

Intergraph GeoMedia Access and SQL Server Warehouse Reader/Writer

FORMAT NOTES:

This format is not supported by FME Base Edition.

The GeoMedia Access Warehouse Reader/Writer module provides the Feature Manipulation Engine (FME) with access to Intergraph GeoMedia Access and SQL Server Warehouses, which store both spatial and attribute data.

Note: The following information applies both to Access and SQL Server warehouse reading and writing unless otherwise specified.

FM0 Quick Facts

Format Type Identifier	FM0
Reader/Writer	Both
Licensing Level	Professional
Dependencies	Writer: GeoMedia installed
Dataset Type	File
Feature Type	Table name
Typical File Extensions	.mdb
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	fm0_type

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

FM0_SQL Quick Facts

Format Type Identifier	FM0_SQL
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	fm0_type

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

Overview

GeoMedia warehouses store both geometry and attributes for features in the form of columns within the tables of a database. Tables can be divided into two groups based on content. The first group contains meta-information about the formatting of the data, including coordinate systems, tables aliases, modification logs, a data table list and a field level index. The second group is the list of tables that actually contain the geometry features and their attributes. For example, a single Microsoft Access .mdb file or a single SQL Server database contains all the necessary information for an image.

In order to retrieve information from a GeoMedia warehouse, an Open DataBase Connectivity (ODBC) source must be set up. Depending on the source dataset format, users can specify a filename, database name or a valid existing ODBC data source name. If the source format is of type "GeoMedia Access warehouse" then either a filename or DSN can be used. If the source format is of type "GeoMedia SQL Server warehouse"

then either a database name with associated parameters or DSN can be used. Once the GeoMedia Reader has all the required information, it then dynamically creates a temporary ODBC source (when a filename or database name is supplied) to connect to that database.

Point, line, area, arc, and text primitive geometric data can be stored in the tables produced by GeoMedia, as well as composites (aggregates), boundaries (donuts) and collections (aggregates) of those types. A given data table holding multiple types of geometries can be read by the reader, but the writer only produces tables containing one specific geometry type each, including boundaries, composites or collections of that type. The result is that one table containing many types of features may be converted into several tables – one for each type of feature by the writer. This is especially true of the collection type in GeoMedia which is very generally a collection of any primitive type and has no corresponding equivalent in FME. Syntactically, the type is simply appended to the table name separated by an underscore (for example, `tableOfManyTypes_area`).

The key table for determining the names of the spatial tables is called the `GFeatures` table. This table contains the list of names of the geometry and attribute data tables. This information can also be retrieved from the `FieldLookup` table which also contains the specific fields of each geometry table. Before either of these lookups can happen, however, you have to find the `GFeatures` and `FieldLookup` tables. Table names can be aliased and there is only one table that must have a constant name (the `GAliasTable`). From here, you can look up the given names of the other metadata tables in the specific database you are viewing.

GeoMedia warehouses hold three-dimensional geometries. A geometry table may contain any mix of attributes the user has specified, but must contain a column containing the actual geometry object. This column is a blob type and is simply an encoded block of binary data. Each geometry blob type is encoded in a unique way and varies in length.

GeoMedia can store text in two variations: plain text and rich text. Since FME supports plain text only, the GeoMedia reader will convert all rich text to plain text and set the text size to either the default (1 ground unit) or to the user-supplied size in ground units using the `TEXT_SIZE_GROUND_UNITS` keyword. If the text being read is in rich text format, it also sets the attribute `fm0_rtf_text_string` to the original formatted string.

The GeoMedia writer will first check to see whether or not the attribute `fm0_rtf_text_string` is set. If it is set, then the formatted string will be used to write out as rich text. If the attribute is not set, then the GeoMedia writer by default will write plain text unless the `PLAIN_TEXT` keyword is set to `NO`. In this case, the GeoMedia writer will write rich text using either a default font size of 10 or a user-supplied font size using `FONT_SIZE` keyword. Allowed font size is between 1 to 1024 inclusive. At this time the font size for rich text is the only supported style for writing.

Reading is performed by parsing the tables and respective binary BLOBs directly from the Microsoft Access database or the SQL Server database. Therefore, the GeoMedia Access Warehouse Reader does not require GeoMedia to be installed in order to run. The GeoMedia Access Warehouse Writer, however, uses the GeoMedia COM objects to create and write the tables and BLOBs, and therefore cannot be used without a licensed GeoMedia installation. The GeoMedia SQL Server Warehouse Reader/Writer does not require the installation of GeoMedia but does require access to Microsoft SQL Server.

Coordinate system support exists for both reading and writing: in the best case, a GeoMedia warehouse that contains a defined coordinate system can be read and converted to any of the supported writer formats in FME which also support coordinate systems, with the same coordinate system name. If the coordinate system is not specifically identified to have the exact same defined name within FME, the attributes of the coordinate system are still transferred, producing an identical coordinate system in all but name. The reverse is also true: the writer can interpret any given FME coordinate system and convert it to either a named GeoMedia coordinate system or an equivalent created coordinate system.

One issue involves the type of projection used in a coordinate system definition which GeoMedia calls the Base Storage Type. This type can be set to one of projected, geographic or geocentric. Projected types are the usual case and are handled normally by FME, though it is notable that the storage center values from the GeoMedia coordinate system become built into the coordinates and are cleared in translated warehouses. All geographic types are represented in the Lat/Long coordinate system but remain identical in appearance. The storage center values here will represent how to convert the coordinates to radians since they will be provided in degrees, as is consistent with the way GeoMedia itself handles geographic types. Finally, the geocentric type is not supported by FME.

Native spatial indexing in GeoMedia is supported by both the reader and the writer. However some conditions apply. The reader will preserve spatial indexes read from a GeoMedia warehouse and a writer will automatically create new indexes (when creating new feature tables) based on the data, as long as the following is true:

The `CREATE_SPATIAL_INDEX` keyword is not set to `NO`,

The registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Intergraph\Applications` has a string value called `DefaultJCache` which must be set to the correct version of GeoMedia (for example "GeoMedia Professional_04.00"), and

The `autodt.ini` file must contain a valid datum mapping between the current coordinate system datum and the WGS84 datum. (See the file for more detail. It is probably located in a directory such as `C:\Program Files\GeoMedia Professional\Program\cssruntm\cfg\autodt.ini`).

When appending to an existing warehouse, the creation of a Spatial Index depends on whether the `SpatialKeyFieldName` property is set for the geometry column and the column itself exists. If the property is set and the column exists, then a spatial key will be automatically created for the geometry, regardless of the `CREATE_SPATIAL_INDEX` keyword setting.

For the most part, spatial index creation should happen automatically for most known coordinate systems since the default for the writer creating them is `yes`, and the registry key mentioned above should be set when GeoMedia is installed. Additionally, data tables can be created with either primary or secondary indexes by the GeoMedia Access Warehouse Writer.

When translations are run with enhanced geometry handling turned ON, it enables the Geomedia and Geomedia SQL readers to read complex geometries like paths, and enables the FME to store them. The Geomedia and Geomedia SQL writers have been enabled to write FME features with enhanced geometries as well. Altogether, the addition of the enhanced geometry model support increases accuracy of geometric

representation in Geomedia-to-Geomedia translations, as well as the creation and interpretation of more accurate features when translating to or from other FME formats

Reader Overview

The GeoMedia Warehouse Reader produces FME features for all data held in the Microsoft Access `.mdb` files or an MS SQL Server database, with the exception of image (coverage) data. The reader opens the connection to the source dataset and reads the `GAliasTable` to determine the proper table names to be used. Next it reads the table of `GFeaturesTable` type to determine the list of tables that contain geometry data. This list of data tables to be read is modified by the specified `ID` and `DEF` lines specified in the mapping file or on the command line. Each geometry table is then read and its features are processed and returned one at a time. When a table is exhausted, the reader starts on the next data table in the list until all tables are read. Issues with reading in a table may result in a specific feature being skipped and sometimes an entire table depending on the severity of the error, but the reader will always try to perform as much translation as possible.

Geometries from GeoMedia do not map exactly to FME geometries. This will have the following effects on the resulting FME features:

- Collections map to one aggregate feature for each FME `fm0_type` depending on which types exist in the collection.
- Multilevel composites may be flattened out to simpler first- or second-level nesting.
- Because GeoMedia is not strict in its typing, the reader can produce some nonsensical features that may be skipped, e.g., a line aggregate containing points.

Reader Directives – all GeoMedia Warehouses

DATASET

Required/Optional: *Required*

The value of this keyword depends on the format of the source dataset. For GeoMedia Access warehouse, it is either the filename of MS Access database or an existing ODBC data source name; for GeoMedia SQL Server warehouse, it is either the database name or an existing ODBC data source name.

When specifying data source name for GeoMedia SQL Server warehouse, a username and password is also required. For GeoMedia Access warehouse, a password is required only if the source warehouse is password-protected.

Range: Valid File Name or ODBC source for Access warehouses

Default: None

Example:

A typical mapping file fragment specifying an input GeoMedia dataset looks like:

```
FM0_DATASET D:\Warehouses\featuredata.mdb
or
FM0_DATASET Access_ODBC_Source

FM0_SQL_DATASET myDatabase
```

or
FM0_SQL_DATASET Sql_Server_ODBC_Source

Workbench Parameter: [<WorkbenchParameter>](#)

DEF

Required/Optional: *Optional*

Each GeoMedia table may optionally be defined before it is read. The definition specifies the name of the table, the type of geometry on each row, the names and types of all attributes and possibly an optional `WHERE` clause, or even an entire SQL statement with which to query the table. The syntax of a GeoMedia `DEF` thus may appear in one of two forms.

The first form allows specification of a `WHERE` clause:

```
<ReaderKeyword>_DEF <tableName> \
[SQL_WHERE_CLAUSE <whereClause>] \
    FM0_GEOMETRY fm0_point|fm0_arc|fm0_line|fm0_area| \
    fm0_text|fm0_none \
    [<attrName> <attrType>]+
```

Note that the Reader uses only the `tableName` and possible SQL modifications, ignoring the other geometry and attribute information, which is relevant only to the writer. An example of an SQL `WHERE` clause that could be used is

```
FM0_DEF States \
    SQL_WHERE_CLAUSE "id > 35" \
    fm0_type fm0_area \
    id integer \
    name char(20)
```

Note: The value for `SQL_WHERE_CLAUSE` should always be enclosed within double quotation marks when specified on `DEF` lines.

where "id" is a column of the table "States". The user is responsible for ensuring the column is contained in the table, and this `WHERE` clause is only appended to the SQL statement `select * from <tableName>`. The word *WHERE* is not included in the `WHERE` clause, but is appended automatically when a clause is specified.

The second form of a `DEF` line involves specification of an entire SQL statement:

```
<ReaderKeyword>_DEF <tableName> \
[SQL_STATEMENT <sqlStatement>] \
    FM0_GEOMETRY fm0_point|fm0_arc|fm0_line|fm0_area| \
    fm0_text|fm0_none \
    [<attrName> <attrType>]+
```

The specified SQL statement is used to place the query against the database. It is again the responsibility of the user to ensure its correspondence with the correct table and columns. An example SQL statement might be

```
SELECT GEOMETRY FROM POINTS WHERE ID=0
```

The following table shows the attribute types that are supported.

Field Type	Description
char (<length>)	Character fields store fixed-length strings. The <code>length</code> parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
date	Date fields store dates as character strings with the format <code>YYYYMMDD</code> . Note that <code><fieldname>.full</code> contains the time as well and is of the format <code>YYYYMMDDHHMMSS</code> .
smallint	Integer fields store whole numbers. This one is 2 bytes or 16 bits long.
integer	Integer fields store whole numbers. This one is 4 bytes or 32 bits long.
float	Real fields store decimal numbers. This one is 4 bytes or 32 bits long.
double	Real fields store decimal numbers. This one is 8 bytes or 64 bits long.
number (<width>, <decimals>)	Fields created with this option will be converted to <code>smallint</code> , <code>integer</code> or <code>double</code> depending on the value of <code>width</code> and <code>decimals</code> parameters. The <code>width</code> parameter is the total number of characters allocated to the field, including the decimal point. The <code>decimals</code> parameter controls the precision of the data and is the number of digits to the right of the decimal. If the <code>decimal</code> is zero and <code>width</code> is less than 5, then the field type will be changed to <code>smallint</code> . If the <code>decimal</code> is zero and <code>width</code> is greater than 5 and less than 10, then the field type will be changed to <code>integer</code> . For all other cases, the field type will be treated as <code>double</code> .

Range: YES | NO

Default: NO

Workbench Parameter: [<WorkbenchParameter>](#)

IDs

Required/Optional: *Optional*

This optional specification is used to identify a specific set of tables to be read from the data source. If nothing is specified, all tables are returned. This feature is useful only if a data source contains many tables and for efficiency you want to read only one table, rather than all of them.

Range: Valid table name in the data source

Default: None

[Workbench Parameter: <WorkbenchParameter>](#)

PASSWORD

For a password-protected GeoMedia Access warehouse, this is a required parameter; otherwise, it is optional. For a GeoMedia SQL Server warehouse, this is a required parameter and will specify the password required to log in to the database specified by DATASET keyword.

TEXT_SIZE_GROUND_UNITS

Specifies the text size in ground units.

Range: Any positive number less than 2,147,483,647

Default: 1

USE_ORIENTED_POINTS

Required/Optional: *Optional*

This directive specifies how the GeoMedia oriented points will be read. If YES is specified, then all GeoMedia oriented points will be returned with orientation information. If NO is specified, then all GeoMedia-oriented points will be returned as normal points without the orientation information. The default value of this keyword is NO.

[Workbench Parameter: <WorkbenchParameter>](#)

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 source database.

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

Range: YES | NO

Default: NO

[Workbench Parameter: <WorkbenchParameter>](#)

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 specification 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 is chosen. If this value is not specified, then it is assumed to be No.

Range: YES | NO

Default: NO

Workbench Parameter: [<WorkbenchParameter>](#)

Reader Directives – GeoMedia Access Warehouse

In addition to the reader directives that apply to all GeMedia warehouses, these directives are specific to GeoMedia Access Warehouses. For GeoMedia SQL Server Warehouse-specific directives, see *Reader Directives – GeoMedia SQL Server Warehouse*.

The directives listed below are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the GeoMedia Warehouse Reader is `FM0`.

Reader Directives – GeoMedia SQL Server Warehouse

In addition to the reader directives that apply to all GeMedia warehouses, these directives are specific to GeoMedia SQL Server Warehouses. For GeoMedia Access Warehouse-specific directives, see *Reader Directives – GeoMedia Access Warehouse*.

The directives listed below are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the SQL Warehouse Reader is `FM0_SQL`.

SQL_SERVER

Required/Optional: *Required*

Specifies the name of the server hosting the MS SQL Server that stores the GeoMedia warehouse.

Range: String

Workbench Parameter: [<WorkbenchParameter>](#)

USER_NAME

Required/Optional: *Required*

This is required only for GeoMedia SQL Server Warehouse. The username for the database must be supplied here, either through the command-line interface or the user interface settings for translation.

WHERECLAUSE

Required/Optional: *Optional*

This optional keyword specifies a `WHERE` clause which is applied to the columns of a table to limit the resulting features.

The main difference between this `WHERE` clause and the one specified on `DEF` lines is that this `WHERE` clause will be applied to all the source feature tables, and the one specified on the `DEF` lines applies to one particular table. In the case when both the `WHERE` clauses are specified, then the `DEF` line `WHERE` clause takes precedence.

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE

Required/Optional: *Optional*

This optional keyword specifies the spatial extent of feature retrieval somewhat similar to GeoMedia's "spatial filtering by fence". If this is not supplied, then all the features are returned. The only current interaction for the bounding box is for features overlapping the specified bounding box.

Based on the value of this keyword, GeoMedia Reader creates a spatial query WHERE clause on the four spatial extents columns <Geometry_XLO>, <Geometry_YLO>, <Geometry_XHI> and <Geometry_YHI>.

Example:

```
<ReaderKeyword>_SEARCH_ENVELOPE <minx> <miny> <maxx> <maxy>  
FM0_SQL_SEARCH_ENVELOPE 0 0 1234.45 2345.56
```

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE_COORDINATE_SYSTEM

Required/Optional: *Optional*

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

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

The syntax of the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is:

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

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

Writer Overview

The GeoMedia Access Warehouse Writer writes features to a Microsoft Access database identified by the DATASET keyword. If the database does not exist, an empty database file is copied and used as a template. The GeoMedia SQL Server Warehouse Writer writes to existing Microsoft SQL Server databases. If the database does not exist, the translation will be terminated. If the metadata tables and/or data tables do not exist in the database specified in the DATASET keyword, they will be created provided user has enough permissions to create and write to tables to the database. If the metadata tables exist, then information about newly created data table(s) will be appended to them. When writing to GeoMedia Access warehouses, if data tables exist and OVERWRITE is NO, then the data is appended to the table without trying to match the schema. When writing to GeoMedia SQL Server warehouses, if data tables exist and fm0_truncate_table is NO, then data is appended to the table without trying to match the schema. If fm0_truncate_table is YES and data tables exist, then all the existing data in the table will be deleted before writing.

As features are routed to either GeoMedia Warehouse Writer by the FME, they determine the layer (feature type) they are in and write the features to the corresponding table. Only one Microsoft Access file database is written during a single FME session, but many tables can be created within the database. Similarly, the SQL Server writer writes to one database, but may create many tables with that one database. The exception to this rule occurs when either of the writer types (MS Access or MS SQL Server) are used in conjunction with the FME Multi-writer and thus could write to several databases or files in a given FME session.

Additional Information About the MS SQL Server Writer

The GeoMedia SQL Server Warehouse Writer does not require GeoMedia to be installed. However, for successful writing, users must have sufficient permissions for creating and deleting tables, and inserting and updating rows in existing tables. All metadata and feature tables will be created with "dbo" ownership, which ensures that any user with permissions has access to the tables. A table with "dbo" ownership means that it is owned by the database administrator.

When writing to GeoMedia SQL Server warehouse, the writer will create four additional columns for storing the extents of geometry. These columns are required by GeoMedia. The names for these columns are derived from the name of the geometry column in the table being written. For example, if the name of the geometry column is *Geometry*, then the four derived column names will be *Geometry_xlo*, *Geometry_ylo*, *Geometry_xhi* and *Geometry_yhi*.

In situations where you are reading from a GeoMedia SQL Server warehouse table and writing to the same or different GeoMedia SQL warehouse, the names of extent columns read from the source warehouse may clash with the derived ones for writing if the geometry column name is same. Such a translation will fail with the error:

```
"SQL Server returned following error message(s):  
- [Microsoft][ODBC SQL Server Driver][SQL Server]Column names in each  
table must be unique. Column name 'Geometry_XLO' in table '<table-name>'  
is specified more than once. ** AND ** [Microsoft][ODBC SQL Server  
Driver][SQL Server]Statement(s) could not be prepared."
```

There are two possible solutions to avoid such errors:

1. Remove the offending extent column names from the destination DEF lines. In workbench this could be accomplished by deleting the attributes names from the destination feature type.

or

2. Change the destination geometry column name. In Workbench, this can be easily done from the Parameters tab on the **Destination Feature Type Properties** dialog or in in mapping files by setting the *fm0_geometry_column* attribute on the destination DEF line to the desired name.

DEF Line options

Creating Indexes

Data tables can be created with either primary or secondary indexes by the GeoMedia Warehouse Writer by appending an indexing suffix on the DEF line of the value of the field to be indexed. To create a primary index on a field, add the suffix

```
,primary
```

to the value. To create a secondary index, add

```
,indexed
```

Note: FME will accept `primaryindex` as a suffix for backwards compatibility, but `primary` is the recommended suffix.

Note that there can only be one primary index per table. If one is not specified, the default `PRIMARYINDEX` column will be created and indexed as the primary index. Alternatively, users can also provide a different name for `PRIMARYINDEX` column using the `fm0_primary_index_column` option on the DEF line. If a column is specified as "primary" indexed then it will override the `fm0_primary_index_column` option for that table.

Note: GeoMedia SQL Server writer allows to create index using multiple columns. This can be done by adding the appropriate suffix next to the column definition.

The following example creates a primary index on the field `ID` and a secondary index on the field `Number` for the table `mytable`.

```
FM0_DEF mytable                                     \
  FM0_GEOMETRY fm0_point                           \
    ID      integer,primary                         \
    Name    char(255)                               \
    Number  float,indexed                           \
```

Example using the `fm0_primary_index_column` option

```
FM0_DEF mytable                                     \
  FM0_GEOMETRY fm0_point                           \
    fm0_primary_index_column MyIndex                \
    ID      integer,primary                         \
    Name    char(255)                               \
    Number  float,indexed                           \
```

Specifying Primary Geometry Column Name

Data tables can be created with a user-specified Geometry column name by supplying its name on the DEF lines.

```
FM0_DEF mytable                                     \
  FM0_GEOMETRY fm0_point                           \
    fm0_geometry_column MyGeometryColumn           \
    ID      integer                                 \
    Name    char(255)                               \
    Number  float                                   \
```

Specifying Whether to Truncate Tables

This option applies only to SQL Server writing. When writing features to data tables, users can specify whether to append to the existing table or delete all the features from the existing table before writing, using the 'fm0_truncate_table' option. If this option is YES then all the features(rows) in the existing data table will be deleted and new features written.

```
FM0_DEF mytable \
    FM0_GEOMETRY fm0_point \
    fm0_truncate_table YES \
    ID integer \
    Name char(255) \
    Number float
```

Creating Clustered or Non-clustered Indexes

This option applies only to SQL Server writing. When writing features to data tables, users can specify whether to create clustered index or not. In SQL Server database only one column per table can have clustered index. Using the fm0_clustered_index option which can have one of three values {PRIMARY, EXTENTS, NONE}, users can specify if the clustered index will be on the primary key column, extent columns or none of the columns. If PRIMARY is specified then the clustered index on the primary key column will be created. If EXTENTS is specified then a clustered index on the four extent columns (<geom>_xlo, <geom>_ylo, <geom>_xhi and <geom>_yhi) will be created and the primary key column will have non-clustered index. If NONE is specified, then all indexes are created as non-clustered.

```
FM0_DEF mytable \
    FM0_GEOMETRY fm0_point \
    fm0_clustered_index PRIMARY \
    ID integer \
    Name char(255) \
    Number float
```

Writer Directives – all GeoMedia Warehouses

The following directives apply to both GeoMedia Access Warehouses and GeoMedia SQL Server Warehouses. The directives listed below are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for GeoMedia Access Warehouse is FM0 and the <WriterKeyword> for GeoMedia SQL Server Warehouse is FM0_SQL.

The remaining writer-specific directives are discussed in *Writer Directives – GeoMedia Access Warehouse* on page 1219 and in *Writer Directives – GeoMedia SQL Server Warehouse* on page 1220.

DATASET

The DATASET directive operates in the same manner as it does for the reader.

DEF

Required/Optional: *Required*

Each GeoMedia table must be defined before it is written to. The definition specifies the name of the table, the type of geometry on each row, the names and types of all attributes and possibly an optional WHERE clause, or even an entire SQL statement with which to query the table. The syntax of a GeoMedia DEF thus may appear in one of two forms.

The first form allows specification of a WHERE clause:

```
<WriterKeyword>_DEF <tableName> \
    fm0_type fm0_point|fm0_arc|fm0_line|fm0_area| \
    fm0_text|fm0_collection|fm0_none \
    [fm0_geometry_column <column name>] \
    [fm0_primary_index_column <column name>] \
    [fm0_drop_table (yes|no)] \
    [fm0_truncate_table (yes|no)] \
    [<attrName> <attrType>]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a <fieldType> is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the fm0_drop_table DEF line parameter is set to YES, or it will be truncated if the fm0_truncate_table DEF line parameter is set to YES. Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 30 characters in length.
fm0_geometry_column	This specifies the name of the column used to store the geometry. Default: Geometry
fm0_primary_index_column	This specifies the name of the column used to store the default primary index. Default: PRIMARYINDEX
fm0_drop_table	This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition. Default: NO
fm0_truncate_table	This specifies that if the table exists by this name, it should be cleared prior to writing. Default: NO
fieldName	The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 30 characters in length.

Parameter	Contents
fieldType	See the Attribute Types section below.

Attribute Types

The following table shows the attribute types that are supported.

Field Type	Description
char (<length>)	Character fields store fixed-length strings. The <code>length</code> parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
date	Date fields store dates as character strings with the format <code>YYYYMMDD</code> . Note that <code><fieldname>.full</code> contains the time as well and is of the format <code>YYYYMMDDHHMMSS</code> .
smallint	Integer fields store whole numbers. This one is 2 bytes or 16 bits long.
integer	Integer fields store whole numbers. This one is 4 bytes or 32 bits long.
float	Real fields store decimal numbers. This one is 4 bytes or 32 bits long.
double	Real fields store decimal numbers. This one is 8 bytes or 64 bits long.
number (<width>, <decimals>)	Fields created with this option will be converted to <code>smallint</code> , <code>integer</code> or <code>double</code> depending on the value of <code>width</code> and <code>decimal</code> parameters. The <code>width</code> parameter is the total number of characters allocated to the field, including the decimal point. The <code>decimals</code> parameter controls the precision of the data and is the number of digits to the right of the decimal. If the <code>decimal</code> is zero and <code>width</code> is less than 5, then the field type will be changed to <code>smallint</code> . If the <code>decimal</code> is zero and <code>width</code> is greater than 5 and less than 10, then the field type will be changed to <code>integer</code> . For all other cases, the field type will be treated as <code>double</code> .

WAREHOUSE_VERSION

The value of this keyword implies compatibility with warehouses created by GeoMedia – the value does not correspond to GeoMedia versions.

- GeoMedia Access Warehouse:
Required/Optional: *Optional*

Access warehouses created with GeoMedia version 4 are different from Access warehouses created with GeoMedia version 5 because of the changes to metadata tables. FME supports creating Access warehouses which are compatible with ware-

houses created by all the GeoMedia versions from 4 to 6. If `OVERWRITE_DATAFILE` is `NO`, then the writer will determine the version of the existing warehouse and write to it regardless of the setting of `WAREHOUSE_VERSION`. If the warehouse does not exist, the writer will create a new warehouse whose version will be determined by `WAREHOUSE_VERSION` keyword setting.

Range: 4, 5 or 6

Default: 5

By default, GeoMedia Access warehouse version 5 is created for a new warehouse or when overwriting an existing warehouse.

[Workbench Parameter: <WorkbenchParameter>](#)

- GeoMedia SQL Server Warehouse:

Required/Optional: *Optional*

With GeoMedia version 5.2, there have been changes to the metadata tables for SQL Server warehouses. FME can create new warehouses or write to existing SQL Server warehouses. When writing to an SQL Server warehouse, there is no option to overwrite, so if a warehouse already exists, then the value of this keyword will be ignored and data will be written to correspond to the warehouse version that exists. This keyword only applies when creating new SQL Server warehouses.

Range: 5, 5.2 or 6

Default: 5

Example:

```
FM0_WAREHOUSE_VERSION 5
```

or

```
FM0_SQL_WAREHOUSE_VERSION 5.2
```

[Workbench Parameter: <WorkbenchParameter>](#)

PLAIN_TEXT

Required/Optional: *Optional*

Note: This directive applies only to writing features of FME type `fme_text`.

By default, the GeoMedia Access Warehouse Writer will format text objects as rich text format (RTF) to insert the text size as part of the object, since text size cannot be set any other way during translation. In some cases, however, plain text may be desired instead of the default RTF text formatting.

Range: YES or NO

Default: YES

Example:

```
FM0_SQL_PLAIN_TEXT YES
```

```
FM0_PLAIN_TEXT NO
```

[Workbench Parameter: <WorkbenchParameter>](#)

FONT_SIZE

Required/Optional: *Optional*

This directive allows you to specify the font size in points.

Range: 1 - 1024

Default: 10

Example:

```
FM0_FONT_SIZE 12
```

Workbench Parameter: [<WorkbenchParameter>](#)

Writer Directives – GeoMedia Access Warehouse

These directives apply only to GeoMedia Access Warehouses. For other GeoMedia directives, see *Writer Directives – GeoMedia SQL Server Warehouse* and *Writer Directives – all GeoMedia Warehouses*.

The directives listed below are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoMedia Warehouse Writer is `FM0`.

OVERWRITE_DATAFILE

Required/Optional: *Optional*

When writing to a Microsoft Access database, the default action performed depends on the existence of the file or ODBC data source defined in `FM0_DATASET`. If the file does not exist, a template warehouse is used as a base and the tables and features to be written are appended to it. If the file exists, the tables and features are appended to it. This is the default behavior and is equivalent to setting the `OVERWRITE_DATAFILE` to `NO`. Setting this directive to `YES` means the template warehouse is used regardless of whether or not the data file exists, and anything existing previously in the warehouse named by `FM0_DATASET` is lost.

Range: YES | NO

Default: NO

Workbench Parameter: [<WorkbenchParameter>](#)

CREATE_SPATIAL_INDEX

When creating a new warehouse or overwriting an existing warehouse, native GeoMedia spatial index creation is performed automatically by the FME GeoMedia Access Warehouse Writer as long as the input data comes from a known coordinate system that has a corresponding datum mapping in the `autodt.ini` file under the GeoMedia install directory. If a spatial index is not desired for the new warehouse, its creation can be turned off by setting this keyword to `NO`.

When appending to an existing warehouse, creation of spatial index depends on whether the `SpatialKeyFieldName` property is set for the geometry column and the spatial key column actually exists. If the property is set and the column exists, then a spatial

key will be automatically created for the geometry, regardless of the `CREATE_SPATIAL_INDEX` keyword setting.

Note: When using GeoMedia 5.0 (05.00.23.04) the GeoMedia Hotfix 05.0023.50 must be applied for spatial indexes to be properly created. This applies to both GeoMedia and GeoMedia Professional. The issue is corrected in GeoMedia 5.1. Recall that a valid source coordinate system is still required in all cases.

Example:

```
FM0_CREATE_SPATIAL_INDEX YES
```

Workbench Parameter: [<WorkbenchParameter>](#)

MODIFICATION_LOG

When writing to a Microsoft Access database, any changes to feature tables or metadata tables are written to `ModificationLog` and `ModifiedTables` metadata tables if modification logging is enabled. By default, modification logging is disabled. When writing large number of features, modification logging can add to the size of database.

Example:

```
FM0_MODIFICATION_LOG NO
```

Workbench Parameter: [<WorkbenchParameter>](#)

MDB_VERSION

This keyword allows you to specify the Microsoft Access file version.

Example:

```
FM0_MDB_VERSION 2002
```

Workbench Parameter: [<WorkbenchParameter>](#)

Writer Directives – GeoMedia SQL Server Warehouse

These specific directives apply only to GeoMedia SQL Server Warehouses. For other GeoMedia directives, see *Writer Directives – GeoMedia Access Warehouse* and *Writer Directives – all GeoMedia Warehouses*.

The directives listed below are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoMedia SQL Server Warehouse is `FM0_SQL`.

SQL_SERVER

Required/Optional: *Required*

Specifies the name of the server hosting the MS SQL Server that stores the GeoMedia warehouse.

Range: String

Workbench Parameter: [<WorkbenchParameter>](#)

USER_NAME

Required/Optional: *Required*

This is required only for GeoMedia SQL Server Warehouse. The username for the database must be supplied here, either through the command-line interface or the user interface settings for translation.

PASSWORD

This is required only for GeoMedia SQL Server Warehouse. It will specify the password required to log in to the database specified by DATASET keyword.

START_TRANSACTION

This statement tells the writer when to start actually writing features into the database. The writer does not write any features until the feature is reached that belongs to `<last successful transaction> + 1`. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter: *<last successful transaction>*

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

Example:

```
FM0_SQL_START_TRANSACTION 0 Workbench Parameter: <WorkbenchParameter>
```

TRANSACTION_INTERVAL

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database. If the TRANSACTION_INTERVAL statement is not specified, then a value of 1000 is used as the transaction interval.

Example:

```
FM0_SQL_TRANSACTION_INTERVAL 1500
```

[Workbench Parameter: <WorkbenchParameter>](#)

IMMEDIATE_WRITE

This statement instructs the FME to immediately write each row of data to the database, rather than batching up writes into bulk arrays. Bulk arrays are normally preferred, as they require fewer queries sent to the database in order to store the data.

Example:

```
FM0_SQL_IMMEDIATE_WRITE NO
```

[Workbench Parameter: <WorkbenchParameter>](#)

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.

GeoMedia features consist of geometric shapes which have their associated attributes as part of their definition. All GeoMedia features contain an `fm0_type` attribute, which identifies the geometric type. The feature contains additional attributes specific to its geometric type. Additional attributes are described in subsequent sections. The common attributes of all GeoMedia features are shown below. The table name is used as the feature type.

Attribute Name	Contents
<code>fm0_type</code>	The GeoMedia geometric type of this entity. Range: <code>fm0_point </code> <code>fm0_line </code> <code>fm0_arc </code> <code>fm0_area </code> <code>fm0_text </code> <code>fm0_none</code> Default: No default

The blob breakdowns for the composite (aggregate), boundary (donut) and collection (aggregate) features are not mentioned below with the simple features, each of which has its corresponding attributes and blob data parsed.

Note: Geometries from FME do not map exactly to GeoMedia geometries. Usually, however, this is not an issue because although some of the internals of the geometries may be changed, their appearance in GeoMedia is still the same.

Points

fm0_type: `fm0_point`

GeoMedia point features specify a single set of coordinates, which is converted, and are devoid of any additional geometric attributes.

fm0_type: `fm0_oriented_point`

GeoMedia-oriented point features specify a single set of coordinates along with a rotation vector. This rotation attribute is stored as a rotation angle.

Attribute Name	Contents
<code>fm0_rotation</code>	This attribute specifies an optional rotation for the shape where the clockwise direction is positive. Range: -360 to 360 degrees Default: 0

Lines

fm0_type: `fm0_line`

GeoMedia line features consist of a list of two or more points. No additional attributes are required to control GeoMedia lines.

Arcs

fm0_type: fm0_arc

GeoMedia arcs consist of a start and end point as well as a normal vector and radius. From these points and their values, the center point of an arc, as well as the start and sweep angles, can be calculated and used to render the arc in FME format. Note that the normal vector serves to identify an arc as being drawn either clockwise or counter-clockwise. Similarly, a positive radius indicates an arc of greater than 180 degrees, while a negative radius indicates an arc of less than 180 degrees.

Attribute Name	Contents
fm0_primary_axis Applicable only with classic geometry.	The length of the semi-major axis in ground units (x-axis). Range: Any real number > 0 Default: No default
fm0_secondary_axis Applicable only with classic geometry.	The length of the semi-minor axis in ground units (y-axis). Range: Any real number > 0 Default: No default
fm0_start_angle Applicable only with classic geometry.	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Default: 0
fm0_sweep_angle Applicable only with classic geometry.	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
fm0_rotation Applicable only with classic geometry.	This attribute specifies and optional rotation for the shape where the clockwise direction is positive. Range: -360 to 360 degrees Default: 0

Areas

fm0_type: fm0_area

GeoMedia area features specify various polygon features. An area is simply a closed line and can be an individual area, a donut, or an aggregate of areas. As with lines, there are no additional attributes for area features.

Text

fm0_type: fm0_text

GeoMedia text is simple and straightforward, offering no font, color or style attributes. Text attributes include the string to be written, a rotation and a possible size.

Note: Although the reader attempts to provide a reasonable value for the text size, it may be necessary to adjust the text size during a Workbench or mapping file translation. Similarly, when writing, the result may have to be adjusted through the GeoMedia product for better results.

Attribute Name	Contents
fm0_justification	<p>The alignment of the text around the origin (not present in Reader)</p> <p>Range: 0..2, 4..6, 8..10</p> <p>0 centered vertically, centered horizontally</p> <p>1 centered vertically, left of the origin</p> <p>2 centered vertically, right of the origin</p> <p>4 above the origin, centered horizontally</p> <p>5 above the origin, left of the origin</p> <p>6 above the origin, right of the origin</p> <p>8 below the origin, centered horizontally</p> <p>9 below the origin, left of the origin</p> <p>10 below the origin, right of the origin</p> <p>Default: 9</p>
fm0_text_size Applicable only with classic geometry.	<p>The size of the text in ground units (not present in Writer)</p> <p>Range: Any real number ≥ 0</p> <p>Default: None</p>
fm0_text_font_size Applicable only with classic geometry.	<p>The size of the text in points (not present in Reader). If this is not specified in the writer, the value from the FONT_SIZE writer keyword is used.</p> <p>Range: Any real number between 0 and 1024.</p> <p>Default: None</p>
fm0_text_string Applicable only with classic geometry.	<p>The text string to be displayed.</p> <p>Range: Any character string</p> <p>Default: None</p>
fm0_rtf_text_string	<p>This attribute is used in both the reader and writer. If this attribute is set when reading, the original RTF text string from the source dataset will be included. If this attribute is set when writing, the RTF text string will be written. In this case, the keyword PLAIN_TEXT will be overridden by this attribute. This option is provided for two main reasons: When translating from GeoMedia to GeoMedia if the source dataset contains RTF text strings, then they will be translated without any loss of formatting information; and it also gives the user the ability to supply custom formatted RTF text strings per feature, written out to the destination dataset.</p> <p>Range: Any RTF character string.(No syntax verification done by the writer)</p> <p>Default: None</p>

Attribute Name	Contents
fm0_rotation	This attribute specifies an optional rotation for the shape where the clockwise direction is positive. Range: -360 to 360 degrees Default: 0
Deprecated – applicable only with classic geometry.	

None

fm0_type: fm0_none

Features with no coordinates are tagged with this type when reading or writing to or from GeoMedia.

```
<ReaderKeyword>_DEF <tableName> \
[SQL_STATEMENT <sqlStatement>] \
    FM0_GEOMETRY fm0_point | fm0_arc | fm0_line | fm0_area |
    fm0_text | fm0_none \
[<attrName> <attrType>]+
```

Troubleshooting

Common issues that arise when using the GeoMedia Access Warehouse Reader and Writer are sometimes a matter of knowledge about how the product works, as well as its limitations.

Spatial Indexes

To create spatial indexes in GeoMedia Professional 5, you will need to install Hot Fix 05.00.23.50, available on Intergraph's website at:

www.intergraph.com/gis/support/GMProHotFix5.asp

Text Size

This can be an awkward issue in writing. Often a dataset will appear with very small text that you cannot see until you zoom in closely, and other times the text seems too large for the data. It is suggested that for reading, use the `TEXT_SIZE_GROUND_UNITS` keyword to set the size appropriate to the bounds of your source dataset. For writing, set the `PLAIN_TEXT` to `NO` and provide a suitable font size using `FONT_SIZE` keyword.

"Class not found" Errors

This is a message from GeoMedia that is passed back through FME. There are two basic causes:

1. The GeoMedia Access Warehouse Writer requires GeoMedia to be installed in order to run; if it is not installed, then this message may result.
2. If GeoMedia is installed, then there is likely more than one copy or version of GeoMedia installed on the machine and the installations are in conflict. This is possible due to the registration scheme of the GeoMedia products. The solution recommended by Intergraph is to uninstall all GeoMedia products, clean the registry of anything related to GeoMedia (especially the `HKEY_LOCAL_MACHINE\Soft-`

ware\Intergraph\Applications key), and then reinstall the single version of GeoMedia you want to use.

Translation Errors

Translation errors can occur if the destination dataset is set to create a file in a directory that does not exist. Since the GeoMedia Warehouse Writer is a file-based writer, it requires that the path in which the destination file is to be created must already exist.

Translation Errors in Workbench

There is a known issue with the GeoMedia SQL Server warehouse writer and Workbench Feature Type Properties.

The Feature Type Properties for the GeoMedia SQL Server warehouse writer are shown at right.

The **Database User** field should be left blank. Entering a username in the field may cause the FME translation to fail. However, even if the translation is successful, GeoMedia will not be able to read the resulting table.