

Swiss INTERLIS (Tydac) Reader/Writer

FORMAT NOTES:

- This format is not supported by FME Base Edition.
- This format requires an extra-cost plug-in, available from TYDAC AG:
<http://www.tydac.ch>

INTERLIS features a powerful relational data model that is described in its data definition language. INTERLIS is the standard interchange format for Swiss Surveying and was developed to address data interchange between survey companies and local, regional, and federal governments. As a result, INTERLIS is an excellent format for storing and exchanging geographical data in a vendor-neutral manner. The INTERLIS reader enables data that is stored in INTERLIS to be translated to any of the popular vendor formats supported by the Feature Manipulation Engine (FME).

Overview

A model can be defined for any kind of application most effectively by using a data definition language. INTERLIS is, therefore, suitable for all kinds of applications.

INTERLIS follows a relational data model enriched with elements typically used in geographical applications. The language syntax follows the rules of modern programming languages such as PASCAL and MODULA2.

Normally an INTERLIS data set consist of two files:

File Name Extension	Contents
.ili	Data Definition
.itf	Data

INTERLIS supports the storage of point, line, polyline, arc, and polygon geometric data. INTERLIS also stores features with no geometry. Features having no geometry are referred to as having a geometry of *none*.

INTERLIS (Tydac) Quick Facts

Format Type Identifier	INTERLIS
Reader/Writer	Both
Licensing Level	Professional
Dependencies	Requires an extra-cost plug-in from TYDAC AG
Dataset Type	File
Feature Type	Feature role
Typical File Extensions	.ili (.itf)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	interlis_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		

Reader Overview

The INTERLIS reader produces FME features from the features held in an INTERLIS data set. The INTERLIS reader first opens the data definition file `.ili` and retrieves the contained information. It determines which tables are to be read from the data file `.itf` and creates an FME mapping file on the fly. Simultaneously, a model consistency check is performed, however no syntax check is done. The INTERLIS reader then extracts the tables from the data file one at a time and passes them on to the FME where they are processed and converted.

Note: The `.ili` and the `.itf` files must have the same name; for example, `test.ili` and `test.itf`.

Reader Keywords

The following table lists the keywords processed by the INTERLIS reader. The suffixes shown are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the INTERLIS reader is INTERLIS.

DATASET

Required/Optional: *Required*

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

```
INTERLIS_DATASET D:\INTERLIS\TEST.ITF
```

Workbench Parameter: [<WorkbenchParameter>](#)

DEF

Required/Optional: *Required*

Each INTERLIS table must be defined before it can be read. The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of a INTERLIS DEF line is:

```
<ReaderKeyword>_DEF <baseName> \
  [<attrName> <attrType>]+
```

The table names are read using the data definition file. The attribute definition given must match the definition of the data file being read.

The maximum length of topic, table, and attribute names is 24 characters. Character definitions should not be ASCII > 127 codes in topic, table, or attribute names.

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed length strings. The <i>width</i> parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store dates as character strings with the format YYYYMMDD.
decimal(<width>,<decimals>)	Decimal fields store single and double precision floating point values. The <i>width</i> parameter is the total number of characters allocated to the field, including the decimal point. The <i>decimals</i> parameter controls the precision of the data and is the number of digits to the right of the decimal.
integer	Integer fields store 32 bit signed integers.

Field Type	Description
smallint	Small integer fields store 16 bit signed integers and therefore have a range of -32767 to +32767.

The following mapping file fragment defines an INTERLIS table. Notice that neither definition specifies the geometric type of the entities it will contain since INTERLIS tables may contain any of the valid geometry types.

```
INTERLIS_DEF Fixpunkte_LFP          \
  Identification char(12)          \
  Datum2         date              \
  Datum1         date              \
  interlis_oid   integer            \
  Description    char(30)
```

Workbench Parameter: [<WorkbenchParameter>](#)

Writer Overview

The INTERLIS writer creates and writes feature data to an INTERLIS data set specified by the `DATASET` keyword. **Any existing .itf files in the directory are overwritten** with the new feature data. As features are routed to the INTERLIS writer, it determines the tables into which the features are written and outputs them accordingly.

If an `.ili` file exists, it will be used for the model. Otherwise, a new file is created.

The file naming convention is as follows:

```
topic_table
```

which means, as an example,

```
bodenbedeckung_boflaeche
```

is written as **topic** bodenbedeckung and **table** boflaeche.

If the `.ili` file exists, the names must correspond to the definitions in the file, otherwise they are skipped.

Writer Keywords

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

Feature Representation

INTERLIS features consist of geometry and attributes. The attribute names are defined in the `DEF` line and there is a value for each attribute in each INTERLIS feature. In addition, each INTERLIS feature contains several special attributes to hold the type of the geometric entity and its display parameters. All INTERLIS features contain an `interlis_type` attribute, that identifies the geometric type. Depending on the geo-

metric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
<code>interlis_type</code>	The INTERLIS geometric type of this entity. Range: interlis_point interlis_polyline interlis_region interlis_arc interlis_none Default: No default

Points

interlis_type: interlis_point

INTERLIS point features normally specify a three-dimensional (3D) point with x, y and z coordinates in addition to any associated user-defined attributes. The FME INTERLIS writes these points as two-dimensional (2D) points to the output format, because in some cases—for example, with optional points—INTERLIS contains points in a 2D format only. The altitude is often as well-defined as an attribute. In cases where you have 3D points and the altitude is not contained as an attribute, you can use the FME function `@Coordinate`.

In a later release, it is planned to read the z coordinate if available, and to write it to the target format as an attribute.

Polylines

mif_type: interlis_polyline

INTERLIS polyline features specify linear features defined by a sequence of x and y coordinates. In INTERLIS z coordinates would be supported too, however they are not used by the FME.

Regions

interlis_type: interlis_region

INTERLIS region features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes.

Arc

interlis_type: interlis_arc

INTERLIS arc features are linear features used to specify circular arcs. In cases where required by the target format, the arc can be turned into a polyline (see the tip below).

The function `@Arc()` can be used to convert an arc to a linestring. This is useful for storing arcs in systems not directly supporting them.

None

interlis_type: interlis_none

INTERLIS none features represent attribute tables and have no geometry.

Translation Parameters

When you translate an INTERLIS file to any of the supported formats, you can optionally define the following parameters using the **Settings** button:

- Complex Feature Handling
- Arc Handling

Complex Feature Handling

Complex features are objects defined using a combination of lines and arcs. This option allows you to combine such objects into linear features – arcs are converted using the option `arc handling`, see below – or to split them into separate features. The option `Split Into Separate Features` keeps the original geometry and splits the object into separate features, such as lines and arcs. However, this requires that the target format supports such features.

Currently, only the option `Combine Into Linear Features` is supported.

Arc Handling

This option defines how arcs are converted to lines if the option `Combine Into Linear Features` (see above) is chosen. Arcs are stroked into lines using the `Max. Deviation` parameter. This parameter defines the maximum deviation between arc and line in the coordinate units of the source file, normally metres. If no parameter is entered, a default value of 0.5 is used.

Here is an example of the entry shown:

Arc Handling and Overlaps

In INTERLIS, overlaps can be defined as shown in this example:

```

OPTIONAL TABLE BBNachfuehrung =
  Identifikator: TEXT*12;
  Beschreibung: TEXT*30;
  Perimeter: SURFACE WITH (STRAIGHTS, ARCS) VERTEX LKoord
WITHOUT OVERLAPS > 0.200;
  Gueltigkeit: Status;
  Datum1: DATE;
  Datum2: OPTIONAL DATE; !! Vergabe durch Kanton
IDENT
  Identifikator;
END BBNachfuehrung;

```

Overlaps, basically nothing other than errors in the data, means arcs can overlap with other features such as lines. To be able to deal with these errors, if the entered `Max. Deviation` parameter is smaller than the `OVERLAP` value in the INTERLIS data definition file, a `Max. Deviation` of 1.5 times the `OVERLAP` factor is used. 1.5 times is used because INTERLIS data is normally rounded to millimetres (mm) (3 digits) and can add on to the `OVERLAP` error.

Note: The `OVERLAP` value is defined for each table. If you want to make sure that the result is so that the same `Max. Deviation` is used for every table, check all overlaps in the INTERLIS data definition file and use the maximum value of the tables you want to have translated.

Erroneous Arcs

In some INTERLIS files, erroneous arcs were found. This was due to the rounding of coordinates to millimetres causing arcs to become straight lines. This can occur in cases of very small arc features with a very large radius. For instance, in one example an arc feature with a length of 10cm and a radius of 30,000m was found.

In such cases, the translator uses the original points of the INTERLIS file—arc beginning, midpoint, and end. A message, similar to the following, is then written to the log:

```

Invalid arc, writing as line: POINT (764152.604 168727.808)
POINT (764152.658 168727.881) POINT (764152.712 168727.954)

```

Example

The example below shows a sequence of an FME mapping file used to translate some point features from the INTERLIS format into a MapInfo file. The mapping file defines the data set and gives the MapInfo definitions for the file to be written. At run-time, the MapInfo writer is given FME features with full attributes to output.

```

# =====
#
# This mapping file was automatically generated by the FME
# on 01/21/98 15:49:53 for translation between INTERLIS and MAPINFO
#
#=====
INTERLIS_DATASET test.itf
#=====
#-----
# Topic: Fixpunkte
# Table: LFPNachfuehrung
INTERLIS_DEF Fixpunkte_LFPNachfuehrung

```

```

Identifikator      char(12)          \
Datum2             date              \
Datum1            date              \
interlis_oid      integer           \
Beschreibung      char(30)          \
MAPINFO_DEF Fixpunkte_LFPNachfuehrung \
Identifikator      char(12)          \
Datum2             date              \
Datum1            date              \
interlis_oid      integer           \
Beschreibung      char(30)          \
INTERLIS Fixpunkte_LFPNachfuehrung    \
interlis_type     interlis_none      \
Identifikator     %identifikator      \
Datum2            %datum2            \
Datum1            %datum1            \
interlis_oid      %interlis_oid      \
Beschreibung      %beschreibung     \
MAPINFO Fixpunkte_LFPNachfuehrung    \
mapinfo_type     mapinfo_none      \
Identifikator     %identifikator      \
Datum2            %datum2            \
Datum1            %datum1            \
interlis_oid      %interlis_oid      \
Beschreibung      %beschreibung     \

```

Advanced Usage of Mapping Files

Normally, you only use the 1:1 translation of INTERLIS files using the FME Graphical User Interface (GUI). In cases where you have a rich target format that supports, for example, rotated labels such as the MapInfo interchange or native formats, you may want to perform advanced operations on the tables. In other cases, you might want to join tables during translation. This requires the generation and parametrization of an FME mapping file.

To create an FME mapping file, use the **Mapping File/Generate** menu on the FME GUI. Once created, you can edit the mapping file to your needs. However, this requires quite a bit of expertise. You will find some examples below to help you.

Using Lookup Tables

The FME translator automatically creates lookup tables for you. These tables contain the information found in the INTERLIS data definition file such as:

```

TOPIC Bodenbedeckung =
DOMAIN
BBArt =
(Gebaeude,befestigt
(Strasse,Weg,Trottoir,Verkehrsinsel,Bahnflaeche,
Hauserschliessung_f_Fahrzeuge,Flugpisten_Rollwege,
Wasserbecken,uebrige_befestigte),
humusiert
(Acker_Wiese_Weide,
Intensivkultur (Reben, uebrige_Intensivkultur),
Garten_Rasen_Gruen_in_Paerken,Hoch_Flachmoor,

```

```

uebrige_humusierte),
Gewaesser
(stehendes_fliessendes,Schilfguertel),
bestockt
(geschlossener_Wald,uebrige_bestockte),vegetationslos
(Fels,Gletscher_Firn,Geroell_Sand_Ruefe,Abbau_Deponie,
uebrige_vegetationslose));

```

Accordingly, the translator creates the following look-up table in the FME Mapping file:

```

Lookup ArtLut \
  0 Gebaeude \
  1 befestigt-Strasse_Weg \
  2 befestigt-Trottoir \
  3 befestigt-Verkehrinsel \
  4 befestigt-Bahn \
  5 befestigt-Flugplatz \
  6 befestigt-Wasserbecken \
  7 befestigt-uebrige_befestigte \
  8 humusiert-Acker_Wiese_Weide \
  9 humusiert-Intensivkultur-Reben \
  10 humusiert-Intensivkultur-uebrige_Intensivkultur \
  11 humusiert-Gartenanlage \
  12 humusiert-Hoch_Flachmoor \
  13 humusiert-uebrige_humusierte \
  14 Gewaesser-stehendes \
  15 Gewaesser-fliessendes \
  16 Gewaesser-Schilfguertel \
  17 bestockt-geschlossener_Wald \
  18 bestockt-uebrige_bestockte \
  19 vegetationslos-Fels \
  20 vegetationslos-Gletscher_Firn \
  21 vegetationslos-Geroell_Sand \
  22 vegetationslos-Abbau_Deponie \
  23 vegetationslos-uebrige_vegetationslose

```

This look-up table can then be easily joined with the Table `Bodenbedeckung` and be output to any format as shown in the following example which is based on MapInfo.

We now add an attribute definition of `ArtValue` to the `MIF_DEF` statement.

```

MIF_DEF Bodenbedeckung_BoFlaeche \
  interlis_oid integer \
  Entstehung integer \
  Qualitaet char(30) \
  Herkunft char(30) \
  Art integer \
  ArtValue char(80)

```

In the MIF statement, we use the function `@Lookup` (described in the *FME Functions, Factories and Transformers* manual) to read the variable from the look-up table.

```

MIF Bodenbedeckung_BoFlaeche \
  mif_type mif_region \
  interlis_oid %interlis_oid \

```

Entstehung	%entstehung	\
Qualitaet	%qualitaet	\
Herkunft	%herkunft	\
Art	%art	\
ArtValue	@Lookup(ArtLut,%art)	

The MapInfo output table `Bodenbedeckung_BoFlaeche` now includes an additional column named `ArtValue` containing the alphanumeric information of the landcover in addition to just the numerical one. This column can then, for example, be used to automatically generate a legend.

Example 1: Extracting Features and Merging Attributes

INTERLIS is based on a relational data definition. However, many small systems such as ArcView do not store their data in a relational manner by default. For some users who may only need a few tables such as landcover and parcels, it may make sense to *construct* the tables according to their needs during translation. Our target is to extract all buildings from the landcover as a separate layer and to have one single attribute—the number of the building—attached to them. The structure of the landcover files might look as follows:

Files:

- `Bodenbedeckung_BoFlaeche` containing the polygons
- `Bodenbedeckung_Gebaeudenummer` containing the Building Number

Structure:

The key fields are `interlis_oid` and `Objekt`. The field needed is `Nummer`. The target table we want to create will be named `Gebaeude`.

The first step is to create an FME Mapping file using the function Tools > Generate. The Source format is INTERLIS and Destination format is ESRI Shape.

Use a text editor to edit the mapping file:

```
# =====
# Topic: Bodenbedeckung
# Table: BoFlaeche

FACTORY_DEF INTERLIS PolygonFactory \
    INPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche_Geometrie \
    END_NODED \
    OUTPUT POLYGON FEATURE_TYPE Bodenbedeckung_BoFlaeche_ \
        Geometrie_POLY interlis_type interlis_region \
    OUTPUT LINE FEATURE_TYPE Bodenbedeckung_BoFlaeche_ \
        Geometrie_POLY_ERROR interlis_type interlis_polyline

FACTORY_DEF INTERLIS DonutFactory \
    INPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche_Geometrie_POLY \
    INPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche_RAW \
    OUTPUT DONUT FEATURE_TYPE Bodenbedeckung_BoFlaeche_DONUT \
    OUTPUT PIP FEATURE_TYPE Bodenbedeckung_BoFlaeche_PIP \
    OUTPUT POINT FEATURE_TYPE Bodenbedeckung_BoFlaeche_ \
        POINT_ERROR \
    OUTPUT POLYGON FEATURE_TYPE Bodenbedeckung_BoFlaeche_DONUT

FACTORY_DEF INTERLIS PIPComponentsFactory \
    INPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche_PIP \
    OUTPUT POLYGON FEATURE_TYPE Bodenbedeckung_BoFlaeche

INTERLIS_DEF Bodenbedeckung_BoFlaeche \
    interlis_oid integer \
    Qualitaet integer \
    MutNr char(12) \
    Art integer

SHAPE_DEF Bodenbedeckung_BoFlaeche_region \
    SHAPE_GEOMETRY shape_polygon \
    OID number(11,0) \
```

```

QUALITAET          number(11,0)          \
MUTNR              char(12)              \
ART                number(11,0)          \
INTERLIS_Bodenbedeckung_BoFlaeche          \
  interlis_type    interlis_region      \
  interlis_oid     %interlis_oid        \
  Qualitaet        %qualitaet           \
  MutNr            %mutnr                \
  Art              %art                  \
SHAPE_Bodenbedeckung_BoFlaeche_region      \
  OID              %interlis_oid        \
  QUALITAET        %qualitaet           \
  MUTNR            %mutnr                \
  ART              %art                  \
#=====
# Topic: Bodenbedeckung
# Table: Gebaeudenummer
INTERLIS_DEF_Bodenbedeckung_Gebaeudenummer \
  Objekt           integer              \
  NumHALI          integer              \
  NumORI           decimal(5,1)         \
  Nummer           char(12)             \
  interlis_oid     integer              \
  NumVALI          integer              \
SHAPE_DEF_Bodenbedeckung_Gebaeudenummer_point \
  SHAPE_GEOMETRY   shape_point         \
  OBJEKT           number(11,0)         \
  NUMHALI          number(11,0)         \
  NUMORI           number(5,1)          \
  NUMMER           char(12)             \
  OID              number(11,0)         \
  NUMVALI          number(11,0)         \
INTERLIS_Bodenbedeckung_Gebaeudenummer \
  interlis_type    interlis_point      \
  Objekt           %objekt              \
  NumHALI          %numhali             \
  NumORI           %numori              \
  Nummer           %nummer              \
  interlis_oid     %interlis_oid        \
  NumVALI          %numvali            \
SHAPE_Bodenbedeckung_Gebaeudenummer_point \
  OBJEKT           %objekt              \
  NUMHALI          %numhali             \
  NUMORI           %numori              \
  NUMMER           %nummer              \
  OID              %interlis_oid        \
  NUMVALI          %numvali            \

```

We now want to merge the two tables, keep only the attribute `Nummer`, extract the objects which are buildings, and write the result to a table named `Gebaeude`. To do so, insert the following lines after the above.

As a first step, create a copy of both tables using the `TeeFactory`. With that done, keep the required attributes using the function `@KeepAttributes`. Be aware that you also have to keep the attribute `interlis_type`.

```
# We use in a first step two Tee Factories to create a copy
```

```
# of the information we need. We keep only the attributes
# needed using the function @KeepAttributes
```

```
FACTORY_DEF INTERLIS TeeFactory \
  INPUT FEATURE_TYPE Bodenbedeckung_Gebaeudenummer \
  OUTPUT FEATURE_TYPE Bodenbedeckung_Gebaeudenummer \
  OUTPUT FEATURE_TYPE Bodenbedeckung_Gebaeudenummer_For_ \
    ReferenceFactory \
    @KeepAttributes(Objekt,Nummer)
```

```
FACTORY_DEF INTERLIS TeeFactory \
  INPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche \
  OUTPUT FEATURE_TYPE Bodenbedeckung_BoFlaeche \
  OUTPUT FEATURE_TYPE Gebaeude \
    @KeepAttributes(interlis_oid,interlis_type)
```

As a second step, join the tables using the ReferenceFactory. Again, keep only the attributes you want—in this case, the attributes Nummer and interlis_type—using the function @KeepAttributes.

```
# In a second step we join the two tables using a Reference_Factory
# We pipe the result into the new table Gebaeude
# which shall later be used for the output to ArcView Shape
```

```
FACTORY_DEF INTERLIS ReferenceFactory \
  INPUT REFERENCEE FEATURE_TYPE \
    Bodenbedeckung_Gebaeudenummer_For_ReferenceFactory \
  INPUT REFERENCER FEATURE_TYPE Gebaeude \
  REFERENCE_INFO ATTRIBUTES \
  REFERENCEE_FIELDS Objekt \
  REFERENCER_FIELDS interlis_oid \
  OUTPUT COMPLETE FEATURE_TYPE * \
    @KeepAttributes(Nummer,interlis_type)
```

As a last step, define the target data structure for the ArcView Shape file. **Be aware** that, as ArcView uses dBASE file structures for the storage of attributes, all column names have to be defined in uppercase and must not exceed a length of 10 characters. In addition, we define the source (INTERLIS) and the target (Shape).

```
# In a third and last step we define the Arcview output table,
# the 'virtual' INTERLIS input table and pass the attributes over to
# the defined ArcView Regions.
```

```
SHAPE_DEF Gebaeude \
  SHAPE_GEOMETRY shape_polygon \
  NUMMER char(12)

INTERLIS Gebaeude \
  interlis_type interlis_region \
  Nummer %nummer

SHAPE Gebaeude \
  NUMMER %nummer
```

Example 2: Merging Two Tables to Create a MapInfo Text File (Label)

INTERLIS files normally contain, as well, text information (labels), such as parcel numbers. These numbers are usually positioned, rotated, etc. Therefore, it would be extremely useful to be able to transfer this information 1:1 to the target format. However, not every target format supports such features. For the following example we will use the MapInfo format. It does not matter if it is native Table (TAB) or MapInfo Data Interchange Format (MIF/MID).

The problem with INTERLIS is that text and position are not often stored in the same table. In addition, it cannot be told if something is really a text and what should be used for the display. Alternatively, you may want to use anything in the table as a label. Such an INTERLIS definition looks like this:

This is the table that contains the text to be labelled—attribute `Nummer`:

```
TABLE Grundstueck =
MutNr: Arbeitsnummer;
Nummer: TEXT*12;
Gueltigkeit: (rechtsgueltig, streitig);
Art: Grundstuecksart;
IDENT
Nummer;
END Grundstueck;
```

This is the table that contains all of the information about the position; such as coordinates, orientation, and alignment:

```
TABLE GrundstueckPos =
Objekt: -> Grundstueck; !! Beziehung 1-m
NumPos: LKoord // Position in der Regel innerhalb der Flaechе //;
NumOri: OPTIONAL SchriftOri; !! Default: 100.0
NumHali: OPTIONAL HALIGNMENT; !! Default: Center
NumVali: OPTIONAL VALIGNMENT; !! Default: Half
NO IDENT
END GrundstueckPos;
```

Be aware that Swiss surveying uses a special angular unit (named *gon*) where a full circle is 400gon instead of 360°. For the text rotation, INTERLIS defines a default value which is normally 100gon and means horizontal text alignment which, in the case of MapInfo, corresponds to 0°. This means you have to translate the INTERLIS rotation angles to MapInfo rotation conventions by subtracting 100, dividing by 400 and multiplying by 360. Furthermore in INTERLIS, rotations are defined clockwise, while in MapInfo, they are defined as counterclockwise. Don't worry—the FME can handle all of that.

If you look at an FME mapping file created for conversion to MapInfo, you will find all kinds of macros at the beginning of the file which are automatically generated for you. These macros define, for example, MapInfo colors, patterns, symbols, etc. Later you can use them for color assignment in the mapping file. The color definition looks like the following example:

```
#=====
# MAPINFO COLOR NAMES
# The following color name macros provide convenient, meaningful
```

```
# names to many common colors. The values are the 24 bit represen-
# tation of 8 bits each of red, green, and blue.
# Exp.: SupplyAttributes(mapinfo_symbol_color,$(MAPINFO_Cherry))
```

```
MACRO MAPINFO_DarkRed          5570560
MACRO MAPINFO_MediumRed       11141120
MACRO MAPINFO_BrightRed       16711680
MACRO MAPINFO_LightRed        16733525
MACRO MAPINFO_Brick           10502208
```

The automatically generated FME mapping file for the above looks as follows:

```
#####
# Topic: Liegenschaften
# Table: Grundstueck
INTERLIS_DEF Liegenschaften_Grundstueck \
  Nummer          char(12) \
  interlis_oid    integer \
  Entstehung      integer \
  Herkunft        char(30) \
  Art             integer \
  Gueltigkeit     integer \
MAPINFO_DEF Liegenschaften_Grundstueck \
  Nummer          char(12) \
  interlis_oid    integer \
  Entstehung      integer \
  Herkunft        char(30) \
  Art             integer \
  Gueltigkeit     integer \
INTERLIS Liegenschaften_Grundstueck \
  interlis_type   interlis_none \
  Nummer          %nummer \
  interlis_oid    %interlis_oid \
  Entstehung      %entstehung \
  Herkunft        %herkunft \
  Art             %art \
  Gueltigkeit     %gueltigkeit \
MAPINFO Liegenschaften_Grundstueck \
  mapinfo_type    mapinfo_none \
  Nummer          %nummer \
  interlis_oid    %interlis_oid \
  Entstehung      %entstehung \
  Herkunft        %herkunft \
  Art             %art \
  Gueltigkeit     %gueltigkeit \
#####
# Topic: Liegenschaften
# Table: GrundstueckPos
INTERLIS_DEF Liegenschaften_GrundstueckPos \
  Objekt          integer \
  NumHali         integer \
  NumOri          decimal(5,1) \
  interlis_oid    integer \
  NumVali        integer \
MAPINFO_DEF Liegenschaften_GrundstueckPos \
```

```

Objekt            integer \
NumHAlI           integer \
NumOri            decimal(5,1) \
interlis_oid     integer \
NumVAlI           integer \

INTERLIS Liegenschaften_GrundstueckPos \
  interlis_type  interlis_point \
  Objekt         %objekt \
  NumHAlI        %numhali \
  NumOri         %numori \
  interlis_oid   %interlis_oid \
  NumVAlI        %numvali \
MAPINFO Liegenschaften_GrundstueckPos \
  mapinfo_type  mapinfo_point \
  Objekt         %objekt \
  NumHAlI        %numhali \
  NumOri         %numori \
  interlis_oid   %interlis_oid \
  NumVAlI        %numvali \

```

Now what has to be done is to join the two tables using the attributes `Objekt` of `Liegenschaften_Grundstueck` and `interlis_oid` of `Liegenschaften_GrundstueckPos`. In addition, you have to define the resulting table as a `MapInfo` type `mapinfo_text`.

To do so, you have to define two *TeeFactories*. This type of factory implements a T-junction in the factory pipeline. It replicates each feature that matches the input specifications and outputs a copy of the feature for each `OUTPUT` clause specified. This factory is useful when copies of features need to be generated for further processing by other factories—in our case, a *ReferenceFactory*—or for output to different layers—in our case, the `Text-Layer`—or themes in the output data set.

Our mapping looks as follows:

```

# In a first step we define two tee factories to copy
# the needed attributes to be processed by the reference
# factory where we join the two tables. We only need the key
# attributes (in this case interlis_oid and Objekt)
# and the attributes Nummer from the table
# Liegenschaften_Grundstueck (which is the label) and
# NumOri from Liegenschaften_GrundstueckPos
# (which is the label rotation)
# We do this using the function @KeepAttributes

FACTORY_DEF INTERLIS TeeFactory \
  INPUT FEATURE_TYPE Liegenschaften_Grundstueck \
  OUTPUT FEATURE_TYPE Liegenschaften_Grundstueck \
  OUTPUT FEATURE_TYPE Liegenschaften_Grundstueck_For_ \
    ReferenceFactory \
    @KeepAttributes(Nummer,interlis_oid) \

FACTORY_DEF INTERLIS TeeFactory \
  INPUT FEATURE_TYPE Liegenschaften_GrundstueckPos \
  OUTPUT FEATURE_TYPE Liegenschaften_GrundstueckPos \
  OUTPUT FEATURE_TYPE Liegenschaften_GrundstueckPosLabel \
    @KeepAttributes(Objekt,NumOri) \

```

The next step is to use the *ReferenceFactory* to join the two tables. The key fields are `interlis_oid` in the table `Liegenschaften_Grundstueck` and `Objekt` in the table `Liegenschaften_GrundstueckPos`. This is given by the INTERLIS data structure. If you do not know which ones are the key attributes to be used for the join, you do an automatic translation in a first step and look at the two tables using the MapInfo Table Browser.

```
# In a second step we join the two tables using a Reference_Factory
# We pipe the result into the new table
# Liegenschaften_GrundstueckPosLabel
# which shall later be used for the output to MapInfo

FACTORY_DEF INTERLIS ReferenceFactory \
  INPUT REFERENCEE FEATURE_TYPE \
    Liegenschaften_Grundstueck_For_ReferenceFactory \
  INPUT REFERENCER FEATURE_TYPE Liegenschaften_ \
    GrundstueckPosLabel \
  REFERENCE_INFO ATTRIBUTES \
  REFERENCEE_FIELDS interlis_oid \
  REFERENCER_FIELDS Objekt \
  OUTPUT COMPLETE FEATURE_TYPE * \
  OUTPUT INCOMPLETE FEATURE_TYPE * \
  OUTPUT NO_REFERENCES FEATURE_TYPE *
```

The last step is to define the output table and pass the results into it. Be aware that no attribute definition is needed for the MapInfo text table (`DEF` statement) as we only use standard attributes:

```
# In a third and last step we define the MapInfo output table,
# the 'virtual' INTERLIS input table and pass the attributes
# over to the defined MapInfo Text Table.
# As INTERLIS uses 400° units and clockwise, MapInfo
# however 360° and counterclockwise we have to do
# a small calculation on the angle. We do that using
# the @Evaluate function. Font, font size and positioning we define
# manually, as these are incompatible between systems like
# UNIX and WINDOWS

MAPINFO_DEF Liegenschaften_GrundstueckPosLabel

INTERLIS Liegenschaften_GrundstueckPosLabel \
  Nummer %nummer \
  NumOri %numori \

MAPINFO Liegenschaften_GrundstueckPosLabel \
  mapinfo_type mapinfo_text \
  mapinfo_text_string %nummer \
  mapinfo_text_justification "center" \
  mapinfo_text_fontname "Arial" \
  mapinfo_text_height "2" \
  mapinfo_rotation @Evaluate("-((%numori - 100) * 180.0 / 200)")
```

Using this mapping file, a new table named `Liegenschaften_GrundstueckPosLabel` is created in addition to all other INTERLIS tables.

