

Database Reader/Writer

FORMAT NOTES:

- Although this chapter applies to ORACLE_DB, ORACLE8_DB, ODBC, MDB, Excel, MSSQL, and DATABASE, all but ODBC 3.x are deprecated and should no longer be used.

Overview

The Database reader and writer modules provide the Feature Manipulation Engine (FME) with access to attribute data held in live database tables. This data may not necessarily have a spatial component to it.

FME provides read and write access to live databases accessible via Open DataBase Connectivity (ODBC) (ODBC driver version 3), or Oracle SQL*Net.

Tip: See the @SQL function in the *FME Functions and Factories* manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

Database Quick Facts

Format Type Identifier	DATABASE
Reader/Writer	Both
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type Attribute	db_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	polygon	no
circles	no	donut polygon	no
circular arc	no	line	no
elliptical arc	no	point	no
ellipses	no	text	no
none	yes	z values	not applicable
surface	no	solid	no

Reader Overview

FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary WHERE clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the Database reader is DATABASE.

DATASET

Required/Optional: *Required*

For ODBC data sources, this is the data source name.

For ORACLE and ORACLE8 data sources, this is the SQL*Net service available in the local configuration (leave it blank to use the default service).

For MS Excel, this is the name of the Excel file or workbook.

For MS Access (MDB), this is the file system location of the database.

Example:

```
DATABASE_DATASET citySource
```

SERVER_TYPE

Required/Optional: *Required*

The type of the server being used. Valid values are ODBC, ORACLE, ORACLE8, EXCEL, MDB and MSSQL. For example,

```
DATABASE_SERVER_TYPE ODBC
```

The difference between ORACLE and ORACLE8 is that ORACLE uses Oracle 7.x OCI calls, and ORACLE8 uses Oracle 8.x OCI calls.

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. For some database types, this is ignored.

```
DATABASE_USER_NAME bond007
```

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. For some database types, this is ignored.

```
DATABASE_PASSWORD moneypenny
```

DEF

Required/Optional: *Required*

Each database table must be defined before it can be read. There are two forms of the definition may take.

The syntax of the first form is:

```
DATABASE_DEF <tableName>                                     \
    [SQL_WHERE_CLAUSE <whereClause>]                       \
    [<fieldName>      <fieldType>] +
```

In this form, the fields and their types are listed. The <fieldType> of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The <tableName> must match a table in the database. This will be used as the feature type of all the features read from the table.

If no <whereClause> is specified, all rows in the table will be read and returned as individual features, unless limited by a global directive:

<ReaderKeyword>_WHERE_CLAUSE

If a <whereClause> is specified, only those rows that are selected by the clause will be read. Note that the <whereClause> does not include the word "WHERE."

In this example, the all records whose ID is less than 5 will be read from the supplier table:

```
DATABASE_DEF supplier                                     \
    SQL_WHERE_CLAUSE "id < 5"                             \
    ID integer                                             \
    NAME char(100)                                         \
    CITY char(50)
```

The syntax of the second form is:

```
DATABASE_DEF <tableName>                                     \
    SQL_STATEMENT <sqlStatement>
```

In this form, an arbitrary complete <sqlStatement> will be executed. The statement is passed untouched to the database (and therefore may include non-portable database constructions). The results of the statement will be returned, one row at a time, as features to FME. This form allows the results of complex joins to be returned to FME.

All features will be given the feature type <tableName>, even though they may not necessarily have come from that particular table. Indeed, with this form, the <tableName> need not exist as a separate table in the database.

In this example, the results of joining the `employee` and `city` tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to `complex`.

```

DATABASE_DEF complex \
  SQL_STATEMENT \
  "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"

```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `SQL_WHERE_CLAUSE` or `SQL_STATEMENT` specified in its `DEF` line, the global `<ReaderKeyword>_WHERE_CLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `SQL_WHERE_CLAUSE` or `SQL_STATEMENT`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
DATABASE_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
DATABASE_WHERE_CLAUSE LENGTH > 2000
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no `IDs` are specified, then all defined and available tables are read. The syntax of the `IDs` keyword is:

```

DATABASE_IDS <featureType1> \
              <featureType2> ... \
              <featureTypeN>

```

The feature types must match those used in `DEF` lines.

The example below selects only the `HISTORY` table for input during a translation:

```
DATABASE_IDS HISTORY
```

NO_UPPER_ATTRS

Required/Optional: *Optional*

By default (for backward compatibility), the reader will create an uppercase copy of each attribute whose name is not already in uppercase. If this directive is specified with the value `YES` these copies will not be created.

Writer Overview

The Database writer module stores attribute records into a live relational database. The Database writer provides the following capabilities:

- **Transaction Support:** The Database 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 Database writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Bulk Loading:** The Database writer uses a bulk loading technique to ensure speedy data load. The performance vastly exceeds a one-insert-at-a-time approach. Bulk loading is not used for the Excel format.

Writer Directives

The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the Database writer is DATABASE.

DATASET

Required/Optional: *Required*

For ODBC data sources, this is the data source name.

For ORACLE and ORACLE8 data sources, this is the name SQL*Net service available in the local configuration (leave it blank to use the default service).

For MS Excel, this is the name of the Excel file or workbook.

For MS Access (MDB), this is the file system location of the database.

Example:

```
DATABASE_DATASET citySource
```

SERVER_TYPE

Required/Optional: *Required*

The type of the server being used. Valid values are ODBC, ORACLE, ORACLE8, EXCEL, MDB and MSSQL. For example,

```
DATABASE_SERVER_TYPE ODBC
```

The difference between ORACLE and ORACLE8 is that ORACLE uses Oracle 7.x OCI calls, and ORACLE8 uses Oracle 8.x OCI calls.

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. For some database types, this is ignored.

```
DATABASE_USER_NAME bond007
```

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. For some database types, this is ignored.

```
DATABASE_PASSWORD moneypenny
```

DEF

Required/Optional: *Optional*

Each database table must be defined before it can be written. For the Database writer, only one form of the DEF line is used:

```
DATABASE_DEF <tableName>                                     \
  [SQL_DROP_TABLE      (Yes|No)]                             \
  [<fieldName>         <fieldType>[,indexed]] +
```

In this form, the fields and their types are listed. If the table already exists in the database, and `SQL_DROP_TABLE` is not specified with a parameter of `Yes`, FME will append its information to the existing database table. In this case, it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. If the fields and types are listed, they must match those in the database. However, not all fields must be listed.

If the table does not exist, or `SQL_DROP_TABLE` is specified with a value of `Yes`, 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. If the table does not exist, or `SQL_DROP_TABLE` is specified with a value of `Yes`, 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. If the `<fieldType>` has a suffix of `,indexed` then FME will attempt to build an index on that field. Dropping tables is not supported for MS Excel files.

Note: If the `<fieldType>` is specified as `LOGICAL`, FME treats it as an `UNSIGNED CHAR(1)` or `RAW(1)` field, depending on the underlying database being used.

Additionally, SQL Server's `NCHAR` and `NVARCHAR` types are now supported to work with strings of wide characters.

This example defines the `SUPPLIER` table for the FME. If the table did not exist, it will be created just before the first `SUPPLIER` row is written. If the table already exists, the data will be appended to the existing table.

```
DATABASE_DEF SUPPLIER                                     \
  ID integer                                             \
  NAME char(100)                                         \
  CITY char(50)
```

The following example is exactly the same, except that it replaces any existing table named `SUPPLIER` with a new table having the specified definition. If the table `SUPPLIER` does not exist in the database, then a new table is simply created.

```
DATABASE_DEF SUPPLIER                                     \
  SQL_DROP_TABLE Yes                                     \
  ID integer                                             \
  NAME char(100)                                         \
  CITY char(50)
```

In the next example, the definition line only make the pre-existing `EMPLOYEE` table known to the FME. Features may later be routed to this table.

```
DATABASE_DEF EMPLOYEE
```

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the Database writer module when to start actually writing features into the database. The Database 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:

```
DATABASE_START_Transaction 0
```

TRANSACTION_INTERVAL

Required/Optional: *Optional*

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

If the `DATABASE_TRANSACTION_INTERVAL` statement is not specified, then a value of 2000 is used as the transaction interval.

Parameter: *<transaction_interval>*

Example:

```
DATABASE_Transaction_INTERVAL 5000
```

IMMEDIATE_WRITE

Required/Optional: *Optional*

This statement instructs 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 trips to the database in order to store the data, but are not supported by some databases. (In particular, some ODBC drivers, such as the Excel driver, do not support bulk array loading.)

Example:

```
DATABASE_IMMEDIATE_WRITE Yes
```

COMPRESS_AT_END

Required/Optional: *Optional*

This statement instructs the FME to compact the database after all writing has been done. This makes use of the existing MDB database option to compact. The compact operation compresses the output database to a small size on disk.

Example:

```
COMPRESS_AT_END Yes
```

VERSION

This keyword applies only when writing to Microsoft Access Database or Microsoft Excel files. Use this keyword to specify the type or version when creating the MDB or Excel file. The default value of this keyword is `default` which will create the appropriate version of MDB or Excel file based on the version of Microsoft Jet Database Engine on the machine. Other options are: 95, 97 or 2000 for Access, and 3, 4, 5, 7, and 97 for Excel.

If you select option 95 or 97, then the MDB file will be of `Access 95 file format`, which works with both Microsoft Access 95 and 97. Selecting option 2000 will create `Access 2000 file format`, which works with Microsoft Access 2000 and 2002. For Excel, selecting version 5.0 or version 7.0 will create an Excel 5.0 file, which works with both Excel 5.0 and 7.0. Excel 97 files will work with Microsoft Excel 97, 2000 and 2002.

Example:

```
MDB_VERSION 97
```

Additional Notes for Excel Files

Excel table names cannot be of the form `<letter><number>`. Examples of this are `A1`, `B22`, `K3000`. Attempting to create a table with this format will result in a failed translation and a corresponding message being returned from the writer.

Additionally, when reading from Excel files, the first row of the worksheet must contain column names. If this is not the case, the first row of data will be read as column names.

When reading or writing an Excel file, it is advisable not to have it open in another program simultaneously as this can severely degrade performance and cause problems in the ODBC driver.

Lastly, it is not possible to create a worksheet outside of FME and then write data to it without first inputting column headings on the first row. FME can write data to an existing worksheet with column headings, or create a new worksheet within an existing Excel file.

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.

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the `DEF` line if the first form of the `DEF` line was used. If the second form of the `DEF` line was used, then the attribute names

are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each Database feature is as defined on its `DEF` line.

Features written to the database have the destination table as their feature type, and attributes as defined on the `DEF` line.

DATE and DATETIME Fields

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

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

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

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

The Database writer looks for both attributes when a date or datetime column is being output. Either may be specified. If both attributes are specified, then the value specified in `UPDATE_DATE.full` is used to populate the `DATE` or `DATETIME` portion of the date, otherwise, this portion is set to 0.

