

TM Units of Measure Conversion

Introduction

Transformation Manager (TM) includes an automatic unit of measure (UOM) conversion system for all numeric data types and for string types, if this is appropriate.

The conversion system is based on the principles of Class and Unit. Example Classes are length, volume, etc. For each Class (e.g. length) there will be a number of defined Units (e.g. metre, centimetre, inch, mile etc).

For each UOM Class an international standard (SI) Unit is defined (e.g. metre for Class length). All units within the same class are defined in terms of this standard unit. Thus, conversion between any two units within the same class is possible by converting to and from the standard unit.

The standard classification system is defined in an XML file `si_uom.xml` and is provided with TM. This file is loaded when TM is launched. If UOM conversions are required during the transformation process the rules specified by this file are applied.

To specify the Class and Unit for attributes in a particular model the model is exported to an XML file. This file is edited by the user to specify each attribute's Class and Unit. The file is then re-imported. The UOM attribute information is stored in the repository.

Users may extend known UOM Classes by adding new Units to the model export file. The new unit must be defined in terms of the standard unit of the class that is being extended. These non-standard units, and their conversions to other units within the same class, will be available for TM projects that use that model.

A built-in UOM Conversion function is provided, to be called explicitly when the source is a local variable, an expression, or the result of a UDF.

Using Units of Measure Conversion

There are two project preferences that enable the UOM system:

- 'Preferences | Generate SML | UoM - Generate for dynamic UoM system'. This causes the system to look for and generate UOM conversions for all numeric types.
- 'Preferences | Generate SML | UoM - Generate for dynamic UoM for string data'. This causes the system to look for and generate UOM conversions for string types. This is useful if a model has been originally defined generically as containing string data but in practice contains numeric data.

By default, neither preference is enabled.

The standard SI classes and units are defined in a file `si_uom.xml`. This file is provided with TM in the installation directory. The name and location of this file can be changed using the global setting (available when TM is disconnected from a repository):

- Preferences | Builder General | UOM SI Default Conversions File. Note that the file may be specified using '`$CONN_PATH`' and '`$INCL_PATH`' to position the file within a repository.

si_uom.xml

The `si_uom.xml` file describes classes, the units within each class, and each unit's conversion factors in relation to the standard unit for its class.



Below is a cut-down version of the si_uom.xml file. The example shown contains 2 classes, 'metre' and 'ohm'. Many of the units within the 'metre' class have been omitted.

Note that in this example, each class is named according to its SI unit. Thus, the class describing units of length is named 'metre' rather than 'length'. This convention is used throughout the si_uom.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<UOMConversions>
  <Class description="metre" name="metre">
    <Y tgt="m">
      <X c="0" d="0" description="yards" g="1" m="0.9144" src="yd"/>
      <X c="0" d="0" description="sixtyfourths of an inch" g="64" m="0.0254"
src="64ths"/>
      <X c="0" d="0" description="millimetres" g="1" m="0.001" src="mm"/>
      <X c="0" d="0" description="inch" g="1" m="0.0254" src="in"/>
      <X c="0" d="0" description="kilometre" g="1" m="1000" src="km"/>
      <X c="0" d="0" description="mile" g="1" m="1609.344" src="mi"/>
      <X c="0" d="0" description="nautical mile" g="1" m="1852" src="nautmi"/>
      <X c="0" d="0" description="nanometres" g="1" m="0.000000001" src="nm"/>
      <X c="0" d="0" description="chains" g="39.370147" m="792" src="chSe"/>
      <X c="0" d="0" description="centimetre" g="1" m="0.01" src="cm"/>
      <X c="0" d="0" description="decimetre" g="1" m="0.1" src="dm"/>
      <X c="0" d="0" description="foot" g="1" m="0.3048" src="ft"/>
    </Y>
  </Class>
  <Class description="ohm" name="ohm">
    <Y tgt="ohm">
      <X c="0" d="0" description="microohm" g="1" m="0.000001" src="uohm"/>
      <X c="0" d="0" description="milliohm" g="1" m="0.001" src="mohm"/>
    </Y>
  </Class>
</UOMConversions>
```

The definition file is loaded when TM is launched.

Each unit defines the 'm', 'c', 'd' and 'g' values needed to convert to the standard unit for its class, according to the general formula:

$$y := (mx + c) / (dx + g).$$

Because each unit in a class can be converted to the standard unit, conversion between any two units within a class is possible by converting to and from the standard unit.

Therefore given two formulae for conversion of two units (Y and X) to SI units:

$$SI = (mY + c) / (nY + d)$$

$$SI = (aX + w) / (bX + v)$$

Then

$$mY + c) / (nY + d) = (aX + w) / (bX + v)$$

And Y can be expressed in terms of X as follows:

$$(mY + c) (bX + v) = (aX + w) (nY + d)$$

$$mY(bX + v) - nY(aX + w) = d(aX + w) - c(bX + v)$$

$$Y = (d(aX + w) - c(bX + v)) / (m(bX + v) - n(aX + w))$$

Note that TM at design time will refuse to load conversions which assign both d and g to zero.

Adding UOM information to your model.

When a model is first loaded into TM, the system has no knowledge of its attributes, units and classes. Users have to provide this information, which is subsequently stored with the model in the repository.

UOM information for a model is initially specified by exporting the model to a file, specifying class and unit information for the attributes and importing the file.

In the general case you should always begin by using the export facility. This builds an XML-like template structure to which the actual units of measure information are added.



The export file is created by right-clicking on the model node in the tree display and selecting **Export Model | Export UOMs**. The name and location of the export file is specified and the file is created. The model file can have the suffix `.uom` or `.xml`.

The contents of an example model file would be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- UOM type export generated by Transform Manager (http://www.etlsolutions.com) -->
<!-- Model: UNITS_1_0 exported at Wed Jan 04 10:53:44 GMT 2006 -->
<UOM>
  <TABLE name="DISTANCE">
    <COL name="height_In_cm"      uom="[UNKNOWN]" class="[UNKNOWN]" />
    <COL name="height_In_feet"    uom="[UNKNOWN]" class="[UNKNOWN]" />
    <COL name="height_In_miles"   uom="[UNKNOWN]" class="[UNKNOWN]" />
  </TABLE>
  <TABLE name="TEMPERATURE">
  </TABLE>
  <TABLE name="UNITS">
  </TABLE>
</UOM>
```

Note that UOM is enabled for numeric types only, i.e. only attributes with numeric types will be listed in the model file. The attribute type can be changed before the export process if required.

The model file should be opened in any text editor and the information specified:

```
<COL name="height_In_cm"      uom="cm" class="metre" />
<COL name="height_In_feet"    uom="ft" class="metre" />
<COL name="height_In_miles"   uom="mi" class="[UNKNOWN]" />
```

Note that the class field does not need to be completed if the unit is unique across all classes. The class will be inferred automatically by the system.

When the `model.uom` file has been edited, it should be imported by right-clicking the model node in the tree display and selecting **Export Model | Import UOMs...**

Note that the unit of measure data will be permanently attached to the model and stored in the repository. There is no need to re-import the UOM data every time the model is opened.

If you re-export the UOM data, it will be exported with the unit and class information.

If you re-export the UOM data it will exported with class information. There is no need to restart TM, nor re-open the project when modifying `.uom` files, nor when re-importing, since TM will load be changes you have made dynamically and will save them when you save the project.

Note finally that TM displays the use of automatic UOM conversions in the Design Tool and on the message bar. By right-clicking on an UOM assignment TM will display the formula used to calculate conversion.

UOM Conversions UDFs

To achieve a UOM conversion if the source is a local variable, an expression, or the results of a UDF it is necessary explicitly to call a UOM Conversion function with syntax. For example:

```
target_attr := UoM_Convert(source_value, source_units);
```

An example of this might be:

```
height_in_metres := UoM_Convert(height_in_centimetres, 'cms');
```



Using the system at run-time.

If the units are unchanged between design time and run-time, then nothing additional needs to be done at run-time. The system will use the units and conversions as defined automatically. Note that the `si_uom.xml` file does not need to be present at run-time.

It may be required to change the UOM to another set at run-time, for example if a particular data store is sometimes stored as metric and sometimes stored as imperial. In this case, a `.uom` should be prepared containing the correct set of units. Then, at run-time this file should be present and loaded in from a built-in function:

```
loadUOMFile(IsSrc, path-to-.UOM-file);
```

For example:

```
loadUOMFile(false, '....\metricSetForMyTgtModel.uom'); to set the  
target to use a different UOM system
```

or

```
loadUOMFile(true, '....\metricSetForMySrcModel.uom'); to set the  
target to use a different UOM system.
```

Note that the call to `loadUOMFile` should occur in a `$PreDocument` or `$Document` transform – leaving it too late may result in missed units of measure conversions.

The same functionality is available to be called from the TM Java API – the function is called `loadUOMFile` and is found in the class `net.etl.tm.TMTransformerCommon`.

Note that no mechanism has currently been supplied for altering the conversion formulae at run-time. It would easily be possible to add such a method if one were required.

Extending the UOM system

The UOM system can be extended so that user-defined units can be added to existing classes and that completely new classes can be added also. This is done on a per-model basis, rather than changing the `si_uom.xml` file.

When the export file of a model is being edited, the extensions can be added as follows:

```
<Class description="std cubic metres" name="std_cubic_metres">  
  <Y tgt="m3">  
    <X c="0" d="0" description="barrel" g="1" m="0.1589873" src="bbl"/>  
    <X c="0" d="0" description="billion cubic feet" g="1" m="28316850" rc="bcf"/>  
  </Y>  
</Class>
```

The Class elements should appear after the Table elements within the UOM element.

Compatible Units

For a UOM conversion to occur the units of both sides of an assignment must be compatible, that is, they must belong to the same class.

When a project is built in TM, the system evaluates the UOM units on each side of an assignment and infers their class. The source attribute's class must match the target attribute's class. Thus, the system prevents incorrect assignments.

Note that when the unit's type name is unique in the UOM classifications there is no need to assign values to the class column. Even in the case where the type name is not unique but can be deduced from a unique UOM involved in the assignment (source or target) then TM will assume that the non-unique type name has the same class as the unique UOM and provide a warning. In the case where both the source and target are not unique an error will be reported. In the cases where these assumptions



prove to be incorrect (or to provide a more failsafe conversion process) the user may defined the UOM class as well as the type name when adding UOM details. For example, a linear measurement cannot be assigned directly to a volume measurement.

Thus:

```
GasVolume := LinearMeasurement;
```

will be rejected by the SML parser (provided the UOM preference is enabled).

Such an assignment would require a user defined function:

```
GasVolume := getCubicVolume(LinearMeasurement);
```

However, if the unit on the source side is ambiguous, but on the target side is defined, then according to the class on the target side, this is the call that should be made.

Further Information

Website : www.etsolutions.com

Email : info@etsolutions.com

Telephone : +44 (0) 1912 894040