabase writer, if the value is 0 or is not specified, and the value for the <code>GRID_1</code> directive is 0 then the grid size will be automatically calculated. The File-based Geodatabase writer will also automatically calculate sizes for grids 2 & 3.</p> <p>The following example defines a grid size of 200 for the level 1 grid:</p> <pre data-bbox="664 1123 935 1144">GEODB_GRID{1} 200</pre>
GEODB_GRID{2}	<p>This optional parameter defines the level 2 grid size. If it is not desired, then the value should not be specified or should be set to 0. Before the level 2 grid size can be specified, the level 1 grid size must be specified. This parameter must be a real number greater than zero.</p> <p>The following example defines a grid size of 600 for the level 2 grid:</p> <pre data-bbox="664 1375 935 1396">GEODB_GRID{2} 600</pre>
GEODB_GRID{3}	<p>This optional parameter defines the level 3 grid element size. If it is not desired, then the value should not be specified or should be set to 0. Before the level 3 grid size can be specified, the level 2 grid size must be specified. This parameter must be a real number greater than zero.</p> <p>The following example defines a grid size of 1800 for the level 3 grid:</p> <pre data-bbox="664 1627 951 1648">GEODB_GRID{3} 1800</pre>

Parameter	Contents
GEODB_AVG_Num_points	<p>This optional field specifies the estimated average number of points per feature. It is used in the creation of the spatial index for the feature class. If this configuration parameter is not specified, then a default value will be assigned to the feature class, depending on its geometry type. If the geometry is a point the default value will be 1; if the geometry is a multipoint, the default will be 10; if the geometry is a polyline, the default value will be 20; if the geometry is a polygon, the default value will be 40. This parameter must be an integer.</p> <p>For example, if a new feature class called <code>Roads</code> is to be created and we believe that the average number of points for a feature from the <code>Roads</code> feature class will be 3, then on the <code>DEF</code> line for the <code>Roads</code> feature class, we would find the following:</p> <pre>GEODB_AVG_NUM_POINTS 3</pre>
GEODB_CONFIG_KEYWORD	<p>This optional field can be used to specify the configuration parameters of the table to be written. This directive can be used effectively if the database is accessed through ArcSDE. It is ignored when writing to Personal database. For example</p> <pre>GEODB_CONFIG_KEYWORD TEST_CONFIG</pre> <p>The default value is <code>DEFAULTS</code>.</p>
GEODB_XORIGIN	<p>This is the same as the writer directive <code>X_ORIGIN</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. This parameter is only used if the <code>GEODB_XYSCALE</code> parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number.</p> <p>For example:</p> <pre>GEODB_XORIGIN -53040</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_YORIGIN	<p>This is the same as the writer directive <code>Y_ORIGIN</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. This parameter is only used if the <code>GEODB_XYSCALE</code> parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number.</p> <p>For example:</p> <pre>GEODB_YORIGIN 1043.89</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>

Parameter	Contents
GEODB_XYSCALE	<p>This is the same as the writer directive <code>XY_SCALE</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. If this parameter does not appear or is set to 0, all the <code>x,y,z</code> origins and scales and the grid 1 size are taken from the writer directives, even if some of these values are supplied on the <code>DEF</code> line. If this value is set to a non-zero value, then all the <code>DEF</code> line parameters for the <code>x,y,z</code> scales and origins and grid 1 size must be specified. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number greater than or equal to 0.</p> <p>For example:</p> <pre>GEODB_XYSCALE 1000</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_HAS_Z_VALUES	<p>This optional parameter specifies whether or not the features will contain <code>Z</code> values. The only valid values are <code>YES</code> and <code>NO</code>.</p> <p>For example:</p> <pre>GEODB_HAS_Z_VALUES YES</pre> <p>Because Geodatabase does not allow mixed 2D and 3D features in the same feature class, it is best to put a value of <code>YES</code> for this parameter if you have mixed dimensions. The 2D features will be forced to 3D.</p> <p>If you get an error message saying your feature class does not support <code>Z</code> values, you cannot simply add this configuration parameter to your <code>DEF</code> line and perform the translation again. Since this parameter is only used when a feature class is created, rather than an existing feature class being opened, you must either delete the existing feature class or translate to a new Geodatabase.</p> <p>If this configuration parameter is not specified, then a default value of <code>NO</code> will be assumed.</p>
GEODB_ZORIGIN	<p>This is the same as the writer directive <code>Z_ORIGIN</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. This parameter is only used if the <code>GEODB_XYSCALE</code> parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number.</p> <p>For example:</p> <pre>GEODB_ZORIGIN 0</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>

Parameter	Contents
GEODB_ZSCALE	<p>This is the same as the writer directive <code>Z_SCALE</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. This parameter is only used if the <code>GEODB_XYSCALE</code> parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number greater than 0.</p> <p>For example:</p> <pre>GEODB_ZSCALE 10</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_HAS_MEASURES	<p>This optional field specifies whether or not the features will contain measures. The only valid values are <code>YES</code> and <code>NO</code>. If this configuration parameter is not specified, then a default value of <code>NO</code> will be assumed. For example:</p> <pre>GEODB_HAS_MEASURES YES</pre>
GEODB_MEASURES_ORIGIN	<p>This is the same as the writer directive <code>MEASURES_ORIGIN</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. If this parameter does not appear, then the value for the directive <code>MEASURES_ORIGIN</code> is taken. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. This parameter must be a real number.</p> <p>For example:</p> <pre>GEODB_MEASURES_ORIGIN -232</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_MEASURES_SCALE	<p>This is the same as the writer directive <code>MEASURES_SCALE</code> above, except this configuration parameter applies only to the table whose <code>DEF</code> line it is on. If this parameter does not appear, then the value for the directive <code>MEASURES_SCALE</code> is taken. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. This parameter must be a real number greater than zero.</p> <p>For example:</p> <pre>GEODB_MEASURES_SCALE 5489.6</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>

Parameter	Contents
GEODB_ANNO_REFERENCE_SCALE	<p>This optional field specifies what reference scale to use when creating an annotation feature class. The reference scale determines the scale at which the text's size, on the screen, is the size indicated on each annotation feature. When the scale is larger in value than the reference scale, then the text appears smaller than that indicated on the annotation feature and vice versa. If no value is specified for this field, then FME uses first annotation feature for the annotation feature class. If the feature contains the attribute <code>geodb_text_ref_scale</code> then the value for the attribute is used as the reference scale. If the attribute doesn't exist, then the default value for the attribute is used, which is 1. For example:</p> <pre data-bbox="664 573 1170 598">GEODB_ANNO_REFERENCE_SCALE 12000</pre>

Creating Dimension Feature Classes

Currently, there is no way to specify any dimension-specific settings when using the Geodatabase writer to create a dimension feature class. As a result, default settings are used. The default dimension style created by the writer is equivalent to the default one ArcCatalog creates. Likewise, the reference scale and map units are set to the default values used by ArcCatalog, 1 and decimal degrees, respectively. If the dimension feature class is created within a feature dataset, it will inherit the units of the feature dataset rather than being set to decimal degrees.

If the default settings are not sufficient, the dimension feature class should be created before the translation using ArcCatalog. ArcCatalog makes it easy to import dimension styles from existing feature classes and then change them as required.

Improving the Speed of Translations Using the Geodatabase Writer

You can speed up translations involving the Geodatabase writer by lengthening the interval between committing transactions. Committing transactions is an expensive operation and therefore it is recommended that you make the transaction interval as big as possible (or alternatively, if transactions are not needed, then you can turn them off). In speed tests performed at Safe Software Inc., changing the transaction interval from 500 (the default) to 1000 resulted in a specific translation being 2.5% faster. Changing the transaction interval to 5000 resulted in the same translation running 5.5% faster. Turning off transactions resulted in an improvement of either 12% or 19%. The performance advantages of changing the transaction interval or of turning transactions off will differ between various datasets.

Another way in which the speed of writing features can be increased is by creating all the feature datasets, feature classes, and non-spatial tables ahead of time so that the Geodatabase writer only needs to open them, rather than create them.

Tips for Using the Geodatabase Writer

- When writing to a Geodatabase, those fields that are not assigned values (i.e., no attribute exists on the feature for that field) will be assigned a NULL value if the field allows NULL values.

- When writing to Geodatabase, if your translation fails with the ArcObjects message number of -2147216072 (FDO_E_SE_DB_IO_ERROR), there may be several reasons for this, such as:
 - The database that you are writing to may be out of space.
 - One of the values that you are trying to insert is too large for the underlying database to handle.
 - One of the values is too large for the data type specified. Try changing the data types of your columns.
 - One of your column names is invalid, possibly due to characters that are not considered capitalized. Try renaming the faulty column.

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.

The Geodatabase modules make use of the following special attribute names.

Attribute Name	Contents
geodb_type	<p>The type of geometric entity stored within the feature. The valid values are listed below:</p> <ul style="list-style-type: none"> geodb_table geodb_point geodb_multipoint geodb_polyline geodb_arc geodb_ellipse geodb_polygon geodb_multipatch geodb_annotation geodb_dimension geodb_simple_junction geodb_simple_edge geodb_complex_junction (read-only) geodb_complex_edge geodb_metadata geodb_attributed_relationship (read-only)
geodb_measures	<p>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 that correspond to the vertex measures. The first value is for the first vertex, second for the second, and so on.</p>
geodb_feature_is_simple (Reader only)	<p>Indicates whether or not the geometry is simple. Only present on spatial features and when ArcGIS 9 is installed.</p>
geodb_subtype_name	<p>When reading, if RESOLVE_SUBTYPE_NAMES is set to YES, then the value corresponding to the subtype code is stored in this attribute. When writing to a table with subtypes and no integer value is supplied for the subtype field, then specifying this attribute with an actual value (i.e. not "") will trigger a look-up for the code corresponding to the value supplied in this attribute. If the code is found, it will be written to the subtype field; otherwise the feature will fail to be written. If the value specified was the empty string (i.e. "") then the default value will get used.</p>

Attribute Name	Contents
<attribute-name>_resolved	When reading, if RESOLVE_DOMAINS is set to YES, then the description corresponding to the domain code is stored in this attribute. When writing to a field associated with a coded value domain, specifying this attribute instead of <attribute-name> will trigger a look-up for the corresponding code. If the code is found, it will be written to <attribute-name>.

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

Tables

geodb_type: geodb_table

Features with this value consist of no coordinates. This value is used by both the reader and the writer.

Points

geodb_type: geodb_point

Features with this value are point features. This value is used by both the reader and the writer.

Multipoints

geodb_type: geodb_multipoint

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

Note: If a multipoint feature is written to an existing point feature class, then the feature will be split up and each point written out as a separate feature. Each new feature will have the same attribution as the original feature; the only difference will be the geometry. If a multipoint feature is written to a point feature class that has not yet been created, then a multipoint feature class will be created instead of a point feature class.

Polylines

geodb_type: geodb_polyline

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

Arcs

geodb_type: geodb_arc

Features with this value contain either a circular arc or an elliptical arc. Arc features are like ellipse features, except two additional angles control the portion of the ellipse boundary which is drawn.

Tip: The Function `@Arc()` can be used to convert an arc to a line. This is useful for representing arcs in systems that do not support them directly.

Note: During reading, if an arc is encountered that has become a line, FME will create a line feature out of it rather than creating an arc feature and its `geodb_type` will be set to `geodb_polyline` instead of `geodb_arc`.

Attribute Name	Contents
<code>geodb_primary_axis</code>	The length of the semi-major axis in ground units. If the arc is circular, this will be the same as the <code>geodb_secondary_axis</code> .
<code>geodb_secondary_axis</code>	The length of the semi-minor axis in ground units. If the arc is circular, this will be the same as the <code>geodb_primary_axis</code> .
<code>geodb_start_angle</code>	Refer to the <code>@Arc</code> (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of <code>start_angle</code> . Default: 0
<code>geodb_sweep_angle</code>	Refer to the <code>@Arc</code> (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of <code>sweep_angle</code> .
<code>geodb_rotation</code>	The rotation of the major axis. The rotation is measured in degrees counter clockwise up from horizontal. Default: 0
<code>geodb_arc_start_x</code>	The x coordinate of the start point of the arc. This attribute is only present on classic geometry. On rich geometry, this information is stored within the geometry itself. Available only with classic geometry.
<code>geodb_arc_start_y</code>	The y coordinate of the start point of the arc. On rich geometry, this information is stored within the geometry itself. Available only with classic geometry.
<code>geodb_arc_start_z</code>	The z coordinate of the start point of the arc. Only applicable when dealing with 3D arcs. On rich geometry, this information is stored within the geometry itself. Available only with classic geometry.
<code>geodb_arc_end_x</code>	The x coordinate of the end point of the arc. On rich geometry, this information is stored within the geometry itself. Available only with classic geometry.

Attribute Name	Contents
geodb_arc_end_y	The y coordinate of the end point of the arc. On rich geometry, this information is stored within the geometry itself.
Available only with classic geometry.	
geodb_arc_end_z	The z coordinate of the end point of the arc. Only applicable when dealing with 3D arcs. On rich geometry, this information is stored within the geometry itself.
Available only with classic geometry.	
geodb_original_arc_direction	Always used in conjunction with the start and end points of the arc. This attribute indicates whether the arc goes clockwise beginning from the start point, or counterclockwise.
Available only with classic geometry. Default: counterclockwise	

Ellipses

geodb_type: geodb_ellipse

Ellipse features are point features used to represent both circles and ellipses. The point serves as the centre of the ellipse.

Tip: The Function @Arc() can be used to convert an ellipse to a polygon. This is useful for representing ellipses in systems that do not support them directly.

Attribute Name	Content
geodb_primary_axis	The length of the semi-major axis in ground units.
geodb_secondary_axis	The length of the semi-minor axis in ground units. Default: the value of geodb_primary_axis
geodb_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Default: 0

Polygon

geodb_type: geodb_polygon

Features with this value are features or multi-part features consisting of polygons and/or donut polygons.

Multipatch

geodb_type: geodb_multipatch

Features with this value consist of a 3D geometry, used to represent the outer surface of features which occupy a discrete area or volume in three-dimensional space. Geodatabases directly support 3D polygonal faces, triangle fans, triangle patches and triangle strips. By definition, the surfaces which compose a multipatch do not need to be connected. This value is used by both the reader and the writer.

For writing, all 3D geometry types are supported. Any types of 3D geometry which are not directly supported in a geodatabase (e.g., solids) are decomposed into a set of 3D polygonal faces prior to writing.

Annotation

geodb_type: geodb_annotation

Feature-linked Annotation

Annotations are separate features but can be linked to other features through feature-linked annotations. Feature-linking occurs when there is a relationship between an annotation feature class and some other feature class. The attribute `geodb_linked_feature_id` controls which annotations are linked to which features.

Note: Feature-linked annotations can be read and written using FME if the necessary feature classes and relationships are created and set up before the translation. Currently, it is only possible to read or insert feature-linked annotations, not update or delete them.

If the feature to be linked to by the annotation has not yet been written, then it is possible for the Geodatabase writer to write the feature, retrieve the object ID of the new feature, and then write the annotation feature linking to it, supplying the correct value for `geodb_linked_feature_id`. This is accomplished by supplying attributes (Geodatabase-specific annotation attributes and user-defined attributes) from the annotation feature on the non-annotation feature; of the annotation feature attributes, only `geodb_text_string` must be specified. In addition, the following attributes must also be specified:

1. `geodb_text_feat_class_name` - specifies the annotation feature class to which the annotation will be written
2. `geodb_text_x_coord` & `geodb_text_y_coord` - specifies the location of the annotation

The result is that the one FME feature contains enough information to write two features: one annotation feature and one non-annotation feature.

To specify multiple feature-linked annotations, the list attribute `geodb_text{i}` must be used to group together each annotation's attributes. There are 4 mandatory annotation attributes that must be present for each annotation in the list:

1. `geodb_text{i}.geodb_text_string` - the text string to write
2. `geodb_text{i}.geodb_text_feat_class_name` - the name of the destination annotation feature class
3. `geodb_text{i}.geodb_text_x_coord` - the x coordinate of the annotation's location
4. `geodb_text{i}.geodb_text_y_coord` - the y coordinate of the annotation's location

Other than these 4 attributes, as many or as few annotation attributes can be used. Keep in mind that default values will be used for attributes that are not specified. A list of all available annotation attributes is given at the end of this section on annotations.

For example, to insert 3 annotation features, each belonging to a different anno feature class, for the one 'streets' feature written, a feature like this would be needed:

```

feature type: streets
fme_geometry = fme_line
fme_type = fme_line
geodb_type = geodb_polyline
user_defined_field_1 = value
user_defined_field_2 = value
user_defined_field_3 = value
...
geodb_text{0}.geodb_text_feat_class_name = street_names
geodb_text{0}.geodb_text_x_coord = 7504799.45082186
geodb_text{0}.geodb_text_y_coord = 731099.587626632
geodb_text{0}.geodb_text_angle = 10
geodb_text{0}.geodb_text_line_spacing = 0
geodb_text{0}.geodb_text_ref_scale = 100
geodb_text{0}.geodb_text_scale = true
geodb_text{0}.geodb_text_size = 8.2
geodb_text{0}.geodb_text_string = "displayed street name"
geodb_text{0}.user_field_1_for_street_names = value
geodb_text{0}.user_field_2_for_street_names = value
...
geodb_text{1}.geodb_font_bold = false
geodb_text{1}.geodb_font_charset = 0
geodb_text{1}.geodb_font_italic = false
geodb_text{1}.geodb_font_name = Arial
geodb_text{1}.geodb_font_strikethrough = false
geodb_text{1}.geodb_font_underline = false
geodb_text{1}.geodb_font_weight = 400
geodb_text{1}.geodb_text_feat_class_name = alternate_street_names
geodb_text{1}.geodb_text_x_coord = 7504783.11300916
geodb_text{1}.geodb_text_y_coord = 731109.158628889
geodb_text{1}.geodb_text_angle = 56.8
geodb_text{1}.geodb_text_size = 4.3
geodb_text{1}.geodb_text_string = "alternate street name"
geodb_text{1}.user_field_1_for_alternate_street_names = value
...
geodb_text{2}.geodb_text_feat_class_name = old_street_names
geodb_text{2}.geodb_text_string = "old street name"
geodb_text{2}.geodb_text_x_coord = 7504788.43294883
geodb_text{2}.geodb_text_y_coord = 731105.044247817
geodb_text{2}.user_field_1_for_old_street_names = value
...
Geometry Type: Line (2)

Number of Coordinates: 4 -- Coordinate Dimension: 2 -- Coordinate System:
`_FME_0'

(7504779.48166667,731111.522380952)
(7504797.35380952,731098.524285714)
(7504813.13690476,731104.327142857)
(7504805.59357143,731109.201190476)

```

Lastly, the TRANSACTION_TYPE directive must be set to EDIT_SESSION or VERSIONING whenever writing feature-linked annotations.

Annotation Attributes

The following attributes are used to store the annotation information within an FME annotation feature. In ArcGIS 9.1, the schema of annotation feature classes changed and a considerable number of additional fields were added. These new fields contain information about the annotation such as its font, size, angle, offset, leading, etc. When writing, these fields should **never** be set directly; instead the attributes in the table below must be used to control the properties of the annotation. After the translation, the fields will display the desired values because the attributes below correspond to some of the fields.

Attribute Name	Contents
geodb_text_string	The annotation string. It is returned as a UTF-16 encoded string by the reader. The writer converts the value supplied for this attribute into UTF-16.
geodb_text_size	The size of the text in user units. This size gets converted to points, and the text will be displayed at this size when viewed at the reference scale. If this attribute is not supplied, then a default text size of 10 points is used and no conversion is done. Default: 10.0 points
geodb_text_feat_class_name	The name of the destination annotation feature class. It is only used when writing feature-linked annotations.
geodb_text_x_coord	The x coordinate of the annotation's location. It is only used when writing feature-linked annotations.
geodb_text_y_coord	The y coordinate of the annotation's location. It is only used when writing feature-linked annotations.
geodb_linked_feature_id	The ID of the feature to which the annotation is linked. Only applicable when writing feature-linked annotation and when the feature being linked to has already been written. In most cases the non-annotation feature will not yet have been written and so the feature ID will not be known. In this case, the Geodatabase writer can provide the correct value for this attribute. See the section <i>Feature-linked Annotation</i> on page 733 for more information. Default: -1

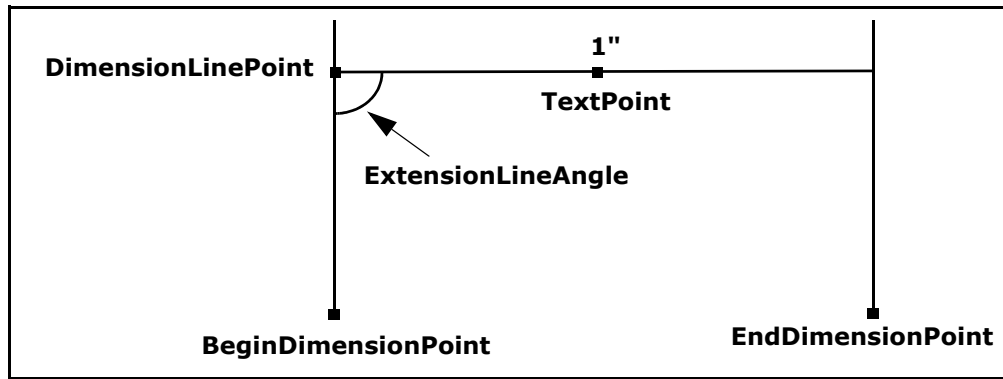
Attribute Name	Contents
geodb_anno_class_id	<p>The ID of the Annotation Class to use when writing the annotation. The ID is an integer: to see what ID an Annotation Class maps to, view the Subtypes tab of the Feature Class Properties dialog in ESRI ArcCatalog® for the annotation feature class. If an invalid ID or -1 was specified then the annotation will be created using an inline text symbol; otherwise the annotation will reference an existing text symbol. Referencing existing text symbols decreases table size and may improve performance.</p> <p>When using an Annotation Class, certain annotation properties can be overridden and others cannot. The following attributes cannot override the properties of the Annotation Class:</p> <ul style="list-style-type: none"> - geodb_font_strikethrough - geodb_font_weight - geodb_font_charset - geodb_text_scale - geodb_text_break_char - geodb_text_clip - geodb_right_to_left <p>The following attributes can override the properties of the Annotation Class:</p> <ul style="list-style-type: none"> - geodb_text_size - geodb_text_angle - geodb_font_name - geodb_font_italic - geodb_font_underline - geodb_font_bold - geodb_text_color - geodb_color - geodb_text_x_offset - geodb_text_y_offset - geodb_h_align - geodb_v_align - geodb_text_line_spacing <p>Default: -1</p>
geodb_font_name	<p>The name of the font used to display the text string. Default: Arial</p>
geodb_font_italic	<p>Indicates whether the string should be Italicized text. Allowable values are Yes and No. Default: No</p>
geodb_font_underline	<p>Indicates whether the string should be underlined text. Allowable values are Yes and No. Default: No</p>
geodb_font_bold	<p>Indicates whether the string should be boldface text. Allowable values are Yes and No. Default: No</p>

Attribute Name	Contents
geodb_font_strikethrough	Indicates whether the string should be "strike-through" text. Allowable values are Yes and No. Default: No
geodb_font_weight	Indicates the weight of the font being used to display the string. The value must be an integer greater than or equal to zero. Default: 400
geodb_font_charset	Indicates the character set being used to display the string. The value must be the integer value associated with a specific character set. For example, the ANSI character set is given the value 0, the default character set is given the value 1, and the symbol character set is given the value 2. Some additional character sets, and their values, are: BALTIC_CHARSET 186 CHINESEBIG5_CHARSET 136 EASTEUROPE_CHARSET 238 GB2312_CHARSET 134 GREEK_CHARSET 161 HANGUL_CHARSET 129 MAC_CHARSET 77 OEM_CHARSET 255 RUSSIAN_CHARSET 204 SHIFTJIS_CHARSET 128 TURKISH_CHARSET 162 Default: 0 (ANSI character set)
geodb_color	The color of the text defined as an RGB string, with each value separated by a comma. Each value must be an integer between 0 and 255 (inclusive). Note that the Geodatabase writer has an internal attribute for color (geodb_text_color) and so the default for this attribute is only used when no value is supplied for the geodb_text_color attribute. Default: 0,0,0 (black)
geodb_text_angle	The rotation of the annotation measured from the horizontal in a counterclockwise direction. It is measured in degrees. Default: 0
geodb_text_ref_scale	The reference scale at which the text's size, on the screen, is the size indicated by geodb_text_size/fme_text_size. When the scale is larger in value than the reference scale, the text appears smaller than that indicated by geodb_text_size/fme_text_size, and vice versa. Default: 1
geodb_text_scale	Indicates whether the text scales with the map. Default: TRUE
geodb_text_break_char	The ASCII value of the character that should be interpreted as the line end. Default: 10 (the line feed character)

Attribute Name	Contents
geodb_text_clip	Indicates whether the text string will be clipped in order to fit into an envelope geometry. Default: No
geodb_text_x_offset	The text offset in the x direction, measured in points. Default: 0
geodb_text_y_offset	The text offset in the y direction, measured in points. Default: 0
geodb_text_leader_line	The geometry of the leader line associated with the annotation, if present. It will be stored in OGC WKT format.
geodb_text_leader_line_anchor_point	The geometry of the leader line anchor point associated with the annotation, if present. It will be stored in OGC WKT format.
geodb_h_align	The alignment of text horizontally if the text spans multiple lines. Options: left, right, center, full Default: left
geodb_v_align	The vertical alignment of text. Options: baseline, bottom, center, top Default: bottom
geodb_right_to_left	If TRUE , then this indicates that the text is written from right to left. If FALSE , then this indicates that the text is written from left to right. Default: FALSE
geodb_text_char_spacing	The amount of character spacing, measured as a percentage of the original character's length. A value of 0 indicates that the standard amount of character spacing, as set by ESRI, will be used. A value greater than 0 increases the amount of character spacing, whereas a value less than 0 decreases the amount of character spacing. Default: 0
geodb_text_line_spacing	The amount of line spacing, measured in font points. The value must be a real number. A value of 0 indicates that the standard amount of line spacing, as set by ESRI, will be used. A value greater than 0 increases the amount of line spacing, whereas a value less than zero decreases the amount of line spacing. If the value is small enough, the order of lines will get reversed (that is, the first line becomes the last line, the second line becomes the second last line, and so on). Default: 0

Dimensions

geodb_type: geodb_dimension



Dimension features are defined by the following attributes:

Attribute Name	Contents
geodb_dim_style_id	The numeric ID describing which style this dimension uses. When writing dimensions this attribute must be supplied on the feature and be assigned an ID for an existing style, otherwise an error will occur. Note: When translating from one Geodatabase dimension feature class to another, make sure that the destination feature class contains all the dimension styles used by the input dimension feature class. This may mean that the destination dimension feature class has to be created before the translation using ArcGIS.
geodb_dim_length	The length of the dimension. A read-only attribute.
geodb_dim_custom_length	A length specified to be displayed instead of the actual length in geodb_dim_length. This value only gets used if geodb_dim_using_custom_length is set to true. Default: 0
geodb_dim_using_custom_length	Specifies whether or not to use the custom length specified instead of the actual length. Permitted values are true and false. Default: false
geodb_dim_type	Specifies whether the dimension is linear or aligned. Please reference ESRI documentation on dimensions for specific definitions. The numbers specified follow the ESRI enumeration esriDimensionType and the values are: 0 = aligned 1 = linear Default: 0

Attribute Name	Contents
geodb_dim_line_display	<p>Specifies which dimension parts appear on the dimension line (that is, if they point inward or outward, etc.). The integer values for this parameter follow the ESRI enumeration <code>esriDimensionDisplay</code> and the values are:</p> <ul style="list-style-type: none"> 0 = Displays both dimension parts. 1 = Displays the beginning dimension part. 2 = Displays the ending dimension part. 3 = Does not display any dimension part. <p>Setting this attribute overrides the value set by the dimension style.</p>
geodb_dim_extn_line_display	<p>Specifies which dimension parts appear on the extension line. The valid values are the same as for <code>geodb_dim_line_display</code>. Setting this attribute overrides the value set by the dimension style.</p>
geodb_dim_marker_display	<p>Specifies how arrows are displayed for the dimension. The values are the same as that of <code>geodb_dim_line_display</code>, except that they apply to markers (arrows) instead of dimension parts. Setting this attribute overrides the value set by the dimension style.</p>
geodb_dim_text_angle	<p>The angle of the text displayed, in radians. From ESRI's documentation: "The <code>TextAngle</code> property will only affect the dimension if the dimension's style's text alignment property is <code>True</code> in which case the text is always parallel to the dimension line." Default: 0</p>
geodb_dim_extn_line_angle	<p>The angle (in degrees) between the dimension line and the extension line. Default: 90</p>
geodb_dim_begin_dimension_x	The X value for the Begin Dimension Point.
geodb_dim_begin_dimension_y	The Y value for the Begin Dimension Point.
geodb_dim_begin_dimension_z	The Z value for the Begin Dimension Point.
geodb_dim_end_dimension_x	The X value for the End Dimension Point.
geodb_dim_end_dimension_y	The Y value for the End Dimension Point.
geodb_dim_end_dimension_z	The Z value for the End Dimension Point.
geodb_dim_line_x	<p>The X value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.</p>
geodb_dim_line_y	<p>The Y value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.</p>

Attribute Name	Contents
geodb_dim_line_z	The Z value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.
geodb_dim_text_x	The X value for the Text Point. If the x,y,z values for the text point are all zero then the default text position is used. Default: 0
geodb_dim_text_y	The Y value for the Text Point. If the x,y,z values for the text point are all zero, then the default text position is used. Default: 0
geodb_dim_text_z	The Z value for the Text Point. If the x,y,z values for the text point are all zero, then the default text position is used. Default: 0

Simple Junctions

geodb_type: geodb_simple_junction

Simple junction features are defined by the following attributes:

Attribute Name	Contents
geodb_edge_feature_count	The number of edge features associated with the junction.
geodb_element_id	The logical network element ID of the junction.
geodb_ancillary_role	The network ancillary role of the junction. Possible values are: none, source, and sink.

Simple Edges

geodb_type: geodb_simple_edge

Currently, this type is only supported by the Reader. Simple edge features are defined by the following attributes:

Attribute Name	Contents
geodb_element_id	The logical network element ID of the junction.
geodb_from_junction_element_id	The junction element ID that corresponds to the from endpoint.
geodb_to_junction_element_id	The junction element ID that corresponds to the to endpoint.

Complex Junctions

geodb_type: geodb_complex_junction

This type is deprecated, and only supported by the Reader. Complex junction features are defined by the following attributes:

Attribute Name	Contents
geodb_junction_element_count	The number of junctions associated with the feature
geodb_edge_feature_count{}	The number of edge features associated with the indexed connection point
geodb_topological_configuration	The configuration of the feature. Possible values are: chain, loop, star, and mesh.
geodb_ancillary_role	The network ancillary role of the junction. Possible values are: none, source, and sink.
geodb_edge_element_count	The number of edge elements associated with the feature.

Complex Edges

geodb_type: geodb_complex_edge

The attributes present on an FME feature depend on the value for the reader directive SPLIT_COMPLEX_EDGES. If the value is NO, the following attributes will be present:

Attribute Name	Contents
geodb_edge_element_count	The number of edge elements associated with the feature.
geodb_from_junction_element_id	The junction element ID that corresponds to the <i>from endpoint</i> .
geodb_junction_feature_count	The number of connected junction features.
geodb_to_junction_element_id	The junction element ID that corresponds to the <i>to endpoint</i> .

If the value is YES, these attributes will be present:

Attribute Name	Contents
geodb_element_id	The element ID of the logical edge element.
geodb_element_index	An attribute created and assigned by FME. It is used to order the edge elements within a complex feature. The index begins at zero, not one.
geodb_from_junction_element_id	The junction element ID that corresponds to the from endpoint. Note: This is the <i>from endpoint</i> of the edge element, not the edge feature.

Attribute Name	Contents
geodb_to_junction_element_id	The junction element ID that corresponds to the to endpoint. Note: This is the <i>to endpoint</i> of the edge element, not the edge feature.

Metadata

geodb_type: geodb_metadata

Metadata features contain metadata about the feature type. Metadata can be read and written. The metadata used is of the same form as when using ESRI's ArcCatalog to export table metadata to (plain) XML.

When reading, the following attributes are supplied on the feature, where applicable:

Attribute Name	Contents
fme_contains_spatial_column	yes or no, depending on whether the feature type is a non-spatial table, attributed relationship, or a feature class
fme_dimension	2 or 3, depending on the dimension of the feature class
fme_feature_identifier	The name of the object ID field
fme_geometry{0}	The geometry of the feature class. This will be set to fme_no_geom for non-spatial tables and attributed relationships
fme_num_entries (Personal Geodb only)	The total number of features in the table
geodb_metadata_string	The geodatabase metadata in XML

If the feature type represents a feature class, the geometry of the metadata feature returned is a polygon, representing the extents of the feature class and the coordinate system of the feature class also gets set on the feature.

When writing, the `geodb_type` of the feature must be `geodb_metadata`; however, the `geodb_type` of the destination feature type must not be `geodb_metadata`, but rather the type of the table itself.

The metadata within `geodb_metadata_string` will overwrite any previous metadata that exists in the table. If multiple metadata features are written to a single table, then the last metadata feature will be used. Viewing the metadata within ArcCatalog after the translation will automatically update certain fields, such as table name & record count, if they were set incorrectly in `geodb_metadata_string`. However, if you use FME to read back the metadata before viewing the results in ArcCatalog then the incorrect fields will not have been corrected. None of the other attributes supplied on an FME feature when reading metadata will get used when writing metadata.

Writing metadata features does not increase the number of features in a table.

Attributed Relationship

geodb_type: geodb_attributed_relationship

Features with this value consist of no coordinates, and can be treated like `geodb_table` features.

This value is used only by the reader. To write to an attributed relationship class, the attributed relationship class must exist before the translation and the `geodb_type` of the destination feature type must be `geodb_table`.