Ansys GRANTA MI 2021 R1

GRANTA MI FEA Exporter Author Guide

Contents

1 Introduction

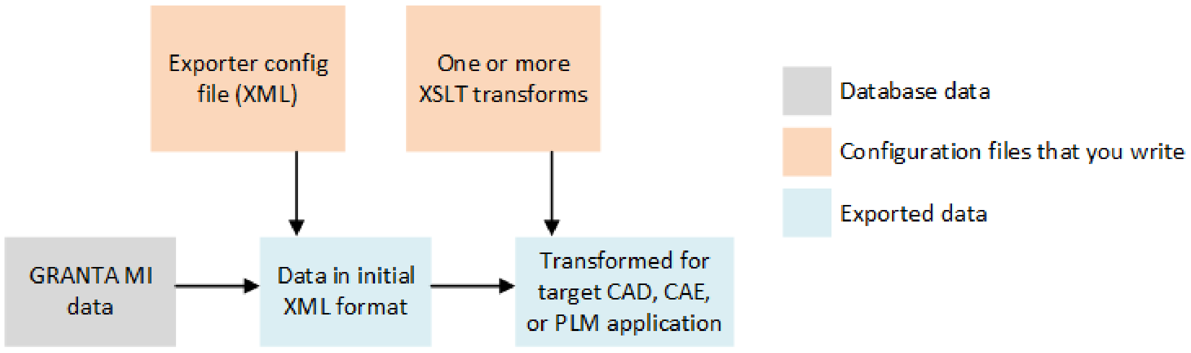
Granta FEA exporters allow you to export data from a GRANTA MI database in formats that can be imported into a number of different CAD, CAE, and PLM applications. You can author your own exporter to export GRANTA MI data in any text-based file format.

All data types except File and Picture data can be exported from a Granta database, including functional data (grid and series), and equations and logic data. Version information and quality ratings may also be exported, and users can choose the unit system for numerical data.

The process of exporting data from a GRANTA MI database and importing it into a CAD, CAE, or PLM applications can be split into two stages;

1. Extracting the data from the database. Data is output as XML that conforms to a published schema (this XML is referred to as "initial XML"). An exporter configuration file (.exp) must be written to specify which data will be extracted from GRANTA MI.
2. Using XSLT (Extensible Stylesheet Language Transformations) to manipulate the initial XML data into a format that the target application can read.

The figure below summarizes the export process.


Exporter config file (XML)
One or more XSLT transforms
GRANTA MI data
Data in initial XML format
Transformed for target CAD, CAE, or PLM application
Database data
Configuration files that you write
Exported data

2 Scope of this document

This document describes how to develop an exporter configuration that will produce the initial XML containing the required data from your GRANTA MI database, and how the exported data appears in the initial XML.

This document does not cover how to write XSLT to transform the initial XML data into the format required by the target CAD, CAE, or PLM package. The XSLT language is specified and maintained by the World Wide Web Consortium (W3C), and for information on how to write XSLT you should consult XSLT reference documentation or online resources, for example the XSLT Tutorial at w3schools.com (external link). The full specification of XSLT (version 1.0) is available on the W3C website at w3.org (external link).

The following XML schema files are provided along with this document in the zip file *FEA_Schema.zip*. You can refer to these files when developing exporters:

- InitialXml.xsd and matml31.xsd — describe the Initial XML output schema, that is, the XML that will be transformed by customer-written XSLT.
- ConfigSchema.xsd — defines the schema of exporter configuration files (.exp files). This file includes comprehensive documentation on all the XML elements and attributes included in the schema, and is an excellent source of information when you are modifying exporters, or developing new exporters.

3 Exporter files

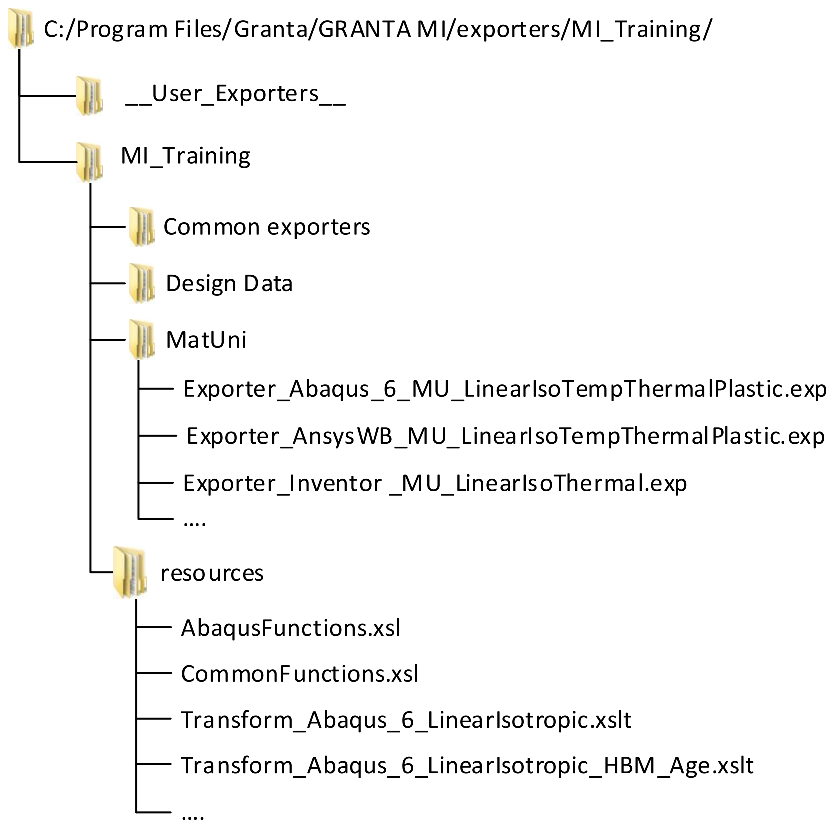
An exporter for a given CAD, CAE, or PLM package and model is comprised of two (or more) files.

- The **exporter configuration file** (.exp), a file that defines the structure of the initial XML, including the name of the target CAE analysis package, the material model type, the attribute data to be exported, and the transform files that can be used with it.
- One or more XSLT **transform files**, which convert the initial XML into the format required by the host application.

Note: XSLT can be used to run arbitrary malicious code, and exporters should only be installed from trusted sources.

Exporters are added to a GRANTA MI database by importing the exporter configuration (.exp) file and associated XSLT file(s) into the database for which it was written using the Schema tool in MI:Admin. For details, see Section 10.

The exporter source files for each database are located in subfolders named after the database key under the MI:Server *exporters* folder (C:\Program Files\Granta\GRANTA MI\Server\exporters), for example:



```
C:/Program Files/Granta/GRANTA MI/exporters/MI_Training/
├── __User_Exporters__
└── MI_Training
    ├── Common exporters
    ├── Design Data
    ├── MatUni
    │   ├── Exporter_Abaqus_6_MU_LinearIsoTempThermalPlastic.exp
    │   ├── Exporter_AnsysWB_MU_LinearIsoTempThermalPlastic.exp
    │   ├── Exporter_Inventor _MU_LinearIsoThermal.exp
    │   └── ....
    └── resources
        ├── AbaqusFunctions.xsl
        ├── CommonFunctions.xsl
        ├── Transform_Abaqus_6_LinearIsotropic.xslt
        ├── Transform_Abaqus_6_LinearIsotropic_HBM_Age.xslt
        └── ....
```

3.1 Developing your own exporters

When writing your own FEA exporters, you may want to store the files on disk while they are under development. You can structure the files as you wish, however, any files that you create should be kept completely separate from the Granta-supplied exporter files, to avoid being overwritten by future updates.

Placing custom exporter files in the specially-named __User_Exporters__ subfolder on the server will ensure that your source files are preserved. The __User_Exporters__ folder is located within the exporters folder for the database, for example:

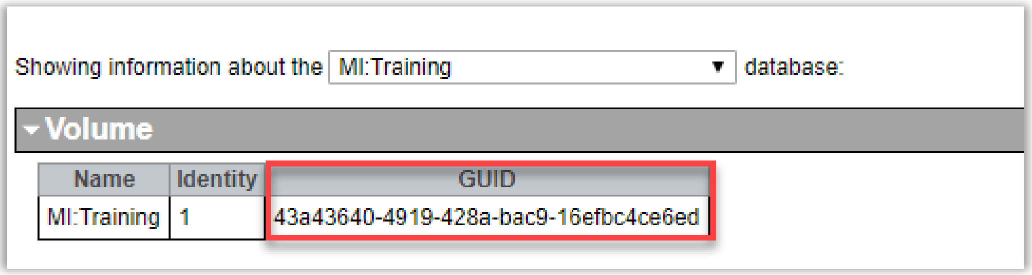
C:\Program Files\Granta\Granta MI\Server\exporters\MyTestDB\__User_Exporters__

After they have been loaded into the database, any changes made to exporter configuration (*.exp*) files in the __User_Exporters__ folder will be picked up by MI:Server after clicking **Refetch exporters** in MI:Admin. Changes to the XSLT will be picked up when the files are called during an export.

3.2 Finding GUIDs

In many cases, you will need to know the unique identity (GUID) or name of a database, a table, or an attribute) in order to refer to it from an exporter configuration file. Every object in the database has a unique ID, a number that is specific to the database. You can find out the identity of database objects from the **Object Identities** page in MI:Viewer. On this page, you can discover GUIDs for

A database:

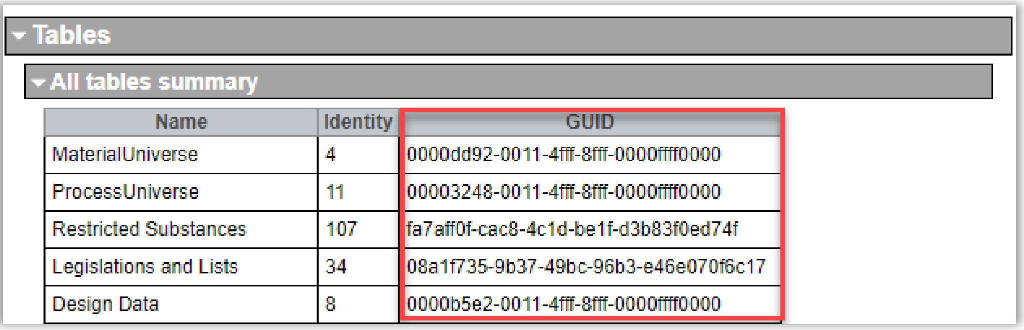


Showing information about the MI:Training database:

Volume

Name	Identity	GUID
MI:Training	1	43a43640-4919-428a-bac9-16efbc4ce6ed

All tables within a database:

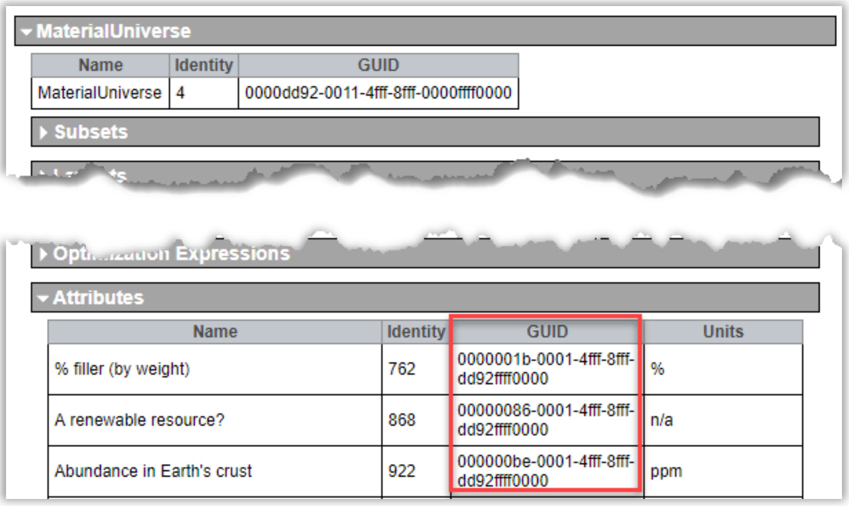


Tables

All tables summary

Name	Identity	GUID
MaterialUniverse	4	0000dd92-0011-4fff-8fff-0000ffff0000
ProcessUniverse	11	00003248-0011-4fff-8fff-0000ffff0000
Restricted Substances	107	fa7aff0f-cac8-4c1d-be1f-d3b83f0ed74f
Legislations and Lists	34	08a1f735-9b37-49bc-96b3-e46e070f6c17
Design Data	8	0000b5e2-0011-4fff-8fff-0000ffff0000

All attributes within a table:



Name	Identity	GUID
MaterialUniverse	4	0000dd92-0011-4fff-8fff-0000ffff0000

Name	Identity	GUID	Units
% filler (by weight)	762	0000001b-0001-4fff-8fff-dd92ffff0000	%
A renewable resource?	868	00000086-0001-4fff-8fff-dd92ffff0000	n/a
Abundance in Earth's crust	922	000000be-0001-4fff-8fff-dd92ffff0000	ppm

3.3 Exporter configuration file parsing and checking

When a user initiates an export from GRANTA MI, MI:Server searches for files with the *.exp* file extension and then parses and checks each matching file it finds.

Note that only standard XML special characters are allowed in XML content. For example, use `&#176;` instead of `&deg;`.

If an exporter fails either parsing or checking, an error message will be displayed to users with administrative privileges (see Section *12, Debugging exporters*). The problem must be resolved before MI:Server will display the exporter in the list of available exporters and allow users to use it.

Parsing and checking process

The *.exp* configuration file is parsed to check that it is valid XML and then the following checks are performed:

3. XSLT files specified in the `Transform` elements are checked to ensure that they exist and are valid XML.

   Note that GRANTA MI only supports XSLT v1.0 at this time.

4. The database tables are checked to ensure that they exist.
5. Data such as attribute names and parameter names are checked to ensure that they are not blank.
6. Other inconsistencies are flagged.
7. If any of these checks fail, then the exporter will not be shown in the list of available exporters in MI:Admin.

More checks (such as whether the attributes required by the exporter can be accessed by the current user) are performed during the export process.

4 Exporter config options - <ExporterConfig>

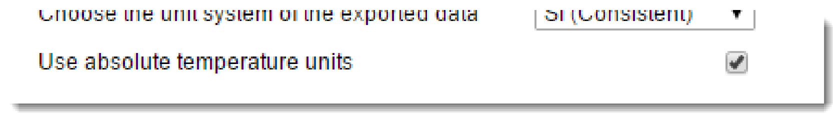
The root element of the exporter configuration file is the `ExporterConfig` element. A number of exporter properties are defined as XML attributes in the `ExporterConfig` element, for example:

```
<ExporterConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
AbsoluteUnitsOptional="true" UseAbsoluteUnits="true"
Name="ABAQUS_6_Lin_TempDep_Iso" MaxRecords="50"
xsi:noNamespaceSchemaLocation="../ConfigSchema.xsd">
```

`AbsoluteUnitsOptional`

Set this to true if the FEA package can accept both absolute and relative units.

- If *true*, the user will be given the choice of whether to have absolute units in the exported data; for example:



Choose the unit system of the exported data: SI (Consistent)

Use absolute temperature units: ✓

- If *false*, the value of `UseAbsoluteUnits` determines whether the export will contain absolute units.

`UseAbsoluteUnits`

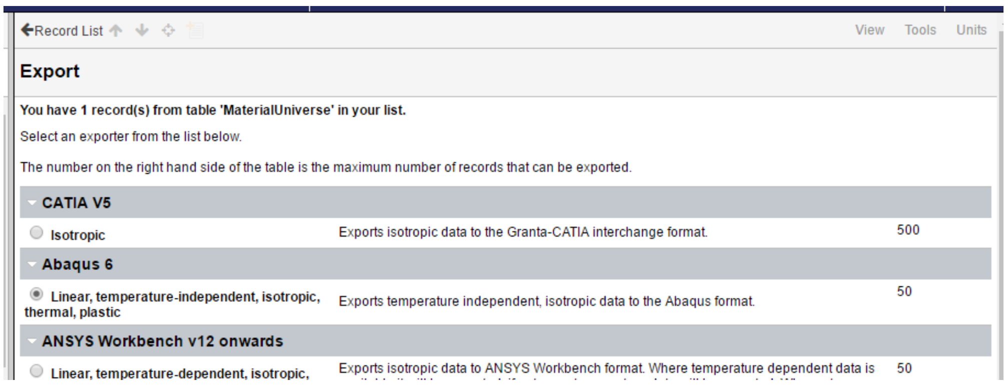
Determines whether units such as temperature are output as absolute (i.e. K or °R) or relative (°C or °F). If `AbsoluteUnitsOptional` is true, then the user can override this value.

`Name`

A name for this exporter (a string)

`MaxRecords`

Sets the maximum number of records that the FEA package can import. If this is less than 1, then no limit is assumed. This value is displayed on the Export page in MI:Viewer:



Export

You have 1 record(s) from table 'MaterialUniverse' in your list.

Select an exporter from the list below.

The number on the right hand side of the table is the maximum number of records that can be exported.

CATIA V5

Isotropic	Exports isotropic data to the Granta-CATIA interchange format.	500

Abaqus 6

[✓ Linear, temperature-independent, isotropic, thermal, plastic]	Exports temperature independent, isotropic data to the Abaqus format.	50

ANSYS Workbench v12 onwards

Linear, temperature-dependent, isotropic,	Exports isotropic data to ANSYS Workbench format. Where temperature dependent data is	50

The `ExporterConfig` element contains the following child elements which provide the information required by the GRANTA MI to export the data. These elements are described in detail in the following sections of this document.

Information category	**Description**	**Child Elements of <ExporterConfig>**
Exporter details	Defines the target CAE analysis package and material model type.	`Details`
Database, table and attribute export options	Specifies from which database, tables and attributes the data will be exported.	`Databases`
Transforms	Specifies names and locations of XSLT transforms that will convert the initial XML into a format suitable for the target package.	`Transforms`

5 Exporter details option - <Details>

5.1 Package, Model, Description

The `Details` element appears in the exporter configuration file under the root `ExporterConfig` element and is required. It defines the target CAE analysis package and material model type. It can include the following elements.

`Package`

Defines the target CAE analysis package. For example:

`<Package>Abaqus 6</Package>`

`<Package>CATIA V5</Package>`

In MI:Materials Gateway exporters, `Package` is used to identify the Gateway Host; see the *MI:Materials Gateway Installation & Configuration Guide* for more information on exporter configuration for Gateway.

`Model`

Defines the material model type. For example:

`<Model>Linear, temperature-dependent, isotropic, thermal, plastic</Model>`

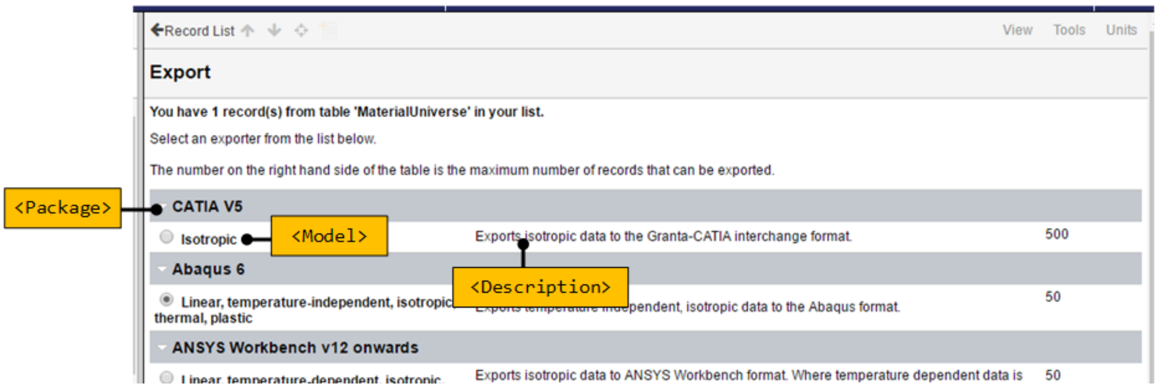
`<Model>Isotropic</Model>`

`Description`

Provides a description of the exporter. For example:

```
<Description>Exports isotropic data to the Granta-CATIA interchange format.
</Description>
```

The Package, Model, and Description all appear on the Export page in MI:Viewer:



Export

You have 1 record(s) from table 'MaterialUniverse' in your list.

Select an exporter from the list below.

The number on the right hand side of the table is the maximum number of records that can be exported.

CATIA V5

Isotropic | Exports isotropic data to the Granta-CATIA interchange format. | 500

Abaqus 6

Linear, temperature-independent, isotropic, thermal, plastic | Exports temperature-independent, isotropic data to the Abaqus format. | 50

ANSYS Workbench v12 onwards

Linear, temperature-dependent, isotropic | Exports isotropic data to ANSYS Workbench format. Where temperature dependent data is | 50

5.2 Applications in which the exporter will be used - <Applicability>

The applications in which an exporter will be used can be specified as part of the exporter configuration. This allows applications to filter the exporters available to users, so that that only relevant exporters are available within each application.

The `<Applicability>` element in the `<Details>` section of the exporter configuration file defines the target application for the exporter. For example:

```
<Applicability>
      <Tag>MIMaterialsGateway</Tag>
    </Applicability>
```

The `<Tag>` value can be one of:

- `MIMaterialsGateway`—available in MI:Materials Gateway applications only; not available in MI:Viewer
- `MIViewer`— available in the MI:Viewer application; not available in MI:Materials Gateway.

MI:Explore can be configured to include only exporters with the `MIViewer` applicability tag, only exporters with the `MIMaterialsGateway` applicability tag, or all exporters; see the *GRANTA MI Explore Configuration Guide* for details.

5.3 Output file options - <OutputFile>

The optional `OutputFile` element can be included in the `Details` element to specify the default output FEA filename.

When an FEA export output file is saved in MI:Viewer, the default filename *feaExport.txt*. is suggested. Most browsers allow the user to choose a different filename.

By including the `OutputFile` element in the exporter configuration file, the exporter author can exert some control over the suggested filename.

The `OutputFile` element may include three elements, specifying the file extension and file name:

`FileNameConvention`

Value	**Output filename is**
`Default`	feaExport
`RecordName`	The record full name
`TreeName`	The record tree name, that is, the name displayed in MI:Viewer's Contents tree. Note that when exporting more than one record, the TableName naming convention will be used instead of RecordName or TreeName.

Value	Output filename is
`TableName`	The name of the database table where the record is located followed by the number of records exported in parentheses – i.e. <table name> (N)
`Custom`	A filename defined with the Custom element.

`Extension`

The file extension (without a period).

`Custom`

A user-defined filename. (Only applies when `FileNameConvention` is set to `Custom`). This may contain the following special replacement tokens:

- `{recordname}` – The name of the first record in your record list
- `{treename}` – The tree-name of the first record in your record list
- `{tablename}` – The table name of the exported records
- `{numberofrecords}` – The number of records in your record list

Example 1: RecordName

```
<Details>
      <OutputFile>
             <FileNameConvention>RecordName</FileNameConvention>
             <Extension>out</Extension>
      </OutputFile>
</Details>
```

When exporting a single record named “Kevlar 149 aramid fiber”, the sample code above would result in a default output filename of: *Kevlar 149 aramid fiber.out*. When exporting 10 records, the sample code above would result in a default output filename of: *MaterialUniverse (10 records).out*.

Example 2: Custom

```
<Details>
      <OutputFile>
             <FileNameConvention>Custom</FileNameConvention>
             <Extension>dat</Extension>
           <Custom>{recordname}-from-{tablename}</Custom>
      </OutputFile>
</Details>
```

When exporting a single record named “Kevlar 149 aramid fiber”, the sample code above would result in a default output filename of: *Kevlar 149 aramid fiber-from-MaterialUniverse.dat*.

When exporting more than one record (assuming that “Kevlar 149 aramid fiber” was the first record in the list) then the above sample will still generate the same filename (because {recordname} and {treename} always refer to the first record only).

5.4 Export file encoding - <FileEncoding>

The optional `FileEncoding` element can be included in the `Details` element to specify the encoding for the output file.

By default, when a user chooses to save a file to disk, the file saved as UTF-8 encoded text. The file will not include the UFT-8 byte order mark (the characters \EF \BB \EF). Using this encoding scheme, the exported data will be able to be read by the majority of Windows applications.

However, in some circumstances it may be necessary to export the file in a different encoding. You can set the encoding of the output using the `FileEncoding` element. If omitted, the default UTF-8 encoding will be used.

To set the encoding you must specify the code page identity. Some common codepages are:

Code Page Identity	Name (and alternative names)
65001	UTF-8
1252	Windows-1252
65000	UTF-7
20127	US-ASCII, US, ASCII, cp367
1200	UTF-16LE, UTF-16, Unicode
1201	UTF-16BE, unicodeFFFE

Some encodings include a special series of characters at the start of the file or byte stream which denote the encoding and byte order of the following data. These characters are known as the byte order mark (or BOM for short).

Most common windows text editors (which can handle many different file encodings) will recognize the byte order mark and will not try to display it. Sometimes, if the byte order mark is not present, certain applications may not open the file correctly because it assumes the incorrect encoding for the file. (For example, Notepad may mistakenly assume that a file is US-ASCII encoded when in fact it is UTF-8 encoded and some non-ASCII characters may be displayed incorrectly).

However, some applications may not know to ignore these characters and treat them as data (you may see spaces or strange symbols at the start of the file).

You can specify whether or not the byte order mark is included in the file by setting the `IncludeBOM` attribute in the `FileEncoding` element.

Both the `CodePage` and `IncludeBOM` attributes are required: even if the code page has no byte order mark, the `IncludeBOM` attribute is still required.

```
<FileEncoding CodePage="1252" IncludeBOM="false" />
```

5.5 Floating point data options - <Precision>

The optional `Precision` element controls how floating point data is output in the Initial XML. It takes the following XML attributes:

- `RespectTrailingZeros` (Required, type: Boolean)

  If true, point and range data that were entered with trailing zeros will always be output with the same number of significant figures as they were entered with. This takes precedence over `MaximumSignificantFigures` for such data.

- `MaximumSignificantFigures` (Optional, type: integer)

  If `RespectTrailingZeroes` is true, this controls the maximum number of significant figures to use for all data / parameter values that were NOT entered with trailing zeros (they will be output with the same number of significant figures they were entered with). If omitted, no rounding will occur.

If `RespectTrailingZeroes` is false, `MaximumSignificantFigures` specifies the maximum number of significant figures to output for all floating point data. If omitted, no rounding will occur.

If the element is omitted then `RespectTrailingZeros` = false is assumed, and no rounding of floating point data will occur.

5.5.1 Examples

Let's consider some examples of mass data entered in kg. (The conversion factor of 1 kg = 2.20462 lb is assumed.)

Precision element omitted

Entered value (kg)	Output unit	Value in the Initial XML	
1	kg	1	Value had no trailing zeros, it is emited as the shortest string that can precisely represent the number (which in this case is simply '1')
1.0	kg	1	Value had trailing zeros but, since no `Precision` element is specified, `RespectTrailingZeroes` is assumed to be false.
1.23456789	kg	1.23456789	Since no `Precision` element is specified, `RespectTrailingZeroes` is assumed to be false.
1	lb	2.20462	
1.0	lb	2.20462	

<Precision RespectTrailingZeroes="true" MaximumSignificantFigures="4" />

Entered value (kg)	Output unit	Value in the Initial XML	
1	kg	1	

Precision element omitted

1.0	kg	1.0	Entered value had trailing zeros, output value has same # significant figures as the input value.
1.23456789	kg	1.235	Entered value had no trailing zeros, output value gets rounded to 4 significant figures.
1	lb	2.205	Entered value had no trailing zeros, output value gets rounded to 4 significant figures.
1.0	lb	2.2	Entered value had trailing zeros, output value has same number of significant figures as the input value.

<Precision RespectTrailingZeroes="false" MaximumSignificantFigures="4" />

Entered value (kg)	Output unit	Value in the Initial XML	
1	kg	1	
1.0	kg	1	Value had trailing zeros but, since `RespectTrailingZeroes` is false, we treat the value the same as '1'.
1.23456789	kg	1.235	Output value gets rounded to 4 significant figures.
1	lb	2.205	Output value gets rounded to 4 significant figures.
1.0	lb	2.205	Value had trailing zeros but, since `RespectTrailingZeroes` is false, we treat the value the same as '1kg' (with no trailing zeros)

<Precision RespectTrailingZeroes="false" />

Entered value (kg)	Output unit	Value in the Initial XML	
1	kg	1	
1.0	kg	1.0	Entered value had trailing zeros, output value has same number of significant figures as the input value.
1.23456789	kg	1.23456789	Entered value had no trailing zeros and no rounding of output value occurs.
1	lb	2.20462	Entered value had no trailing zeros and no rounding of output value occurs.
1.0	lb	2.2	Entered value had trailing zeros, output value has same number of significant figures as the input value.

5.6 Other options

Other output options are available, for example, allowing the output folder to be specified; see the ConfigSchema.xsd file for more information.

6 Database export options - <Databases>

The `<Databases>` element appears under the root `<ExporterConfig>` element and is required. The database export options are stored within a `<Database>` element that is the only child of the `<Databases>` element.

```
<Databases>
    <Database>
        [database options go here]
    </Database>
</Databases>
```

6.1 Database Key and GUID – <Database>

The `Database` element specifies the database from which data is to be exported. The database GUID and key can be specified if you know them. For example:

```
<Database Guid="ea6b62b6-4e8a-4f78-bc4b-cf51ec2351ff" dbKey="Metals_Test">
```

If you do not specify the GUID and key, the exporter configuration file will be modified automatically to include them when you add it to a database via MI:Admin.

6.2 Unit systems – <UnitSystems>

The `UnitSystems` element under `Database` specifies the unit systems available for exports from this database.

A GRANTA MI database has a number of ***unit systems*** defined as standard. These include common unit systems such as *Metric*, *US Customary*, and *UK Imperial*, as well as unit systems that are typically used for exporting material models, such as *SI (Consistent)*, *CGS (Consistent)*, *IPS (Consistent)*, and *FPS (Consistent)*.

To export unit system information, include a `UnitSystem` element in the configuration file, for example:

```
<UnitSystems>
    <UnitSystem Name="SI (Consistent)" />
    <UnitSystem Name="CGS (Consistent)" />
    <UnitSystem Name="FPS (Consistent)" />
    <UnitSystem Name="IPS (Consistent)" />
    <UnitSystem Name="mmNs (Consistent)" />
</UnitSystems>
```

6.3 Tables – <Tables>

The `Tables` element under `Database` defines the tables from which data will be exported.

```
<Tables>
    <Table Name="MaterialUniverse" Guid="0000DD92-0011-4FFF-8FFF-0000FFFF0000">
        ...
    </Table>
<Tables>
```

The schema allows more than one table to be defined in multiple `Table` elements. If you do this, the exporter will become flexible; it will be able to work with records from any of the tables you specify. However, records from different tables cannot be mixed in a single export operation, because the Record List in MI:Viewer can only contain records from one table.

7 Table export options – <Table>

The `Table` element identifies the table from which data will be exported. The table name or GUID must be specified, for example:

```
<Table Guid="E59175DD-8E12-4425-9540-2EC81454C423">
<Table Name="MaterialUniverse">
```

7.1 Exporting all attribute data from a table

Include an `AttributesIncluded` attribute on the `Table` element to export the data from all the attributes in the specified table. For example:

```
<Table Name="xxx" AttributesIncluded="AllIncludingMeta | AllExcludingMeta "/>
```

Options are:

- `AllIncludingMeta` – export the data from all attributes and metadata in the table.
- `AllExcludingMeta` – export the data from attributes only; do not export metadata attributes.

7.2 Exporting table version control information – <VersionControl>

To export record version control information for a table, include a `VersionControl` element in the `Table` element:

```
<VersionControl Include="true | false" />
```

For example:

```
<Tables>
   <Table Name="PolymerUniverse">
     <VersionControl Include="true" />
     <DataQuality Include="true" />
```

Note that the attribute data version number is always output to the initial XML and does not rely on this setting.

7.3 Exporting table data quality information - <DataQuality>

To export quality ratings for the data in a table, include the following element in the `Table` element:

```
<DataQuality Include="true" />
```

For example:

```
<Tables>
   <Table Name="PolymerUniverse">
     <DataQuality Include="true" />
```

If `DataQuality` is omitted, then no data quality information will be included in the initial XML.

7.4 Exporting access control information - <AccessControl>

If you use permission-based access control and the database has an access control schema, then access control information is exported automatically. Note that records and data are only exported if the user has permissions to see that data.

To suppress access control information for a table, include `<AccessControl Include="false" />` in the `<Table>` element. For example:

```
       <Tables>
         <Table Name="PolymerUniverse">
           <AccessControl Include="false" />
```

Regardless of this, the Access Control settings for the user always determine which records and attributes are included; this setting does not allow a user to subvert Access Control.

7.5 Exporting parameter information – <FullParameterDetails>

To export detailed information about all parameters available to functional attributes in the table, include the following lement in the `Table` element:

```
<FullParameterDetails Include="true" />
```

See also: *Section 8.4, Exporting functional data*.

7.6 Exporting attribute data - <DBAttributes>

The `DBAttributes` element specifies a list of the attributes to be exported in the initial XML. Within this element, you include one or more `DBAttribute` elements.

See Section *8, Attribute export options*.

7.7 Exporting user input – <UserData>

Users can be prompted to enter additional data which is exported along with the record data, for example to specify

- A thermal expansion data reference temperature
- The failure criterion for brittle materials.

7.7.1 User input scope – <General>, <Record>

User data may relate to the whole export operation, and/or to each record being exported.

Element	Description
`<General>`	Defines user input that relates to all of the data being exported. For example, a reference temperature when exporting thermal expansion data.
`<Record>`	Defines user input that relates to individual records being exported.

7.7.2 Input types - <Input>

The `Input` element, under the `General` element or `Record` element, describes the data that can be entered.

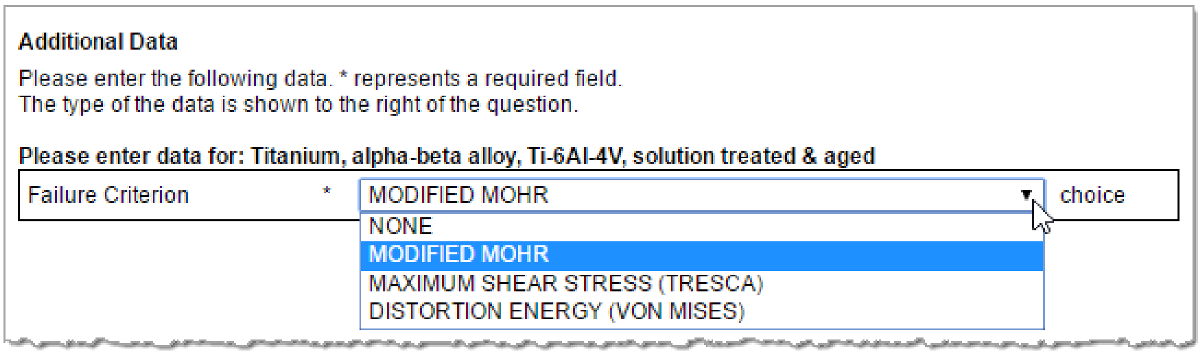
<Input xsi:type="*inputtype*" Value="*value*" Name="*name*" Optional="true|false">

XML Attribute	Description
`xsi:type`	The input data type. One of • UserInput – the user can enter any string value. • ChoiceInput – the user must select one from a list of options. See example below. • IntInput – the user must enter an integer value. • FloatInput – thr user must enter a valid floating point number. See example below.
`Value`	The default value of the input. The user may overwrite this.
`Name`	The input name
`Optional`	Specifies whether the input is optional (true) or mandatory (false). If Optional="false", the exporter software will ensure that the user enters a value.

ChoiceInput example

Available choices are specified using `Choice` elements as shown in this example, where each `Choice` defines the text that will appear on the options in the list box. For example:

```
<UserData>
  <Record>
    <Input xsi:type="ChoiceInput" Value="MODIFIED MOHR" Name="FAILURE_CRITERIA"
Optional="false">
      <Prompt>Failure Criterion</Prompt>
      <Choices>
        <Choice Text="NONE" />
        <Choice Text="MODIFIED MOHR" />
        <Choice Text="MAXIMUM SHEAR STRESS (TRESCA)" />
        <Choice Text="DISTORTION ENERGY (VON MISES)" />
      </Choices>
    </Input>
  </Record>
</UserData>
```



Additional Data

Please enter the following data. * represents a required field.
The type of the data is shown to the right of the question.

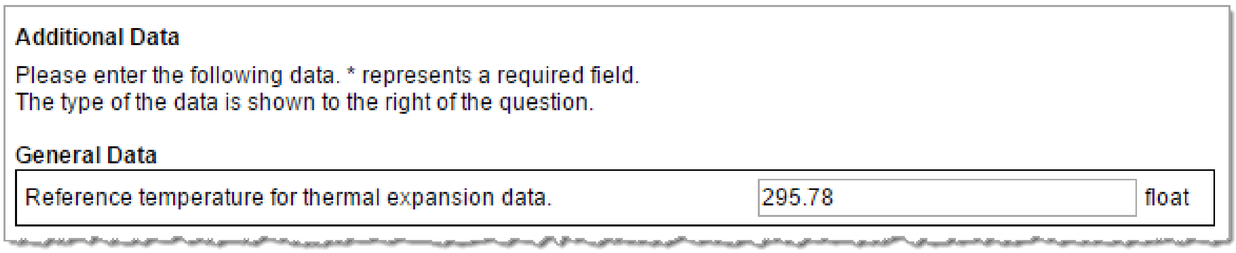
Please enter data for: Titanium, alpha-beta alloy, Ti-6Al-4V, solution treated & aged

Failure Criterion	*	MODIFIED MOHR	choice

Note that `ChoiceInput` can also be used to provide Boolean (e.g. yes/no) user input options.

FloatInput example

```
<UserData>
  <General>
    <Input xsi:type="FloatInput" Value="295.78" Name="RefTemp" Optional="true">
      <Prompt>Reference temperature for thermal expansion data. </Prompt>
    </Input>
  </General>
```



Additional Data

Please enter the following data. * represents a required field.
The type of the data is shown to the right of the question.

General Data

Reference temperature for thermal expansion data.	295.78	float

7.7.3 Text labels - <Prompt>

A `Prompt` element within an `Input` element specifies the text label on the input field on the Export page in MI:Viewer. You can see this in the two examples above.

8 Attribute export options

The `DBAttributes` element includes one or more `DBAttribute` elements which specify each attribute to be exported.

8.1 Identifying the attribute - <DBAttribute>

GRANTA MI attributes can be identified by their standard name, name, or GUID. The **standard name** should always be used, if defined. Attributes and meta-attributes that cannot be identified using a standard name may be identified using their name or GUID.

XML Attribute	Description

`StandardName`	The attribute's standard name, where one is defined. For example:
	`<DBAttribute StandardName="Tensile strength, ultimate" />`
	`<DBAttribute StandardName="Compressive strength, ultimate" />`
	`<DBAttribute StandardName="Tensile strength, yield" />`
	`<DBAttribute StandardName="Thermal expansion coefficient" />`

`Name`	The attribute name. For example: `<DBAttribute Name="Designation" />` However, note that the Name may change, and is not necessarily unique, and so using the standard name is recommended.
`Guid`	All database objects have a GUID (globally unique identifier) which can be used to identify them if no standard name is defined. For example: `<DBAttribute Name="Density" Guid="0000003F-0001-4FFF-8FFF-DD92FFFF0000"/>`

8.2 Using aliases

Aliases can be used to specify attribute and parameter names in the initial XML.

8.2.1 Attribute aliases

By default (when no alias is defined), the name of an exported attribute in the output file will be the same as its name in the source database. For example:

Attribute name	Tens. Yield Stress (L-Dir) with Temp.
Output name	Tens. Yield Stress (L-Dir) with Temp.

However, it is possible to specify an alternative name using an alias; this can be the Standard Name, or any other name. In this case, the specified alias name will be output, for example:

Attribute name	Tens. Yield Stress (L-Dir) with Temp.
Standard name	Tensile Strength, yield with temperature
Output (StandardNameAsAlias="true")	Tensile Strength, yield with temperature
Output (Alias="Yield strength w Temp")	Yield strength w Temp

Aliases are specified for GRANTA MI Attributes using the optional `Alias` or `StandardNameAsAlias` XML attributes on the `DBAttribute` element.

XML Attribute	**Description**
`Alias="newname"`	Use the specified string newname as the attribute name in the output file. This can be any string.
`StandardNameAsAlias="true"`	Use the attribute's standard name in the output file.

8.2.2 Parameter aliases - <ParameterAlias>

Aliases can be used for parameter names as well as attribute names, by including `Alias` or `StandardNameAsAlias` attributes in the `arameterAlias>` element.

For example, to output the parameter name "Stress Ratio" as "R ratio":

```
<ParameterAlias StandardName="R-Ratio" Alias="R Ratio" />
```

For example, to output the parameter name "Stress Ratio" with its standard name:

```
<ParameterAlias StandardName="R-Ratio" StandardNameAsAlias="true" />
```

Without an alias, the parameter name will be output as "Stress Ratio".

8.2.3 Summary of alias options

The effect of using different options to name output objects is summarized in the following table, with examples using the `ParameterAlias` element.

	Name of parameter	**Standard name of parameter**	**Name output in the export**
`<ParameterAlias StandardName="StandardTemp" StandardNameAsAlias="true" />`	Temperature	StandardTemp	StandardTemp
`<ParameterAlias StandardName="StandardTemp" />`	Temperature	StandardTemp	Temperature

	Name of parameter	**Standard name of parameter**	**Name output in the export**
`<ParameterAlias StandardName="StandardTemp" StandardNameAsAlias="true" Alias="ExplicitTemp" />`	Temperature	StandardTemp	**Error - cannot specify both**
`<ParameterAlias Alias="ExplicitTemp" />`	Temperature	StandardTemp	ExplicitTemp
`<ParameterAlias StandardName="StandardTemp" Alias="Temp" />`	Temperature	StandardTemp	Temp

8.3 Exporting meta-attributes

You can export all meta-attributes for a particular attribute by adding the XML property `IncludeAllMeta="true"` to the `DBAttribute` element. For example:

```
<DBAttribute StandardName="Tensile strength, ultimate" IncludeAllMeta="true"  />
```

This will export the attribute with the standard name “Tensile strength, ultimate” and all of its meta-attributes. Since meta-attribute names are not unique within the table (for example, two attributes can both have meta-attributes called “Notes”) the meta-attributes are given the alias (see Section *8.2*) of “*ParentAttributeName_MetaAttributeName*”.

If you want to export specific meta-attributes, you can do so by including a `ParentAttribute` element within the `DBAttribute` element. For example:

```
<DBAttribute Name="Notes">
  <ParentAttribute StandardName="Tensile strength, ultimate" />
</DBAttribute>
```

8.4 Exporting functional data

Functional data is data is stored as a collection of points and parameter values in the database. The format of the data that is output in the initial XML depends upon what is specified in the `DBAttribute` configuration element in the exporter configuration file.

The values of the parameters used for the interpolation are taken from the user’s current preferences, set on the **Export Options** page in MI:Viewer (see below).

For example, assume that the attribute Stress is a function of parameters Strain and Temperature. The exporter configuration file contains the line:

```
<DBAttribute Name="Stress" />
```

In MI:Viewer on the **Export Options** page, if the user clicks **Set Parameter Values**, they will be given the opportunity to set the values of Temperature and Strain. The value output in the initial XML will be a single `simpleValue` element that will contain a range value – calculated by interpolation.

Information about all the parameters available to functional attributes in the table can be exported with the data; see Section *7.5, Exporting parameter information*.

8.4.1 Grid data

Functional data can be stored in two separate formats: grid and series.

Grid functional data can be thought of as a multi-dimensional table of data. It is permissible to interpolate grid data to obtain a value for the attribute provided that we specify parameter values that are within the range covered by the grid.

When interpolating, if there are missing points around the point of interest, the next nearest points are used in the interpolation and the missing points are ignored. Note that we never extrapolate values.

For example, imagine that we have grid data for Stress vs Strain and Temperature. Our data is populated for values of Strain between 100 and 500 MPa (steps of 50 MPa) and Temperatures between 0 and 500 degrees Celsius (steps of 10 degrees Celsius). We are able to interpolate a value of Stress for any combination of the parameters Strain and Temperature provided that we choose a value of Strain between 100 and 500 MPa and a value of temperature between 0 and 500 degrees Celsius.

We may also restrict the value of just one parameter to obtain a new (interpolated) grid. For example, we could choose a fixed value of temperature and get a one-dimensional grid (Stress vs Strain) for that temperature.

Instead of specifying which parameters are fixed, the exporter configuration specifies which parameters are "free parameters" (i.e. not fixed).

8.4.2 Series data

Series data (also called a series graph) can be thought of as a collection of line graphs. For example, we could have series data for Stress vs Strain. This would consist of Stress values for a range of Strain values, which one could plot on a simple line graph. Now the attribute being measured may well depend upon other parameters – in our example, Stress also depends upon Temperature. The series graph for Stress would have several lines – each line is the Stress vs Strain curve for a given temperature.

In this case the parameter Temperature is referred to as a *constraint*. The only free parameter is Strain (the free parameter is often referred to as the X parameter since it is most often plotted on the X axis of charts).

Each line in the series graph may cover a different range of X and the points in different series need not be at the same values of X. Each line may also have a different number of points. Each line is essentially independent of the other lines in the graph; they just quantify the same attribute value.

The most important difference between series and grid data is that we cannot interpolate between constraint values in series data.

For example, let us consider a series graph of Stress vs Strain (as the free parameter) and Temperature (as a constraint). The graph may consist of three curves:

- A curve measured at 50 degrees Celsius for values of Strain between 100 and 400 MPa.
- A curve measured at 150 degrees Celsius for values of Strain between 200 and 300 MPa.
- A curve measured at 200 degrees Celsius for values of Strain between 100 and 500 MPa.

We may **not** infer a value of Stress for Temperature = 75 degrees Celsius, Strain = 250 MPa even though it may appear that we are able to. We may only interpolate a value of Stress at 250 MPa Strain for temperatures of exactly 50, 150 and 200 degrees Celsius. In other words we may only interpolate along a single line – not between lines.

This of course has important ramifications for exporting functional data stored as a series. If we are not exporting the entire graph, data will only be output if the parameter values are set to exactly match the constraint values of a line within the graph.

8.4.3 Free parameters

You can leave a parameter value unset by including the parameter in the `<FreeParameters>` element. For example:

```
<DBAttribute StandardName="Specific heat capacity with temperature">
    <FreeParameters>
        <Parameter StandardName="Temperature" />
    </FreeParameters>
```

Now, if the user clicks **Set Parameter Values** in MI:Viewer, they will not see ***Temperature*** in the list of parameters (since we are not restricting Temperature to a single value). The output will now contain a `complexValue` element, which contains either a `series` or `grid`element (depending on whether the original data was series or grid – see below).

8.4.4 Exporting functional data without interpolation

If you do not want any interpolation to occur, include `EntireGraph="true"` in the `DBAttribute` element, for example:

```
<DBAttribute Name="Stress" EntireGraph="true" />
```

This instructs the exporter to output the functional data in its entirety, that is, the whole of the graph will be output in the initial XML, no interpolation will be done and the data will output exactly as it is stored in the database (unit conversion not withstanding). The user does not get to set any parameter values.

Note that you cannot specify `EntireGraph="true"` and any free parameters, since `EntireGraph="true"` implies that all parameters are free.

8.4.5 Grid and Series data format in the initial XML

When EntireGraph="true"

If we specify `EntireGraph="true"`, then, by default, series graphs will be output in a `series` element and grid data will be output in a `grid` element. Note that both `series` and `grid` elements share the same inner format. The difference between the inner XML is that series graphs constraint parameters are marked with `isconstraint="true"` in the parameter elements.

The default format of the initial XML for functional data can generate quite large files. If you prefer a more concise format, include `GraphOutputType="complexValue2"` in the `DBAttribute` element, for example:

```
<DBAttribute Name="Stress" EntireGraph="true" GraphOutputType="complexValue2" />
```

This will produce an alternative structure in which the data values are collected into a single element as a delimited list, and parameter values are similarly collected into one element per parameter.

With no free parameters

If we do not specify `EntireGraph="true"` or any free parameters for functional data, then the software attempts to interpolate a (single) attribute value (based upon the user's chosen parameter values). This value will be output as a range in the initial XML (since each 'point' in both grid and series data is in fact a range value).

This value can only be calculated for grid data if:

- The parameter values lie within range for grid data.

This value can only be calculated for series data if:

- The parameter values exactly match the constraints values for a line within the series graph.
- The parameter value of the graph's X parameter lies within the range of the line.

With free parameters

Grid data: If we specify one or more free parameters for a functional attribute then data can only interpolate if the parameter values of the fixed parameters lie within the range of the grid data. If this is the case, then a `grid` element is written to the initial XML. There will only be one parameter value for each of the fixed parameters and the `parameter` element will have `isconstraint="true"`.

Series data: Data will only be output if:

- There is only one free parameter specified.
- The free parameter is the X parameter for the series graph.
- The values of the parameters exactly match the constraint values for a series in the graph.

If these conditions are met, then a `series` element is written to the initial XML. This will contain only one line from the series graph.

8.4.6 Warnings generated during export

During the export, it may not be possible to output any data for some functional attributes. Most of these errors are caused by the failure to interpolate functional data.

8.5 Exporting Embedded Equations and Logic (EEL) data

Embedded Equations and Logic (EEL) attribute values can be exported. The '*value'* exported for EEL data in the initial XML consists of two components:

- An `mfdMetaData` element that contains:
    - Information about the expression
    - Information about the curves defined in the database
    - The parameter extents (i.e. the domain over which the expression is valid)
- The numeric value of the EEL data. Like float functional data, the numeric value of EEL data can be exported as a single numeric value or as series data.

Note that user-defined curves (which are stored in the user's session) are not exported. Only the EEL data that is stored in the database will be exported.

8.5.1 Exporting EEL data as a single numeric value

If both of the following are true, then the EEL's expression will be evaluated at the parameter values specified in the "default curve":

- `EntireGraph` is not set to true in `DBAttribute` ***and***
- `DBAttribute` does not include any `FreeParameters`

If an EEL's data uses an expression that is not a function of any parameter (for example, it is a function of the record's attributes only), then it will always be exported as a single point irrespective of the settings in the exporter configuration file.

Parameter values

Note that the parameter values used to evaluate EEL data are likely to be different from those used to interpolate float functional data: all float functional data use the same set of parameter values for interpolation (these can be set by the user running the export). The parameter values used to evaluate EEL data are those stored in the database (in the default curve's parameter values).

If an MI:Viewer user has overridden these parameter values, it will not affect the exported output. Similarly, if the MI:Viewer user has created user-defined curves (which are stored in the user's session), these will not be exported. Only the data that is stored in the database will be exported.

8.5.2 Exporting EEL data as series data

If the `DBAttribute` element in the exporter configuration

- has `EntireGraph="true"` ***or***
- specifies a `FreeParameters` element (with exactly one Parameter)

then the EEL data will be exported as a set of curves. The data's default curve along with all other curves are exported.

Parameter values

Note that the parameter values used to evaluate EEL data are likely to be different from those used to interpolate float functional data. The parameter values for each exported curve are stored along with the curve in the database. User-defined curves (which are stored in the MI:Viewer user's session) will not be exported.

The free parameter

If no `FreeParameters` section is included, then the free parameter is taken to be the EEL data's default X axis parameter.

You can specify the free parameter in the `FreeParameters` element (see Section *8.4.3*). The EEL data must be a function of the specified free parameter, otherwise no data will be exported.

The free parameter extent

The minimum and maximum values of the free parameter are determined by the parameter's extent, which is stored in the EEL data.

By default, the expression is evaluated at 100 points between the minimum and maximum values unless:

- The parameter extent's minimum is exactly equal to the maximum – in which case the expression will be evaluated at this 1 point only
- The free parameter is a discrete parameter – in which case the expression will be evaluated at all the discrete values that are defined within the EEL data's free parameter's extent.

Controlling the number of points exported per curve

If the free parameter is not discrete, the number of points exported for each curve in the EEL data can be controlled by added the attribute `NumberOfPointsPerCurve` to the `DBAttribute` element, for example:

```
<DBAttribute Name="Stress" NumberOfPointsPerCurve="10" EntireGraph="true"/>
```

If you enter a number that is less than 1, it is ignored and the default value of 100 is assumed. If you enter 1 point, then the EEL data will be evaluated at the parameter extent's low value only.

Free parameter value spacing

If the free parameter’s “scale type” (as defined in the parameter’s definition in MI:Admin) is *Linear*, then the software chooses evenly-spaced values between the minimum and maximum value.

If the parameter’s scale type is *Logarithmic*, then logarithmically-spaced values between the minimum and maximum value are used. Note there is no facility to explicitly choose the free parameter values in the exporter definition.

9 Specifying the transforms - <Transforms>

The `Transforms` element element appears under the root `ExporterConfig` element and is required. It specifies the names and locations of the transform XSLT files that will perform the conversion from initial XML to a format suitable for the target FEA package.

For example:

```
<Transforms>
  <Transform xsi:type="XSLTransform" Filename="..\TESTtransform1.xslt"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
  <Transform xsi:type="XSLTransform" Filename=".\XSLT\TESTtransform2.xsl"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</Transforms>
```

The specified `Filename` is relative to the location of the exporter configuration file on disk; see Section *3, Exporter files*.

All exporters must include at least one transform. If you wish to upload an exporter file to GRANTA MI and export database records to initial XML without needing to write an XSLT transform, you can create and refer to an *identity transform* which returns the initial XML unchanged. The box below shows the complete source code for a working identity transform.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*">
    <xsl:copy-of select="." />
  </xsl:template>
</xsl:stylesheet>
```

10 Adding (importing) an exporter to a database

An exporter is added to a database by importing its configuration file and associated XSLT file(s) into the database for which it was written. This is done using MI:Admin; you will need administrative privileges to the database.

- If database security groups have been set for the database, you must be a member of the Admin database security group and a member of one of the system security groups.
- If database security groups have not been set, you must be a member of the Admin system security group.

10.1 To add a new exporter

To import a new FEA exporter to a database, use the Schema tool in MI:Admin:

1. On the **Edit Files** page, click on the **Exporters** tab and then click **Import Files.**
2. Once the files have been added to the database, click **Re-fetch Exporters**. This will sync the exporters in the database and in the *exporters* folder on the GRANTA MI server.

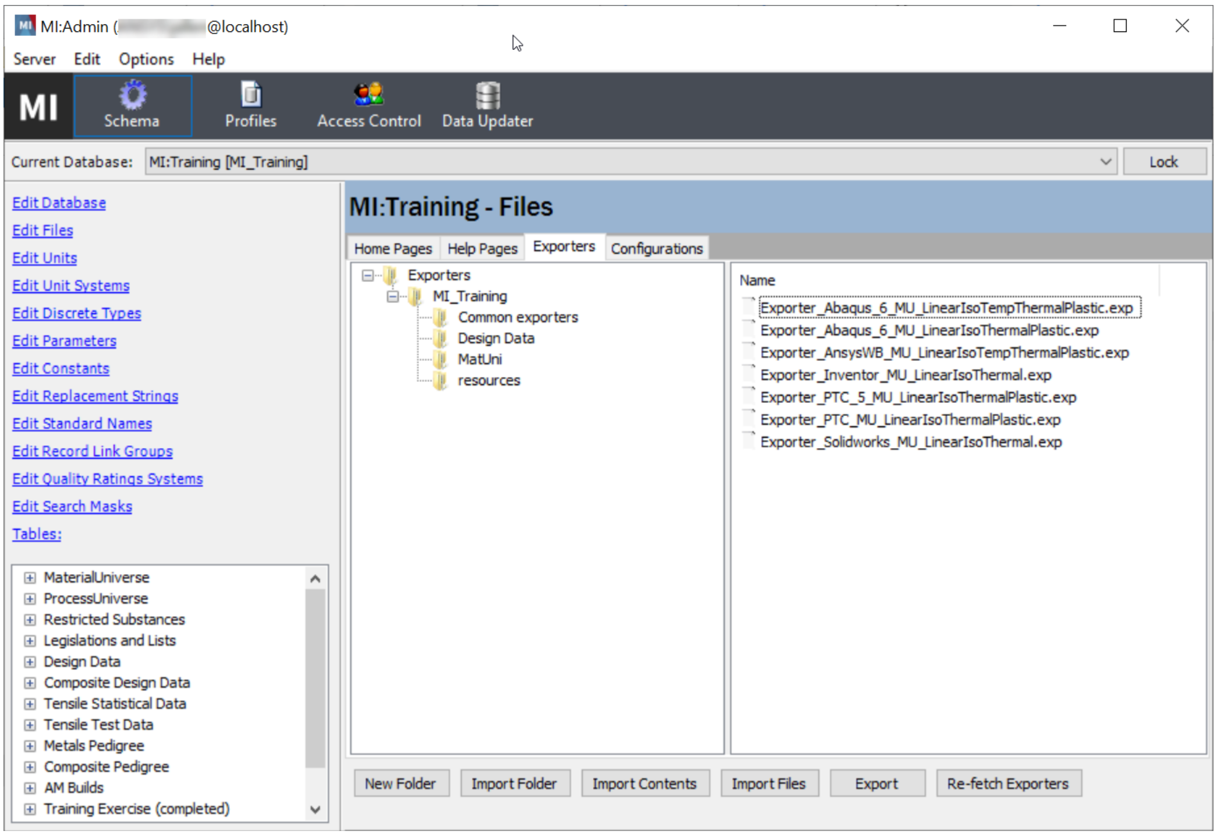
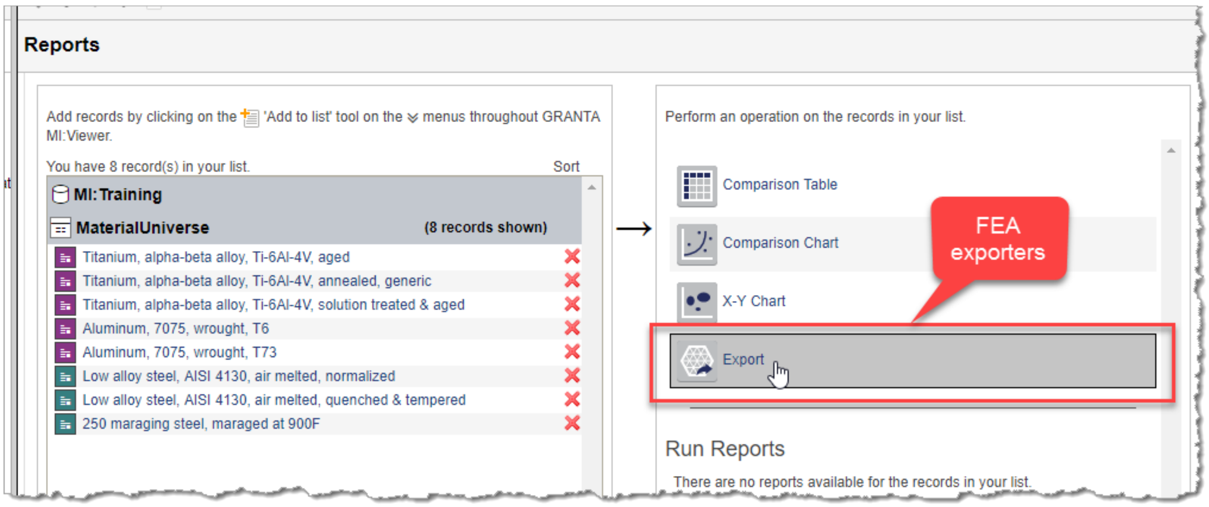


Figure 1. Files page, Exporters tab in the MI:Admin Schema tool

11 FEA Export features for end users

11.1 MI:Viewer

Exporters for the data in the Record List are accessed by clicking **Export** on the Reports page in MI:Viewer.



Reports

Add records by clicking on the 'Add to list' tool on the menus throughout GRANTA MI:Viewer.

You have 8 record(s) in your list.

MI:Training

MaterialUniverse (8 records shown)

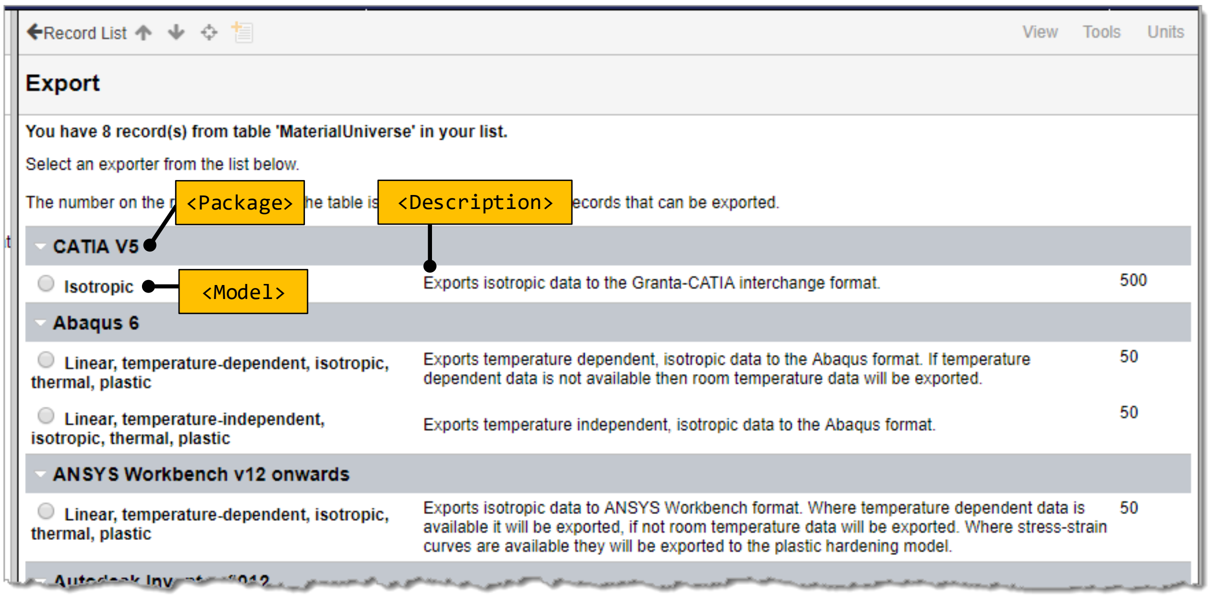
- Titanium, alpha-beta alloy, Ti-6Al-4V, aged
- Titanium, alpha-beta alloy, Ti-6Al-4V, annealed, generic
- Titanium, alpha-beta alloy, Ti-6Al-4V, solution treated & aged
- Aluminum, 7075, wrought, T6
- Aluminum, 7075, wrought, T73
- Low alloy steel, AISI 4130, air melted, normalized
- Low alloy steel, AISI 4130, air melted, quenched & tempered
- 250 maraging steel, managed at 900F

Perform an operation on the records in your list.

Run Reports

There are no reports available for the records in your list.

The available exporters are listed on the Export page. The package and model name, description,m and maximum number of records that can be exported, are all specified in the exporter configuration file.



Export

You have 8 record(s) from table 'MaterialUniverse' in your list.

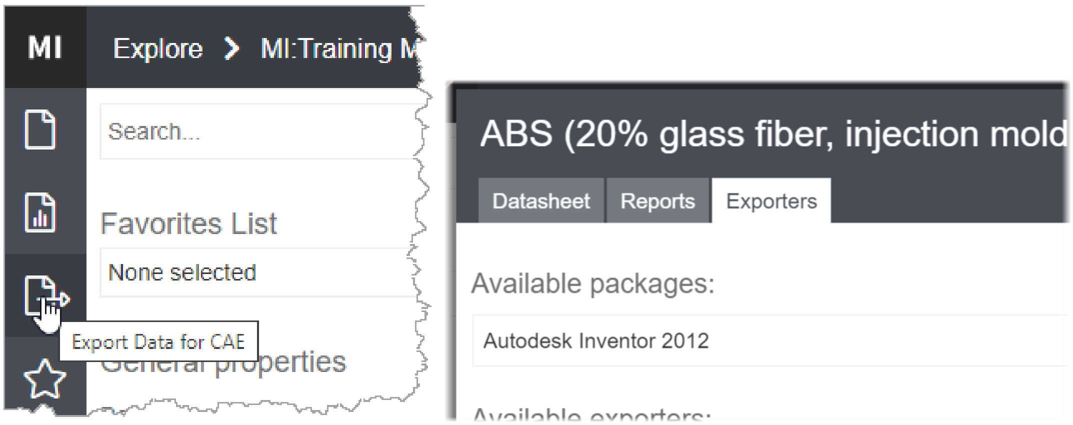
Select an exporter from the list below.

The number on the r he table is ecords that can be exported.

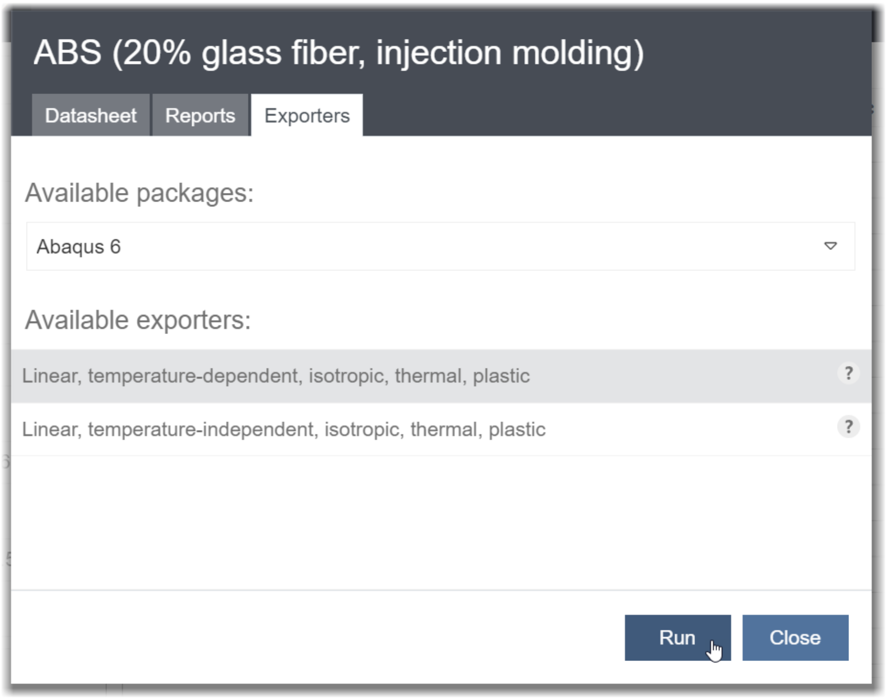
CATIA V5		
Isotropic	Exports isotropic data to the Granta-CATIA interchange format.	500
Abaqus 6		
Linear, temperature-dependent, isotropic, thermal, plastic	Exports temperature dependent, isotropic data to the Abaqus format. If temperature dependent data is not available then room temperature data will be exported.	50
Linear, temperature-independent, isotropic, thermal, plastic	Exports temperature independent, isotropic data to the Abaqus format.	50
ANSYS Workbench v12 onwards		
Linear, temperature-dependent, isotropic, thermal, plastic	Exports isotropic data to ANSYS Workbench format. Where temperature dependent data is available it will be exported, if not room temperature data will be exported. Where stress-strain curves are available they will be exported to the plastic hardening model.	50

11.2 GRANTA MI Explore

In the Explore application, users can access the available FEA exporters by clicking the **Export Data for CAE** option on the toolbar or by clicking on the Exporters tab in a datasheet:



Users can then choose the package and exporter they want to use to export data from the selected record(s):

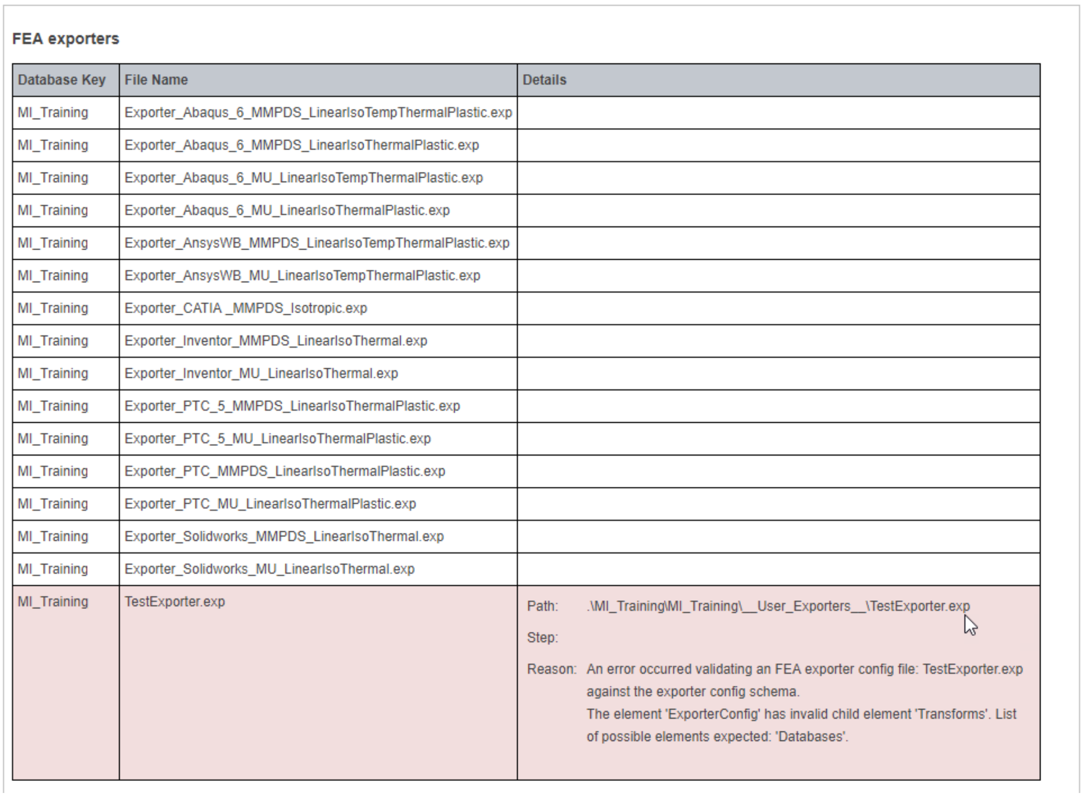

ABS (20% glass fiber, injection molding)
Datasheet
Reports
Exporters
Available packages:
Abaqus 6
Available exporters:
Linear, temperature-dependent, isotropic, thermal, plastic
Linear, temperature-independent, isotropic, thermal, plastic
Run
Close

12 Debugging exporters

Some debugging features are available in MI:Viewer for troubleshooting exporter problems. These features are only available to users with MI Administrator privileges.

12.1 View FEA exporter status in MI:Viewer

Admin users can see the status of all exporters on the Admin > FEA Exporters page in MI:Viewer. If any problems are detected with an exporter it will be highlighted with information shown in the *Details* column. For example:

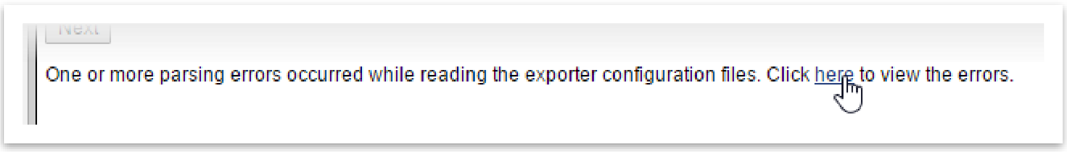


FEA exporters

Database Key	File Name	Details
MI_Training	Exporter_Abaqus_6_MMPDS_LinearIsoTempThermalPlastic.exp	
MI_Training	Exporter_Abaqus_6_MMPDS_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_Abaqus_6_MU_LinearIsoTempThermalPlastic.exp	
MI_Training	Exporter_Abaqus_6_MU_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_AnsysWB_MMPDS_LinearIsoTempThermalPlastic.exp	
MI_Training	Exporter_AnsysWB_MU_LinearIsoTempThermalPlastic.exp	
MI_Training	Exporter_CATIA _MMPDS_Isotropic.exp	
MI_Training	Exporter_Inventor_MMPDS_LinearIsoThermal.exp	
MI_Training	Exporter_Inventor_MU_LinearIsoThermal.exp	
MI_Training	Exporter_PTC_5_MMPDS_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_PTC_5_MU_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_PTC_MMPDS_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_PTC_MU_LinearIsoThermalPlastic.exp	
MI_Training	Exporter_Solidworks_MMPDS_LinearIsoThermal.exp	
MI_Training	Exporter_Solidworks_MU_LinearIsoThermal.exp	
MI_Training	TestExporter.exp	Path: .\MI_Training\MI_Training\__User_Exporters__\TestExporter.exp Step: Reason: An error occurred validating an FEA exporter config file: TestExporter.exp against the exporter config schema. The element 'ExporterConfig' has invalid child element 'Transforms'. List of possible elements expected: 'Databases'.

12.2 Parsing errors

If parsing or checking errors are detected in any exporters, a message and link are displayed below the exporter list on the Report page in MI:Viewer; clicking on the message should provide some information about the error. See also Section *3.2.*

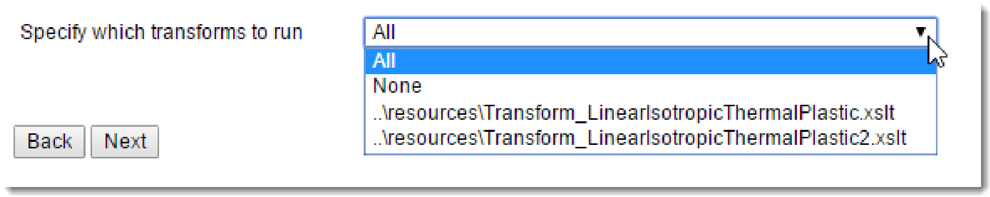


One or more parsing errors occurred while reading the exporter configuration files. Click here to view the errors.

12.3 XSLT transforms

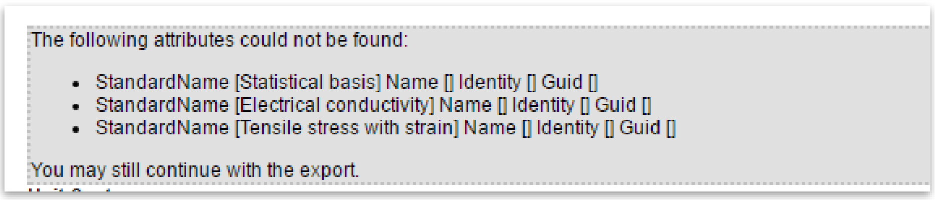
In MI:Viewer, Admin users see a list of available Transforms list on the **Export Options** page.

This allows you to choose which (if any) XSLT transforms are running during the export, and can aid debugging of XSLT transforms if there is more than one XSLT specified in the exporter. You may also specify that no XSLT transforms are run (and thereby get the export's initial XML).


Specify which transforms to run
All
All
None
..\resources\Transform_LinearIsotropicThermalPlastic.xslt
..\resources\Transform_LinearIsotropicThermalPlastic2.xslt
Back
Next

12.4 Missing attributes

If one or more attributes specified in the exporter could not be found in the database, they are listed on the **Export Options** page, for example:.



The following attributes could not be found:

- StandardName [Statistical basis] Name [] Identity [] Guid []
- StandardName [Electrical conductivity] Name [] Identity [] Guid []
- StandardName [Tensile stress with strain] Name [] Identity [] Guid []

You may still continue with the export.

13 The initial XML format for exported data

If you are writing an XSLT transform, you need to understand the initial XML format generated by GRANTA MI. This section briefly summarizes some features of the initial XML. For comprehensive reference information, consult the initialXML.xsd schema file, which is provided along with this document in the zip file *FEA_Schema.zip*.

The root element of the exported XML is the `<FEAExport>` element. It contains two child elements, `<definitions>` and `<data>`, whose contents are described in the subsections below.

```
<?xml version="1.0" encoding="utf-8"?>
<FEAExport decimalSeparator="." xmlns="http://www.grantadesign.com/Granta-
MI/exports" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <definitions>
    ...
  </definitions>
  <data>
    ...
  </data>
</FEAExport>
```

13.1 Definitions section

The `<definitions>` element contains various information about the export. It contains up to six child elements, whose contents described in the table below.

Element	Description
`<info>`	Information including: • the name and version of the exporter • the number of records exported • the database and table from which records were exported • the date and time of the export • the unit system chosen by the user, and • the user who performed the export.
`<parameters>`	Parameters and their values. (This element is present only if the exported data includes functional or multi-value point data.)
`<attributes>`	The names, types and units of all attributes included in the export, plus access control information if appropriate.
`<accessControlSchema>`	Describes the categories & permissions in the access control schema, and the exporting user’s membership of them (only if a permissions-based access control schema is defined for the database).

Element	**Description**
<tableDetails>	Name, access control & data quality information for the table.
<volumeDetails>	Name & access control information for the database.

13.2 Data section

The `<data>` element contains up to two child elements:

- a `<generalUserData>` element that contains *non-record-specific* data provided by the user, if applicable;
- a `<records>` element that contains the data exported from the GRANTA MI records, plus any *record-specific* data provided by the user, if applicable.

The `<records>` element contains one child `<record>` element per exported record. Each `<record>` element contains some XML attributes that specify the properties of the record:

XML Attribute	**Description**
`type`	The record type, either `record` or `generic`.
`recordguid`	The attribute name. For example: `<DBAttribute Name="Designation" />` However, note that the Name may change, and is not necessarily unique, and so using the standard name is recommended.
`historyguid`	All database objects have a GUID (globally unique identifier) which can be used to identify them if no standard name is defined. For example: `<DBAttribute Name="Density" Guid="0000003F-0001-4FFF-8FFF-DD92FFFF0000"/>`
`fullname`	The full name of the record, as it appears at the top of the datasheet.
`shortname`	The short name of the record, as it appears in the tree.
`gruid`, `recordHistoryIdentity`	Other internal identifiers that are rarely used in practice.

The child elements of each `<record>` element include:

- a `<userdata>` element that contains the record-specific user data, if applicable;
- an `<attributes>` element that contains data from the record, with one child `<attribute>` element per exported data value. The structure inside an `<attribute>` element depends on the attribute type, but generally the data will be stored in a `<simpleValue>` element for simple data, or a `<complexValue>` or `<complexValue2>` element for functional data;

- an `<accessControl>` element containing the access control permissions for the record, if applicable;
- a `<version>` element containing version control status information, if applicable.

Appendix A. Exporter configuration file example

This example exporter configuration file exports MaterialUniverse data, to be transformed into a material card for the Pro/ENGINEER Wildfire 5.0 ™/® package. It specifies suitable data to use with a temperature-independent, isotropic, thermal, plastic model, and so it extracts data including:

- two elastic parameters;
- tensile and compressive strengths (both yield & ultimate);
- thermal expansion, thermal conductivity and heat capacity data

```
<?xml version="1.0" encoding="utf-8"?>

<ExporterConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" AbsoluteUnitsOptional="false"
UseAbsoluteUnits="true" Name="B6DF9821-9D52-43D3-8C45-6582E4105BD7" MaxRecords="1"
xsi:noNamespaceSchemaLocation="../ConfigSchema.xsd">

  <Details>
    <Package>TEST Pro/ENGINEER Wildfire 5.0</Package>
    <Model>TEST Linear, temperature-independent, isotropic, thermal, plastic</Model>
    <Description>TEST Exports temperature-independent, isotropic data to the
      Pro/ENGINEER Wildfire 5.0 format. </Description>
    <!-- Setting the name and extension of the file that is exported.
         The file name is set to the name of the record on the tree. -->
    <OutputFile>
      <FileNameConvention>TreeName</FileNameConvention>
      <Extension>mtl</Extension>
    </OutputFile>
  </Details>

  <Databases>
    <Database Guid="43a43640-4919-428a-bac9-16efbc4ce6ed" DbKey=" MI_Training_10.0.0m">
      <!-- UnitSystems, lists unit systems the user can choose from. -->
      <UnitSystems>
        <UnitSystem Name="SI (Consistent)" />
        <!--<UnitSystem Name="CGS (Consistent)" Guid="00000006-0014-4FFF-8FFF-0000FFFF0000"/>
            <UnitSystem Name="IPS (Consistent)" Guid="00000003-0014-4FFF-8FFF-0000FFFF0000"/>
            <UnitSystem Name="FPS (Consistent)" Guid="00000002-0014-4FFF-8FFF-0000FFFF0000"/>
            <UnitSystem Name="Metric" />
            <UnitSystem Name="UK Imperial" Guid="00000007-0014-4fff-8fff-0000ffff0000"/>
            <UnitSystem Name="US Imperial" Guid="00000004-0014-4fff-8fff-0000ffff0000"/>-->
      </UnitSystems>
      <Tables>
        <!-- Using the Guid to identify the table. The Name attribute is ignored
         but is included as a guide when editing this file. -->
        <Table Name="MaterialUniverse" Guid="0000DD92-0011-4FFF-8FFF-0000FFFF0000">
          <!-- If VersionControl, DataQuality, or FullParameterDetails elements are
               omitted, then the default value of Include="false" is assumed. -->
          <VersionControl Include="true" />
          <DataQuality Include="true" />
          <FullParameterDetails Include="true" />
          <!-- If the AccessControl element is omitted, then the
               default value of Include="true" is assumed. -->
          <AccessControl Include="false" />

          <!-- ParameterAliases is optional. Here, Temperature will be called Temp -->
          <ParameterAliases>
            <ParameterAlias StandardName="Temperature" Alias="Temp" />
          </ParameterAliases>
          <UserData>
            <Record>
              <Input xsi:type="ChoiceInput" Value="MODIFIED MOHR" Name="FAILURE_CRITERIA"
Optional="false">
                <Prompt>Failure Criterion</Prompt>
                <Choices>
                  <Choice Text="NONE" />
                  <Choice Text="MODIFIED MOHR" />
                  <Choice Text="MAXIMUM SHEAR STRESS (TRESCA)" />
                  <Choice Text="DISTORTION ENERGY (VON MISES)" />
                </Choices>
              </Input>
            </Record>
          </UserData>
          <!-- Lists the attributes to be included in the initial XML -->
          <DbAttributes>
            <!-- Using standard name to identify the attributes -->
            <DBAttribute StandardName="Material name" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Tensile modulus" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Tensile strength, ultimate" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Compressive strength, ultimate" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Tensile strength, yield" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Thermal expansion coefficient" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Poisson's ratio" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Density" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Specific heat capacity" StandardNameAsAlias="true" />
            <DBAttribute StandardName="Thermal conductivity" StandardNameAsAlias="true" />
          </DbAttributes>
        </Table>
      </Tables>
    </Database>
  </Databases>

<!-- The XSLT version 1.0 files that will convert the initial XML to a
     format suitable for the FEA package -->
  <Transforms>
    <!-- The filename is relative to the location of this file on disk -->
    <Transform xsi:type="XSLTransform" Filename="..\TESTtransform1.xslt"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />
    <!-- This XSL file is in a subdirectory called XSLT -->
    <Transform xsi:type="XSLTransform" Filename=".\XSLT\TESTtransform2.xsl" />
  </Transforms>
</ExporterConfig>
```