

Tabular Data Reader/Writer

The Tabular Data Reader and Writer modules provide the Feature Manipulation Engine (FME) with access to the attribute data held in a variety of tabular data file formats. This data may not necessarily have any spatial component to it. The Tabular Data reader is often used to read data from ASCII files and output it to a Geographic Information System (GIS) format. These modules may also be used in combination with the `@Relate` command to translate normalized data into an object model, or vice versa.¹

Currently, dBASE (DBF), Comma-Separated Value (CSV), Column-Aligned Text (CAT), and freeform ASCII relational files are supported by the Tabular Data reader and writer. The Oracle SQL Loader ASCII format can be written by the Tabular Data writer, but no reading support is provided.

Live database systems are not read by this module. As they have greatly different configuration requirements, live database systems are read by the *Database Reader* section described in this manual.

Overview

The FME considers a Tabular Data dataset to be a collection of relational files in a single directory. Tabular Data files of multiple types may be read or written by a single Tabular Data reader or writer. The type and schema of each Tabular Data file must be defined in the mapping file before it can be read or written.

Tabular Data files consist of repeating tuples of attributes, formatted according to their definition. The feature type of each row read from a Tabular Data file is defined in the mapping file.

1. The Tabular Data reader and writer differs from the `@Relate` command. The `@Relate` command is used to join attributes to existing FME features or write out feature attribute data, while the Tabular Data reader and writer create or output entire FME features.

Tabular Data Quick Facts

Format Type Identifier	TABLE
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory
Feature Type	File base name
Typical File Extensions	.fip, .cat, .csv,.dbf
Automated Translation Support	No
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	Not applicable

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

Reader Overview

The Tabular Data reader module produces FME features for all data held in Tabular Data files residing in a given directory. The Tabular Data reader first scans the directory it is given for all Tabular Data files defined in the mapping file. For each Tabular Data file that it finds, it checks to see if that file is requested by looking at the list of IDs specified in the mapping file. If a match is made or no IDs were specified in the mapping file, the Tabular Data file is opened. The Tabular Data reader extracts data from the table one row at a time, produces FME features from them, and passes them on to the rest of the FME for further processing. When the file is exhausted, the Tabular Data reader starts on the next file in the directory.

Reader Directives

The directives processed by the Tabular Data reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the Tabular Data reader is `TABLE`.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the Tabular Data files to be read. A typical mapping file fragment specifying an input Tabular Data data set looks like:

Example:

```
TABLE_DATASET /usr/data/attributes
```

[Workbench Parameter: <WorkbenchParameter>](#)

DEF

Required/Optional: *Required*

Each Tabular Data file must be defined before it can be read. The definition specifies only the base name of the file and its field description. The syntax of a Tabular Data DEF line is:

```
<ReaderKeyword>_DEF <featureType> \
  <tableType> <fileName> \
  [TABLE_MISSING EOF] \
  [<attrName> <attrType>]+
```

The `<featureType>` is used as the feature type of all features read from the table. The `<fileName>` is the full file name, including the extension, of the Tabular Data file. The `<fileName>` does not contain any directory information and it's assumed to be located in the `TABLE_DATASET` directory.

The `<tableType>` determines the physical structure of the table. The table below describes the values recognized for `<tableType>`.

Table Type	Description
ASCII	Relational files of this type follow a regular, but freeform pattern. Fields are separated by spaces or end-of-line characters. Special attribute names and types in the table definition control the placement of the end-of-line characters.
CAT	Column-Aligned Text (CAT) often stores legacy data produced by FORTRAN Input/Output (I/O) statements. Each field is a fixed width with no separating characters between adjacent fields. Each line in a CAT file corresponds to one record. Several logical CAT files can be stored in a single physical CAT file through the use of <i>record keys</i> , which are special fields on each record that determine the record's logical type.

Table Type	Description
SCAT	Structured Column-Aligned Text (SCAT) often stores legacy data produced by FORTRAN I/O statements. Each field is a fixed width with no separating characters between adjacent fields. Each line in a SCAT file corresponds to one record. Whereas CAT treats files with multiple record types as multiple CAT files embedded in one file, SCAT preserves the order enabling ASCII files to be read in which multiple record types form the components of a single feature. This is used to read ASCII formats in which each feature has several different records in the file which comprise it.
CSV	Comma-Separated Value (CSV) files are ASCII files with the record fields separated by commas. Each line in a CSV file corresponds to one record. The default separator, a comma, can be overridden on the file's definition line.
SQL_LOADER	SQL Loader files are text files formatted for loading into relational databases. In particular, the format of these files works well with the SQL Loader utility supplied by Oracle for use with Oracle databases. This type of file can only be written – it is not possible to read these files.
DBF	dBASE Format (DBF) files are formatted according to the dBASE III specification.

If the `TABLE_MISSING EOF` directive is specified before the attribute list, then the table reader will treat a missing table file as if it were present, but contained no data records. (In other words, an attempt to read from the file will simply return no records, rather than setting an error condition.)

Each different file type supports different attribute types. The following subsections detail the allowable attribute types for each file type.

ASCII Field Types

Field Type	Description
AS_CONSTANT	A freeform ASCII file that may contain constants as part of each record. Such constants are not to be transferred to features as attribute values but when the ASCII file is written, the constant must be output. Fields of the type <code>as_constant</code> are used to represent such fields to the FME. The field name in this case is interpreted as being the value of the constant, and not a field name within a feature.
AS_SPECIAL	This field type is used in conjunction with one of the special ASCII field names listed in the next subsection.
<code>char(<width>)</code>	String fields store character strings. The <code>width</code> parameter controls the maximum number of characters that can be stored by the field.

Field Type	Description												
date	<p>Date fields store dates as character strings with the format YYYYMMDD. When date fields are read, they assign the date in the above format to the fieldname. In addition, they assign these fields:</p> <table border="1"> <thead> <tr> <th>Field</th> <th>Contents</th> </tr> </thead> <tbody> <tr> <td><fieldName></td> <td>YYYYMMDD</td> </tr> <tr> <td><fieldName> .yyyy</td> <td>YYYY</td> </tr> <tr> <td><fieldName> .yy</td> <td>YY</td> </tr> <tr> <td><fieldName> .mm</td> <td>MM</td> </tr> <tr> <td><fieldName> .dd</td> <td>DD</td> </tr> </tbody> </table> <p>When a date field is written, it first looks for the complete date to be held in the fieldname. Failing that, it looks in the .yyyy or .yy, .mm, and .dd fields for the date portions, and creates the date from these.</p>	Field	Contents	<fieldName>	YYYYMMDD	<fieldName> .yyyy	YYYY	<fieldName> .yy	YY	<fieldName> .mm	MM	<fieldName> .dd	DD
Field	Contents												
<fieldName>	YYYYMMDD												
<fieldName> .yyyy	YYYY												
<fieldName> .yy	YY												
<fieldName> .mm	MM												
<fieldName> .dd	DD												
float(<width>,<decimals>)	<p>Float fields store floating point values. 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.</p>												
integer(<width>)	<p>Integer fields store integer values. The <code>width</code> parameter is the total number of characters allocated to the field.</p>												
stringlist	<p>A stringlist field in an ASCII table consists of an integer value indicating the number of elements in the list, followed by a series of lines containing the string elements. Such a field is stored or retrieved from an FME list stored in an FME feature.</p>												

ASCII Special Fields

AS_LINEEND{#} AS_SPECIAL	<p>In freeform ASCII files, line-end characters must be explicitly identified to indicate when line breaks occur in the file. Line-ends are indicated by the <code>as_lineend</code> special field name. Each lineend must be assigned a unique number within the table and this number is appended in braces to the <code>as_lineend</code> field name. The <code>as_lineend</code> field names must always have a type of <code>as_special</code>.</p>
AS_COORDINATELIST (<precision>) AS_SPECIAL	<p>Two-dimensional feature coordinate values may be read from or written to freeform ASCII files. Such coordinates are preceded in the file by an integer indicating the number of coordinates. The coordinates are then written, in X Y pairs separated by blanks, on consecutive lines. The optional precision indicates the number of decimal places that will be output for each coordinate value. If it is not specified, then a precision of 2 decimal places will be assumed. The precision is not used when reading.</p>

AS_COORDINATELIST3D
(<precision>) AS_SPECIAL

Three-dimensional feature coordinate values may be read from or written to freeform ASCII files. Such coordinates are preceded in the file by an integer indicating the number of coordinates. The coordinates are then written, in X Y Z triplets separated by blanks, on consecutive lines. The optional precision indicates the number of decimal places that will be output for each coordinate value. If it is not specified, then a precision of 2 decimal places will be assumed. The precision is not used when reading.

CAT/SCAT Field Types

Field Type	Description
BigEndianInt (<width>, <position>)	The <code>width</code> parameter is either 4, 2, or 1 and indicates the number of bytes used to store the integer value. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1. The value is stored in BigEndian binary format.
BigEndianReal (<width>, <position>)	The <code>width</code> parameter is either 8, or 4 and indicates the number of bytes used to store the floating point value. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1. The value is stored in BigEndian binary format.
Integer (<width>, <position>, <zeroPad>, [<signPositive>, [<zeroFill>]])	Integer fields store integer values. The <code>width</code> parameter is the total number of characters allocated to the field. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1. If the <code>zeroPad</code> parameter is <code>Y</code> , then the number is padded with zeros on the left to fill out the field. If the <code>zeroPad</code> parameter is <code>N</code> , then no zero padding is performed. If the optional <code><signPositive></code> parameter is specified and its value is <code>Y</code> , then all non-negative values will have a <code>+</code> prefixed to them. The default is not to do this. If the <code>zeroFill</code> parameter is <code>Y</code> and if the attribute is blank, then it will be written out as zeros in the output file. If the parameter is <code>N</code> , then the field will be filled with spaces in the output file. The default is <code>Y</code> .

Field Type	Description
Real(<width>, <position>, <decimals>)	Real fields store floating point values. The <code>width</code> parameter is the total number of characters allocated to the field, including the decimal point. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1. The <code>decimals</code> parameter controls the precision of the data and is the number of digits to the right of the decimal.
String(<width>, <position> [, <leftjust> [, <strip>]])	String fields store fixed length strings. The <code>width</code> parameter controls the maximum of characters that can be stored by the field. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1. If <code>leftjust</code> is <code>Y</code> (the default), the string is padded with blanks on the right. If <code>leftjust</code> is <code>N</code> , then the padding is on the left of the string. If <code>leftjust</code> is <code>T</code> , then the string is left justified with no padding spaces, but only if it is the last field in the record. When a character field is read, any padding blank characters are stripped if <code>strip</code> is <code>Y</code> (the default). If <code>strip</code> is <code>N</code> , then padding blanks will remain in the data.
XCoordinate(<width>, <position>, <decimals>)	XCoordinate fields store the next x coordinate in the feature as a floating point value. The parameters are interpreted in the same way as a Real field.
YCoordinate(<width>, <position>, <decimals>)	YCoordinate fields store the next y coordinate in the feature as a floating point value. The parameters are interpreted in the same way as a Real field.
ZCoordinate(<width>, <position>, <decimals>)	ZCoordinate fields store the next z coordinate in the feature as a floating point value. The parameters are interpreted in the same way as a Real field.
OptReal(<width>, <position>, <decimals>)	Optional real fields are much like Real fields. The difference is in the handling of attributes which do not have a valid numerical value. In this instance, an optional real field will be left blank, instead of generating an error.

Field Type	Description
<p>DMS(<width>, <position>, [<zeroFillNull>])</p>	<p>DMS fields store floating point angular values as degrees, minutes, and seconds. The <code>width</code> parameter is the total number of characters allocated to the field, including the decimal point, sign, and spaces. The <code>position</code> parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1.</p> <p>The representation of a DMS number will be "+ddd mm ss.sss", where <code>ddd</code> is the (whole) number of degrees, <code>mm</code> is the (whole) number of minutes, and <code>ss.sss</code> is the (decimal) number of seconds. (The number of decimals for the 'seconds' parameter will be limited by the <code>width</code> parameter.)</p> <p>If the optional <code>zeroFillNull</code> parameter has a value of "Y", then a null (or missing) value of the attribute will be represented by a "zero" value ("+000 00 00.000") instead of the empty string..</p>
<p>VarString(<width>[:<name>;<length>]*, <position>, [<leftJustLength>], [<leftJustMembers>])</p>	<p>VarString fields are used to represent variable-length string data. It can be used to define a single variable length string or a variable number of repeating groups of fixed size members. If member <code>names</code> and <code>lengths</code> are not specified, then the former is assumed. In this case, the <code>width</code> characters of field data represent the number of characters in the integer whose value holds the actual string; the string data itself immediately follows the field data in the file. The number of characters contained in the string data will always be exactly the same as the number indicated in the field data area. The string length stored in the field data will be padded with blanks to make it <code>width</code> characters long. If <code>leftJustLength</code> is set to "Y", the padding will come after the number; otherwise, it will come before the number.</p> <p>If the VarString is used to represent repeating groups of fixed size members, each member must be given a <code>name</code> and a fixed <code>length</code>. In this case, the number stored in the field data area does not indicate the total length of the data that follows, but instead the number of groups that repeat. The data stored in each member field will be stored or retrieved from list attributes with the following format:</p> <p>fieldName{ }.MemberName</p> <p>The member string stored in the member field data will be padded with blanks to make it <code>length</code> characters long. If <code>leftJustMembers</code> is set to "Y", the padding will come after the number; otherwise, it will come before the number.</p> <p>All field types following VarStrings will define their starting positions as if this field had zero length data.</p>

Field Type	Description
<pre>ExpReal (<width>, <position>, <decimals>, [<zeroFillNull>], [<altFormat>])</pre>	<p>Exponential real fields store floating point values with an exponentiated representation (+nnn.nnnE+ee). The width parameter is the total number of characters allocated to the field, including the decimal point, signs, and "E" character. The position parameter is the starting column of the field in the CAT record. The columns are numbered starting from 1.</p> <p>The exponential representation of a DMS number will be "mmm.mmmE+ee", where mmm.mmm is the mantissa and ee is the exponent. The decimals parameter specifies how many digits will appear after the decimal of the mantissa.</p> <p>If the optional zeroFillNull parameter has a value of "Y", then a null (or missing) value of the attribute will be represented by a "zero" value ("0.000e+00") instead of the empty string.</p> <p>If the optional altFormat parameter has a value of "Y", then a leading sign and zeroes will always be inserted at the beginning of the representation. If the value is "E", then no leading "+" signs are used on the number or the exponent. If the value is "N" or by default, a leading "+" is only supplied on the exponent. (for example, "+001.46E+02" for "Y", "1.46E2" for "E", and "1.46E+02" for "N" or by default.)</p> <p>Note: If the altFormat parameter is given a value, then the zeroFillNull parameter must also be given a value.</p>
<pre>Constant (<width> <position>, <constant>, [<mustMatch>], [<copyToAttr>])</pre>	<p>Constant fields are fields which always have a specific value of constant. The constant will be padded with spaces or truncated to make it exactly width characters long.</p> <p>If the optional mustMatch parameter is given a value of "Y", then an error is generated when a Constant field's value does not match the specified constant.</p> <p>If the optional copyToAttr parameter is given a value of "Y", then the value of constant will be written to the attribute when reading from the field. (NOTE: If the copyToAttr parameter is given a value, then the mustMatch parameter must also be given a value.)</p>

CAT Special Fields

<pre>[CAT_SUBTYPE (<length>, <start>, <constant>)]</pre>	<p>If the table is a CAT table, a special field may be listed to identify the record key of the records. This allows several different record types to be held in the same physical CAT file. If no CAT_SUBTYPE is identified, then no record key is assumed to be present in the file. For this field, fieldType is used to convey the specifics of the record key. The parameters of this special field are listed in the following table.</p>
--	--

Name	Range	Description	Optional
<length>	Integer	If the file is a CAT type, an optional record key may be specified. This allows several different record types to be held in the same physical CAT file. This parameter defines the length in characters of the optional CAT record key.	Yes
<start>	Integer	The starting column of the key. Columns are numbered starting with 1.	Yes
<constant>	String	The constant used as the optional CAT record key, which occurs in the record at the <start> column.	Yes

SCAT Special Fields

SCAT_SUBTYPE[.]* (<length>, <start>, <constant>)	If the table is a SCAT table, a special field is used to identify the different record types. This enables the SCAT reader and writer to know how to detect the different record types and know how to handle them properly. When defining a SCAT file, there must be at least one SCAT_SUBTYPE specified, with only the first 12 characters needing identical matching (that is, SCAT_SUBTYPE1 and SCAT_SUBTYPE_TEST may also be used to define subtypes in the same way).
--	---

Name	Range	Description	Optional
<length>	Integer	If the file is a SCAT type, an optional record key may be specified. This allows several different record types to be held in the same physical SCAT file. This parameter defines the length in characters of the optional SCAT record key.	Yes
<start>	Integer	The starting column of the key. Columns are numbered starting with 1.	Yes
<constant>	String	The constant used as the optional SCAT record key, which occurs in the record at the <start> column.	Yes
[SCAT_LINE_TERMINATOR <character>]		This optional directive specifies the character which appears at the end of each line. When reading, the character is first stripped off from the lines as they are read to avoid the end character being included as part of the last field. When writing, this character is put on the end of each line after all the fields have been output.	
[SCAT_LINE_WRAP_LENGTH <integer> [, (Y N)]]		This optional directive specifies the maximum physical line length each line may have within the SCAT file. When reading, logical lines that have wrapped are glued back together automatically. When writing, logical lines that are longer than this maximum length are split into multiple physical lines. Fields are generally not split, but moved to the next line if necessary. Fields are only split if they are String fields with the <code>left justify</code> parameter of T, or if the field is the first field on the line and it is still too long. The assumption is also made that all fields are defined in the order that they appear, and that they do not overlap in any location. If "Y" is supplied after the length value, then all output lines are forced to be this length by padding with blanks. If "N" is supplied, or by default, lines shorter than the length are written out as they stand.	
[SCAT_LINE_WRAP_TERMINATOR <character>]		This optional directive specifies the character which appears at the end of each physical line which has been wrapped and which logically continues on the next physical line. When reading, the character is first stripped off from the lines as they are read and glued back together into a single logical line to avoid the end character being included as part of the last field. When writing, this character is put on the end of each physical line that logically continues on the next line.	

[SCAT_LINE_WRAP_PREFIX
<string>]

This optional directive specifies the character string which appears at the beginning of each physical line which has been wrapped and which is a logical continuation of a previous physical line. When reading, the character string is first stripped off from the lines as they are read and glued back together into a single logical line to avoid the beginning character string being included as part of the first field. When writing, this character string is put on the beginning of each physical line that is a logical continuation of the previous line.

CSV/DBF Field Types

Field Type	Description
number(<width>,<decimals>)	Number fields store floating point values. 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.
char(<width>)	Character fields store fixed length strings. The <code>width</code> parameter specifies the number of characters that can be stored. 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.
logical	Logical fields store TRUE/FALSE data. Data read or written from/to such fields must always have a value of either <code>true</code> or <code>false</code> .
date	Date fields store dates as character strings with the format YYYYMMDD.

CSV Special Fields

[CSV_SEPARATOR (<separator>)]	If the table is a CSV table, a special field may be listed to identify the <i>separator</i> used to divide the fields in the file. By default, a comma is used as the separator. For this field, <i>fieldType</i> contains the new separator, enclosed in parentheses. The separator must be only 1 character long. Note: There must be a space between CSV_SEPARATOR and <separator>.	read/write
[CSV_SKIP_LINES <number>]	If the table is a CSV table, a special field may be listed to indicate the number of lines to skip at the top of the file. By default, no lines are skipped. Each line skipped is logged to the log file. This is useful if the CSV file contains a header line of field names or other descriptive material that should be skipped.	read-only

[CSV_STRIP_QUOTES (yes no)]	If the table is a CSV table, a special field may be listed to indicate if quotation marks should be stripped from each attribute. Some CSV files place quotation marks around all values they contain. By setting this special field to <i>yes</i> , then these quotes can be stripped. The default is <i>no</i> .	read-only
[CSV_DUPLICATE_DELIMS (yes no)]	If the table is a CSV table, this special field may be specified to indicate that multiple contiguous delimiters are to be treated as one delimiter instead of treating each delimiter separately. The default is <i>no</i> .	read-only
[CSV_OUTPUT_FIELDNAMES (yes no)]	When writing a CSV file, if this is set to <i>yes</i> , it directs the CSV writer to output the field names as the first row in the CSV file. The default is <i>no</i> . It is ignored when reading CSV files.	write-only
[CSV_QUOTE_OUTPUT (yes no always)]	This specifies whether the fields written to the CSV file are quoted. If set to <i>always</i> , then every field, including field names, will be quoted. If set to <i>no</i> , no fields will be quoted. If set to <i>yes</i> , fields will be quoted only if they contain a delimiter character. Note that the meaning of a <i>yes</i> value differs slightly between the CSV format writer and the CSV mode of the Tabular Data writer.	Optional
[CSV_QUOTE_FIELD_NAMES (yes no)]	This specifies whether the fields names written to the CSV file are quoted. If set to <i>yes</i> , then field names will be quoted. If set to <i>no</i> , no field names will be quoted. The default value is <i>no</i> .	Optional

SQL Loader Field Types

Field Type	Description
float	Float fields store floating point values.
number	Number fields store floating point values.
integer	Integer fields store integer values.
smallint	Small integer fields store integer values that are in the range of -32767..32766.
varchar2 (<width>)	Character fields store variable length strings. The <i>width</i> parameter specifies the maximum number of characters that can be stored.
date	Date fields store dates as character strings with the format YYYYMMDD.

SQL Loader Special Fields

SQL_DEST_TABLE <table_name>	This sets the table name that the SQL loader uses as the destination of the data when it is loaded into the database. SQL Loader files can only be written -- it is not possible to read these files.
--------------------------------	---

IDs

Required/Optional: *Optional*

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

```
<ReaderKeyword>_IDs <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in DEF lines.

The example below selects only the History Tabular Data file for input during a translation:

```
TABLE_IDS History
```

Workbench Parameter: [<WorkbenchParameter>](#)

Writer Overview

The Tabular Data writer creates and writes feature data to Tabular Data files in the directory specified by the DATASET keyword. As with the reader, the directory must exist before the translation occurs. Any old Tabular Data files in the directory are overwritten with the new feature data. As features are routed to the Tabular Data writer by the FME, it determines which file they are to be written to and outputs them according to the type of file. Many Tabular Data files can be written during a single FME session.

Writer Directives

The writer processes the DATASET and DEF directives as described in the Reader Directives section.

Feature Representation

Tabular Data features consist of a series of attribute values. The attribute names are defined in the DEF line, and there is a value for each attribute in each FME Tabular Data feature. The feature type of each Tabular Data feature is also defined on its DEF line.

Reader

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes* on page 7), when the format of the file is SCAT, this special attribute is used by the reader:

Attribute Name	Contents
scat_subtype{0}	The name of the first SCAT subtype that the record matched. This is the subtype that was used to parse the record.

Writer

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes* on page 7), when the format of the file is SCAT, this special attribute is used by the writer:

Attribute Name	Contents
scat_subtype{#}	This attribute list contains the ordered sequence of SCAT subtype records that the feature will produce. One record of each subtype listed will be written to the output stream, in the order they appear in the list. The # ranges from 0 to the number of SCAT subtype records that will be output.

