Ansys GRANTA MI 2021 R1

GRANTA MI
Text Importer Reference

Contents

1 About the Text Importer

The GRANTA MI:Toolbox Text Importer plug-in is used to import data from text files into a GRANTA MI database.

Figure 1 Workflow to import text data

The Text Importer can place new records within the tree hierarchy of the destination table, and link new records to related records in the database. This is done through the use of templates. Each template provides the Text Importer with all the information it needs to read one type of text file and create new records, or add data to existing records, in a GRANTA MI database. This document describes the specification of Text Importer templates.

A common use of the Text Importer is to import test data. Many of the examples in this document are taken from a template for 'Instron Tensile Test' files.

Note: The Text Importer is unsuitable for importing very large (>1Gb) files.

1.1 Features of the Text Importer

- All data types can be imported into the database except tabular data. This includes functional data (grid, series, discrete), equations and logic data, metadata, pictures, and files for embedded media.
- Multiple files can be imported at once. Multiple files can be combined to create a single record.
- The Text Importer is optimized to handle a relatively small number of records, say twenty or less, although it can import thousands.
- Multiple records can be created from a single file where the file contains multiple tests or specimens.
- Trailing zeroes can be imported for point data and range data; see Section *7.1.3*.
- Unit conversion is carried out during the importing process using the unit conversions in the database.
- New data can be imported into an existing record or as new records. New functional data can be merged with existing functional data; see Section *6.3*.

- Static record links can be automatically generated using data from the text file to search for related records.
- Data links can be automatically generated using data from the text file to search for related records.
- The user can select a location for imported records (if using MI:Toolbox), or records can be automatically placed in the database using criteria given in the template. Automatic placement options allow imported records to be placed at the top (root) of the database tree, or within a defined tree structure, and also specify conflict resolution behavior if record name conflicts occur. See Section *6.3.3*.
- The locale of the data being imported can be specified in the template; see Section *3.2*.
- The Text Importer uses validation to check that files are correctly formatted before they are imported. For example, a template could be set up to ensure that only files that have a specified 'Test date' will be imported.

1.2 Using the Text Importer

The Text Importer can be run from:

- MI:Toolbox; see the MI:Toolbox Help for details
- the Remote Import web browser interface; see the Remote Import tool Help for details
- a command window, via a command-line interface; see *Appendix C*.

1.3 MI:Toolbox documentation

Reference documentation for the MI:Toolbox exporters and importers, including this document, can be found in any of the following locations:

- On your MI:Toolbox host, in the plugin installation folder (for example, C:\Program Files\Granta\GRANTA MI\Toolbox\plugins\Importers\Text\Documentation).
- In MI:Viewer: click on **Help > Reference Documentation**.
- Sign in on the GRANTA MI Support page and go to the *Documentation library* to access all of the reference documentation for the current and previous GRANTA MI releases.

2 Template overview

A Text Importer template instructs the Text Importer how to read text files containing data to be imported into a GRANTA MI database. A template contains information such as names of attributes it is designed to import the data into. Therefore, templates are specific to a table within a database.

Each Text Importer template is an XML file. Templates must be well-formed XML and must conform to the Text Importer schema definition (XSD) found in the default template folder location. Standard XML tools can be used to check that a template is well-formed and conforms to the schema.

A template contains XML elements which define information about the template itself, the data files is it intended to work with, how the data should be located within the database, and how to read the data from the import file. Templates make extensive use of **regular expressions** (see Appendix A) to extract data from the import file.

Every template contains a root `<Template>` element, which declares that the XML file should be validated against a schema and specifies where the schema is located:

```
<Template xsi:noNamespaceSchemaLocation="Schema.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

The `<Template>` element contains the following child elements which provide the information required by the Text Importer to interpret text data files. These elements are described in detail in the following sections of this document.

Template category	Description	Child Elements of <Template>
General template and record information	Description of the template and naming of the new record in the database.	`<General>` *
General file information	Specifies the text data files for import and how the data should be split into records.	`<Files>` *
Validation information	Ensures that the template is compatible with the current version of the Text Importer.	`<Validation>` *
Import options	Information on automatically generating links and placing the new records in the database.	`<Automation>`
Attribute data	Defines how to read data values from the import file.	`<Data>` *

* All required elements are marked with an asterisk.

3 Template and record information – <General>

The `<General>` element appears under the root `<Template>` element and is required.

```
<General>
  <TemplateName>
  <Description>
  <Locale>
  <Encoding>
  <Fullname>
  <Treename>
</General>
```

3.1 Template name and description

`<TemplateName>` and `<Description>` exist under `<General>`, and specify the name and description of the template. The data between the appropriate elements is treated as text data. This information is displayed in the **Select Template** dialog of the Text Importer plug-in.

```
<TemplateName>Instron Tensile Tests (Series IX, .txt and .rlt)</TemplateName>
<Description>
  Test Type: Tensile
  Machine Type: Instron Series IX
  Version: 8.12.00, 8.13.00, 8.15.00
  Units: Imperial
  File extensions: '.rlt' and '.txt'
</Description>
```

3.2 Locale – <Locale>

The locale of the files to be imported with the template can be specified by including a language and culture code inside a `<Locale>` element. This is used to determine whether a comma or a period is recognized as representing the decimal separator. For example, to specify Brazilian Portuguese:

```
<General>
  <TemplateName> Test Text Importer </TemplateName>
  <Description>Text importer test</Description>
  ...
  <Locale>pt-BR</Locale>
</General>
```

A useful reference for information about language codes and language support in Windows can be found on this Microsoft Go Global Developer Center web page: National Language Support (NLS) API Reference.

3.3 Character encoding – <Encoding>

The `<Encoding>` element specifies the character encoding format of the text files to be imported.

For example, to set the encoding to *Western European (Windows)*, a common encoding for Latin-based languages including French, German, Portuguese and Spanish:

```
<Encoding>windows-1252</Encoding>
```

For Cyrillic languages including Russian, use *Cyrillic (Windows*) encoding, for example:

```
<Encoding>windows-1251</Encoding>
```

If no `<Encoding>` element is present in the template, or if the encoding specified there is invalid, the default behavior is to assume the data files are UTF8-encoded. If you find that special characters such as the degree symbol (°) are displayed incorrectly after a data import, you should check that the encoding specified in the import template matches the encoding of the source data files.

3.4 Record name – <FullName>

The record name is derived data (see *Section 7* for more information on attributes and the record name). The `<FullName>` element tells the Text Importer how the name for the record should be created and is required. It can only use attributes declared in the template. If the record name fails to generate, the user is asked to enter a new record name. This example generates the name *filename* : *Specimen ID*:

```
<FullName>
  <Item>FILENAME</Item>
  <Item>Text : </Item>
  <Item>Specimen ID</Item>
</FullName>
```

where

- FILENAME is an attribute in the template and returns the import text data file name.
- Text is an attribute in the template that contains a text string. In this case the attribute contains the string ‘ : ’ and adds this to the record name.
- Specimen ID reads the value of the template attribute Specimen ID from the text file.

3.5 Short name – <TreeName>

The short name is derived data (see *Section 7* for more information on attributes and the record name). The `<TreeName>` element tells the Text Importer how the short name for the record should be created and is optional. If it is omitted, the short name will default to the record name. If the short name fails to generate properly, it will also default to the record name. If the short name

already exists in the table (it must be unique in the folder), the user is asked to enter a new record name.

Example of the template XML:

```
<TreeName>
  <Item>Specimen ID</Item>
</TreeName>
```

4 General file information – <Files>

The `<Files>` element appears under the root `<Template>` element and is required.

The data to be imported can be read from one or more text files. All these files must be in the same directory.

```
<Files>
  <TextFile name="file1">
    <Extension>
    <Validate>
    <UniqueIdentity>
    <SpecimenDelimiter>
    <SeriesDelimiter>
  </TextFile>
  <TextFile name="file2">
   …
  </TextFile>
</Files>
```

4.1 Import file information – <TextFile>

The `<TextFile>` element contains a list of file information for each type of import file. The number of `TextFile` elements in the list will determine the number of files that are needed. The order of the child elements of the `TextFile` element is enforced.

Even if only one file is to be imported, this still must be included in the list, as it is used to verify the correct file has been selected. The file information in the template includes:

- File validation
- Record delimiting information
- Series delimiting information
- File identity

```
    <TextFile name="fileCode">
      <Extension>extension</Extension>
      <Validate>
        <Attribute>Data element name</Attribute>
        <Value>value1</Value>
        <Value>value2</Value>
        <Value>value3</Value>
      </Validate>
      <UniqueIdentity>Data element name</UniqueIdentity>
      <SpecimenDelimiter>regular expression</SpecimenDelimiter>
      <SeriesDelimiter>regular expression</SeriesDelimiter>
    </TextFile>
```

4.1.1 Attributes

Attribute	Required/ Optional	Description
name	Required	The name that will be assigned to this file to identify it during import. This is referred to as the 'FileCode' in the rest of the template.

4.1.2 Elements

Element	Required/ Optional	Description
<Extension>	Required	The allowed extension for this file. See Section *4.2*.
<Validate>	Optional	Reads data from the file and only allows files with a defined set of values to be imported. See Section *4.2*.
<UniqueIdentity>	Optional	This is used to identify the set this file relates to. If omitted, the file name is used to identify the set. See Section *4.3*.
<SpecimenDelimiter>	Optional	A regular expression that is used to delimit the text file into the individual specimens. See Section *4.4*.
<SeriesDelimiter>	Optional	A regular expression that is used to delimit data into individual series for functional data. Section See *4.5*.

4.1.3 Example

```
<Files>
  <TextFile name="fileCode1">
    ...
  </TextFile>
  <TextFile name="fileCode2">
    ...
  </TextFile>
</Files>
```

4.2 File validation – <Extension> and <Validate>

The file can be validated in several ways: by the file extension, validation criteria or a combination of both. This information allows the template to identify the file.

The `<Extension>` element allows you to specify an allowed extension for this file. The `<Validate>` element allows you to specify validation criteria. Both elements exist under the `<TextFile>`

element. One of the two elements is required. Once one element is specified, the other becomes optional.

```
<Extension>file extension</Extension>
<Validate>
   <Attribute> Data element name </Attribute>
   <Value> value</Value>
</Validate>
```

The file validation `<Validate>` operates on two levels.

a) First, it ensures that the file types being imported are for use with the selected template. The Text Importer compares the attributes in the text file against the list of allowed attributes in each `<Attribute>` element in the template and validation fails if it cannot find the attributes in the text file.

b) Second, it ensures that the attributes being imported have the expected values. The Text Importer compares the values of an attribute in the text file against a list of allowed values for the attribute and validation fails if it cannot find the values in the data file.

Multiple `<Validate>` elements can be included in the template. All attributes must pass for the import to continue.

4.2.1 Example

This will only allow files which have the *.rlt* extension.

```
    <TextFile name="file1">
      <Extension>rlt</Extension>
      <Validate>
        <Attribute>File Units</Attribute>
        <Value>US Customary</Value>
      </Validate>
      <Validate>
        <Attribute>Method</Attribute>
          <Value>20</Value>
          <Value>25</Value>
          <Value>30</Value>
      </Validate>
    </TextFile>
```

Examples of use are:

- Limiting the template to a specified test machine.
- Checking that the selected files correspond to a known version number with which the template has been tested.

In the above example, a selected file would be checked to see if the file extension was *rlt*. The Text Importer would then attempt to read the File Units and Method data. If this data is found, it will check it matches one of the listed values. If all stages pass, the selected file will then be labeled as *file1*. If there are several data files, then it will try to validate the file for the other file information. If the file is not valid for any of the listed file information, it will be ignored by the Text Importer.

4.3 Identifying related files – <UniqueIdentity>

The `<UniqueIdentity>` element exists under the `<TextFile>` element and is optional. It is relevant when the Text Importer is used to import data from multiple text files. The element is used to identify related files, for example, files for the same set of test results.

The `<UniqueIdentity>` element contains the name of one of the `Data` elements in the template. This will be used to read the data from the file. The `Data` element must be present for every file that is to be imported. If a value is found, then this will become the file's 'identity'. If it returns no data, then the Text Importer will ignore the file. Where the files have the same identity, they will be marked as being from the same 'set' (see Example 1).

If the `<UniqueIdentity>` element is not present, the Text Importer will default to using the file name as the 'identity' (see Example 2).

4.3.1 Example 1

```
<TextFile name="file1">
  <Extension>txt</Extension>
  <UniqueIdentity>Specimen ID</UniqueIdentity>
</TextFile>

<TextFile name="file2">
  <Extension>rlt</Extension>
  <UniqueIdentity>Test ID</UniqueIdentity>
</TextFile>
```

In this example, the set of text files must consist of two files, one with the .txt extension and one with the *.rlt* extension e.g. *filename1.txt* and *filename2.rlt*. The file names do not need to be identical. The *Specimen ID* in the *filename1.txt* must exactly match the *Test ID* in *filename2.rlt* for them to be recognized as coming from the same set of test results. The *filename1.txt* will be labeled with the FileCode `file1` and *filename1.rlt* will be labeled with the FileCode `file2`.

4.3.2 Example 2

```
<TextFile name="file1">
  <Extension>txt</Extension>
</TextFile>
<TextFile name="file2">
  <Extension>rlt</Extension>
</TextFile>
```

In this example without `<UniqueIdentity>`, the set of text files must consist of two files with ***identical*** file names, one with the .txt extension and one with the *.rlt* extension e.g. *filename.txt* and *filename.rlt*.

The *filename.txt* will be labeled with the FileCode `file1` and *filename.rlt* will be labeled with the FileCode `file2`.

4.4 Record-delimiting information – <SpecimenDelimiter>

This element exists under the `<TextFile>` element and is optional. It is relevant when the Text Importer is used to import test data.

Some test files contain data on more than one specimen. The `<SpecimenDelimiter>` element contains the information for the Text Importer on how to split the data in the file to create one record for each specimen. This is done using a regular expression. See *Appendix A* for more information on regular expressions.

When a text file contains data on more than one specimen, there may be some general data that applies to all specimens in the file e.g. the date of the test, and some data that only applies to the individual specimen e.g. the time of the test. The Text Importer can be instructed to look for an attribute value from the general data or specimen-specific data with the `<Multiple>` element; see Section *7.2.2*.

4.4.1 Example

```
<TextFile name="file1">
  <Extension>txt</Extension>
  <SpecimenDelimiter>Specimen:[#/%e0-9 \.\-]+</SpecimenDelimiter>
</TextFile>
```

In the 'Instron' example, the start of a new specimen can be identified by "Specimen: 1" where the specimen number changes for each specimen. The regular expression for this is:

```
Specimen: [0-9]+
```

The Text Importer searches the *filename.txt* file using this regular expression and splits the file using the matched text as the delimiter.

```
File
Specimen Record 1
Specimen Record 2
```

4.5 Series-delimiting information – <SeriesDelimiter>

This element exists under the `<TextFile>` element and is optional. It is relevant when the Text Importer is used to import functional data, and is structured in a similar manner to the specimen-delimiting information.

The `<SeriesDelimiter>` element contains the information for the Text Importer on how to create one graph for each set of functional data in the file.

The regular expression element contains the instructions on how to split the functional data into the individual series. The delimiting expression is only applied to the functional data.

The Text Importer can be instructed that a graph consists of multiple series with the `<Series>` element. See Section *7.1.3* for more information.

4.5.1 Example

```
<TextFile name="file1">
  <Extension>txt</Extension>
  <SpecimenDelimiter>Specimen:[#/%e0-9 \.\-]+</SpecimenDelimiter>
  <SeriesDelimiter>Cycle Number=[0-9\s]+</SeriesDelimiter>
</TextFile>
```

Series delimiting can be used in conjunction with specimen delimiting. The specimen information is used to split the data into records, and for each record, the series information is used to create the individual series for one graph. In this case, the individual series will be looked for within the delimited specimen text.

```
      File
      Specimen Record 1
            Functional Data Graph
                  Series 1
                  Series 2
      Specimen Record 2
            Functional Data Graph
                  Series 1
                  Series 2
                  Series 3
```

An example of the use of functional data is in cyclic fatigue tests. Each individual series in a graph represents a cycle.

5 Validation information – <Validation>

The `<Validation>` element exists under the root `<Template>` element and is required.

```
<Validation>
  <Version>
  <ValidTableNames>
    <TableName>
    <TableName>
  </ValidTableNames>
</Validation>
```

There are two validation processes carried out by the Text Importer:

- Version validation – `Version`
- Table validation – `ValidTableNames`

5.1 Version validation – <Version>

The `<Version>` element appears under the `<Validation>` element and is required.

The Text Importer plug-in has a version number associated with it, which is used to validate a template's compatibility with the current Text Importer schema. MI:Toolbox will attempt to upgrade templates with older version numbers (from v2.0 onwards). If the user chooses not to let MI:Toolbox automatically upgrade a template, it will not be available to import with. Remote Import and the Text Importer command-line interface are able to use all template files which conform to the current schema, irrespective of their `<Version>` element's value.

```
<Validation>
    <Version>2.9</Version>
</Validation>
```

Template version and GRANTA MI compatibility

Text Importer template version	GRANTA MI version
2.5	Version 7 Version 7 Update 1 and Update 2
2.6	Version 7 Update 3 and later
2.7	Version 8, Version 8 Update 1 Version 8.1 and all v8.1 updates Version 9.0
2.9	Version 9.0 Update 1 and later

5.2 Table validation – <ValidTableNames>

The `<ValidTableNames>` element exists under the `<Validation>` element and is optional.

It contains a list of the tables which the template can be used to import data into. This element is only relevant in templates without auto-placement settings (see 6.3), and therefore only when importing using MI:Toolbox.

The Text Importer is limited to only allowing data to be imported into one table at a time. After selecting a table to import to in MI:Toolbox, only templates which are valid for that table will be available.

```
<Validation>
  <Version>2.9</Version>
  <ValidTableNames>
    <TableName>Fatigue Test Data</TableName>
    <TableName>LCF Test Data</TableName>
  </ValidTableNames>
</Validation>
```

6 Import options – <Automation>

The `<Automation>` element appears under the root `<Template>` element and specifies options for:

- Data linking – link data in the imported records to attributes in existing records. See 6.1.
- Record linking – link the imported records to existing records in the database based on their record names, GUIDs, or particular attribute values. See 6.2.
- Auto-placement of data into a particular database, table, folder, record, and subset, and the strategies used to resolve naming conflicts between incoming data and existing records in the database. See 6.3.
- Placement of records into a specific folder hierarchy within the destination database. See 6.4.

The `<Automation>` element is optional for templates used in MI:Toolbox, but must be present, containing `<AutoPlacement>` settings, in templates used by Remote Import and the command-line interface.

```
<Automation>

    <AutoDataLink>
       <AutoDataLinkConfig>
           <AttributeName>
           <ClusterName>
           <TableName>
           <SourceAttribute>
           <DestinationAttribute>
           <LinkCriterion>
        </AutoDataLinkConfig>

    <AutoLink>
        <AutoLinkConfig>
            <ClusterName>
            <SourceAttribute>
            <SourceValue>
            <DestinationAttribute>
            <DestinationProperty>
            <LinkCriterion>
        </AutoLinkConfig>
    </AutoLink>

    <AutoPlacement>
        <DBKey>
        <TableName>
        <SearchAttribute> | <SearchValue>
        <MatchAttribute> | <MatchProperty>
        <IsChild>
        <CreateIfNotFound>
        <FilterName>
        <ForceToRoot>
        <FunctionalMergeOption>
        <ConflictResolution>
```

```
    </AutoPlacement>

    <TreeHierarchy>

</Automation>
```

6.1 Data links – <AutoDataLink>

The optional `<AutoDataLink>` element contains information on how data should be linked after it is created in the database. A single link configuration `<AutoDataLinkConfig>` may result in multiple links being created. Multiple link configurations can be used in a template. The order of the child elements of the `AutoDataLinkConfig` element is enforced.

```
<AutoDataLink>
  <AutoDataLinkConfig>
    <AttributeName>Attribute Name</AttributeName>
    <ClusterName>Cluster Name</ClusterName>
    <TableName>Table Name</TableName>
    <SourceAttribute>Source Attribute</SourceAttribute> or
      <SourceValue>Source Value</SourceValue>
    <DestinationAttribute>Destination Attribute</DestinationAttribute> or
      <DestinationProperty>Destination Property</DestinationProperty>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoDataLinkConfig>
</AutoDataLink>
```

6.1.1 Elements

The table below describes the elements you can define within an `<AutoDataLinkConfig>` element. Note that:

- the **source** table is the table into which the Text Importer is importing data.
- the **destination** table contains the record(s) to which the imported record will link.
- the data link group must be between the **source** and **destination** tables but can 'belong' to (i.e. be defined on an attribute in) either. If it belongs to the source table, it is described as being in the **forward** direction, and vice versa.

Element	Required/ Optional	Description
`<AttributeName>`	Required	The name of the attribute (in the database) to which the data link group belongs. This attribute can be in either the source or destination table.

Element	Required/ Optional	Description
`<ClusterName>`	Required	The 'forward' name of a data link group (in the database) which belongs to the attribute specified by `<AttributeName>`. When `<AttributeName>` specifies an attribute in the **source** table, the combination of `<AttributeName>` and `<ClusterName>` uniquely identify the data link group. When `<AttributeName>` specifies an attribute in the **destination** table, the `<TableName>` element is also required to identify this table. Both the attribute and the data link group must already exist in the database
`<TableName>`	Optional	`<TableName>` specifies the table the data link group belongs to when this is the **destination** table. The value of `<TableName>` is used in combination with `<AttributeName>` and `<ClusterName>` to identify the data link group in the database. The table must already exist in the database, and the data link group must link to the **source** table.
`<SourceAttribute>` `<SourceValue>`	Required (one of)	A source attribute <SourceAttribute> or a source value <SourceValue> must be specified. This value is used to search the **destination** table in the database for the **record** to make the link to. `SourceAttribute` The value of the attribute in the record (which has been imported from the text file) is used as the search value. If no value is present for the attribute, no links are made. `SourceValue` The value in the template is used as the search value.

Element	Required/ Optional	Description
`<DestinationAttribute>` `<DestinationProperty>`	Required (one of)	Specifies where to look for the search value. `<DestinationAttribute>` is one of the attributes in the destination table. `<DestinationProperty>` is an intrinsic property of the records in the destination table, one of TreeName, FullName, RecordID or GUID. If a match is found, a link is created between the newly-imported record and the record in the destination table that contains the matching value, via the attributes defined in the data link group in MI:Admin.
<LinkCriterion>	Required	Specifies the criterion for a match. The only option is 'Equals'.

Searching and matching work in the same way, whether the linking direction is forward or reverse, for example, the `<DestinationAttribute>` always refers to an attribute in the destination table.

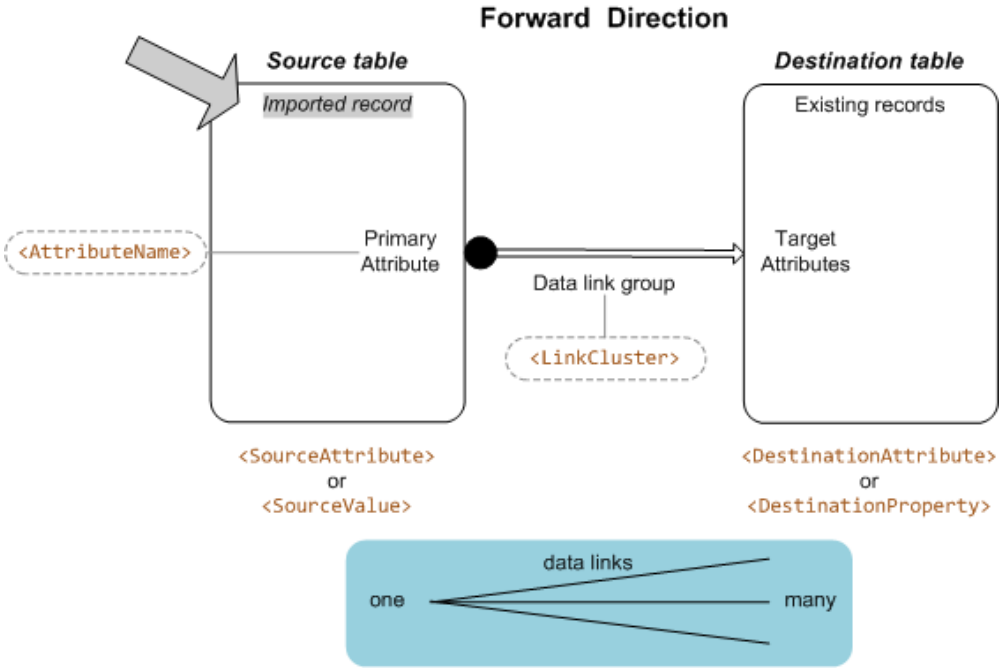


Figure 2 Data links in the forward direction

`<Element>` indicates the element is used to uniquely identify the data link group in the database.

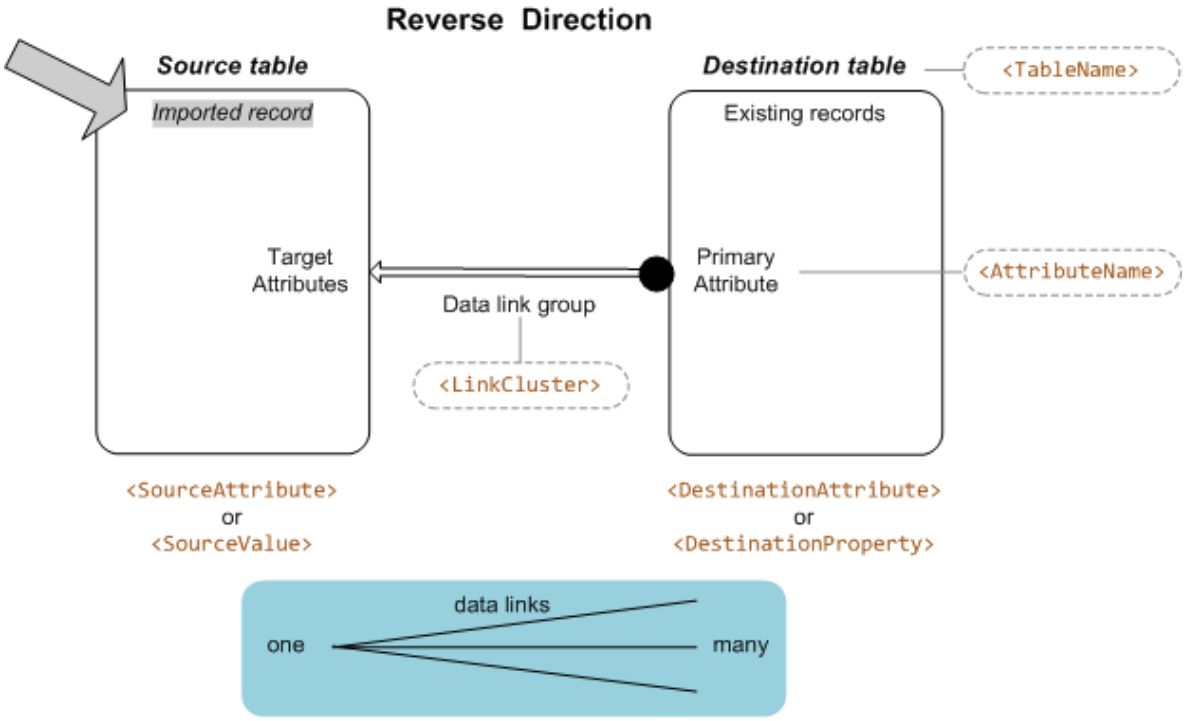


Figure 3 Data links in the reverse direction

All matching records will be linked to the newly-imported record. In the forward direction, the link is from the data in the named attribute in the imported record (source table) to records in the destination table. However, in the reverse direction, the link is from the data in the named attribute in the matched records (destination table) to the imported record in the source table.

A data link is not created if the attribute to which the data link group belongs (`AttributeName`) does not contain any data.

A data link is not created if any of the target attributes in the data link group (as defined in MI:Admin) do not contain data.

6.1.2 Examples

Add data links on the 'Comp. Modulus (L-dir) with Temp.' attribute in table MMPDS Data.

The link group name is 'Original Figure'.

The destination table, MMPDS Master Document, has an attribute called 'Figure name', which has already been populated. This attribute's value will be used to find records to make links to.

The target attribute of the data link group is the meta-attribute 'Source Figure'. The value of this attribute in the matched record(s) will be shown on the imported record's datasheet in MI:Viewer.

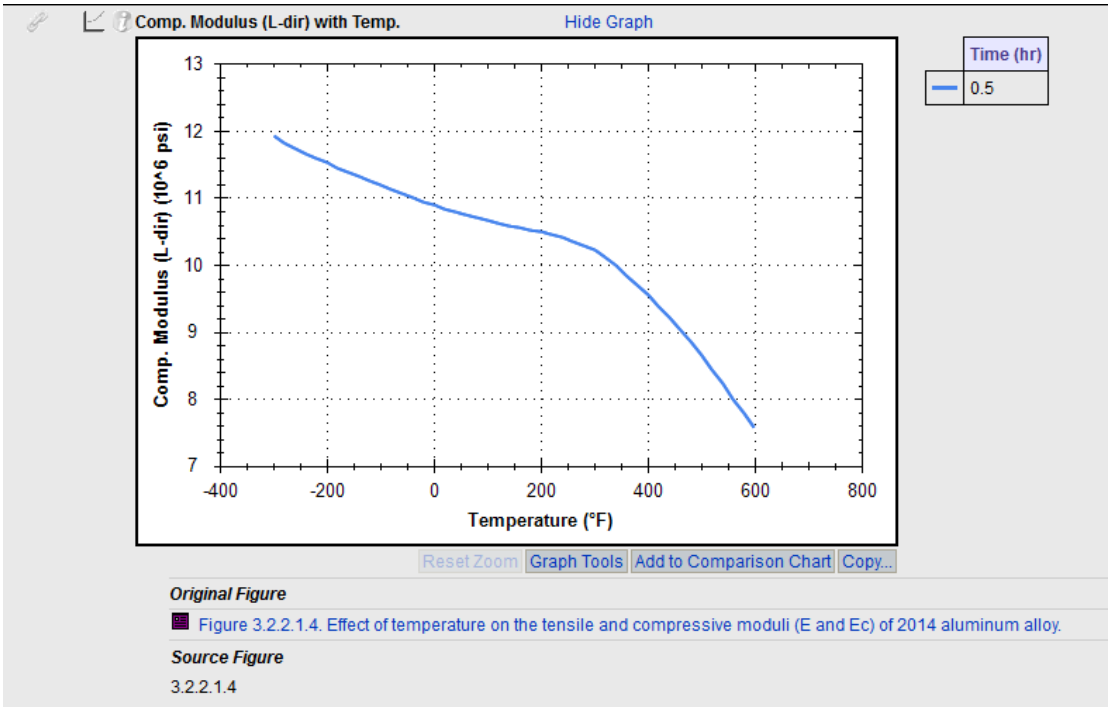


Figure 4 Example record showing a data link

Example 1

Link all records in the text file, based on the value of the regular expression named 'Comp_Modulus_LDir_wTemp_SourceFig'.

```
<AutoDataLink>
  <AutoDataLinkConfig>
    <AttributeName>Comp. Modulus (L-dir) with Temp.</AttributeName>
    <ClusterName>Original Figure</ClusterName>
    <SourceAttribute>Comp_Modulus_LDir_wTemp_SourceFig</SourceAttribute>
    <DestinationAttribute>Figure name</DestinationAttribute>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoDataLinkConfig>
</AutoDataLink>
…
<!-- in the textreaders section of the template, has a regular expression named
'Comp_Modulus_LDir_wTemp_SourceFig' -->
```

```
<RegularExpression name="Comp_Modulus_LDir_wTemp_SourceFig" attributename="Source
Figure">
  <FileCode>FileCode1</FileCode>
  <Expression>CompModulus_LDir_TempSourceFig: (?<ITEM1>.{1,})</Expression>
    <ParentAttribute> Comp. Modulus (L-dir) with Temp.</ParentAttribute>
    <Multiple>true</Multiple>
 </RegularExpression>
```

Example 2

The links will be based on the value '3.2.2.1.4', which must match in the destination table.

```
<AutoDataLink>
  <AutoDataLinkConfig>
    <AttributeName>Comp. Modulus (L-dir) with Temp.</AttributeName>
    <ClusterName>Original Figure</ClusterName>
    <SourceValue>3.2.2.1.4</SourceValue>
    <DestinationAttribute>Figure name</DestinationAttribute>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoDataLinkConfig>
</AutoDataLink>
```

6.2 Record links – <AutoLink>

Auto-linking is used to link imported records to existing records in the database. To link to another record, you need to identify something from the incoming record / text file which matches something in an existing record in the database.

The optional `<AutoLink>` element specifies how a record should be linked after it is created in the database. A single link configuration `<AutoLinkConfig>` may result in multiple links being created. Several link configurations can be used in a template. The order of the child elements of the `AutoLinkConfig` element is enforced.

```
<AutoLink>
  <AutoLinkConfig>
    <ClusterName>Record Link Group Name</ClusterName>
    <SourceAttribute>Source Attribute</SourceAttribute> or
      <SourceValue>Source Value</SourceValue>
    <DestinationAttribute>Destination Attribute</DestinationAttribute> or
      <DestinationProperty>Destination Property</DestinationProperty>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoLinkConfig>
</AutoLink>
```

6.2.1 Elements

Element	Required/ Optional	Description
`<ClusterName>`	Required	The name of an existing record link group in the database schema. The record link group specifies the source and destination table. Record link groups can be created in the MI:Admin application
`<SourceAttribute>` `<SourceValue>`	Required (one of)	Either a source attribute or a source value must be specified. If you specify a source attribute, then the value of the attribute in the record (which has been imported from the text file) becomes the search value. If no value is present for the attribute, no links are made. If you specify a source value, then the value in the template becomes the search value and is then used to search the destination table for a match.
`<DestinationAttribute>` `<DestinationProperty>`	Required (one of)	Specifies where to look for the search value. `<DestinationAttribute>` is one of the attributes in the destination table. `<DestinationProperty>` is an intrinsic property of the records in the destination table, one of TreeName, FullName, RecordID or GUID. If a match is found, a link is created between the newly-imported record and the record in the destination table that contains the matching value.
`<LinkCriterion>`	Required	Specifies the criterion for a match. The only option is 'Equals'.

If the match is made by attribute value, the Text Importer will search all the record data in the destination table and try to match the search value to the value in the destination attribute.

If the match is made by record property `<DestinationProperty>`, the Text Importer will search the specified property in all records in the destination table and try to match the search value to the value in the record property. The following record properties can be searched:

- TreeName
- FullName
- RecordID
- GUID

Using the source value from the template means that the same value will be used for every record created by the template and the same link will always be created. Using the source attribute allows the links to be created from information within the text files and these links can be unique for each record created.

6.2.2 Examples

Example 1: The link will be added to the 'Test Equipment' record link group. This config links the new record to every record in the 'Test Equipment' table named 'Instron'.

```
<AutoLink>
  <AutoLinkConfig>
    <ClusterName>Test Equipment</ClusterName>
    <SourceValue>Instron</SourceValue>
    <DestinationProperty>TreeName</DestinationProperty>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoLinkConfig>
</AutoLink>
```

Example 2: The link will be added to the 'Metals Pedigree' record link group. This config links the new record to every record in the 'Metals Pedigree' table where the Batch Number is the same as the Batch Number in the new record.

```
<AutoLink>
  <AutoLinkConfig>
    <ClusterName>Metals Pedigree</ClusterName>
    <SourceAttribute>Batch Number</SourceAttribute>
    <DestinationAttribute>Batch Number</DestinationAttribute>
    <LinkCriterion>Equals</LinkCriterion>
  </AutoLinkConfig>
</AutoLink>
```

6.3 Record and data placement options – <AutoPlacement>

Auto-placement can be used to allow new records and data to be imported into a specific database, table, folder, record, and/or subset.

Auto-placement settings must be defined in templates used by Remote Import and the command-line interface, where there is no option to select the import destination manually.

The importer will search the destination database for records that match the auto-placement information. It expects a single match from the database. An error will occur if there is more or less than one match.

- The data may be placed into a new record at the top of the tree, using `<ForceToRoot>`.
- Data may be placed into a specific location by searching and matching on existing records, which is done in a similar way to that of auto-linking (see Section *6.2*). The differences are that the search occurs in the same table in which the record will be created, and must only return a single match.

When matching occurs on an existing record or folder, the data may be placed:

- Into the specified record, using `<IsChild>false</IsChild>`.
- Into a record directly below the specified record/folder, using `<IsChild>true</IsChild>`.

If there is no existing record with a matching name, you can specify what should be done using the `<CreateIfNotFound>` option:

- To skip the import, use `<CreateIfNotFound>false</CreateIfNotFound>`.
- To create a new record, use `<CreateIfNotFound>true</CreateIfNotFound>`.

The order of the child elements of the `AutoPlacement` element is enforced.

```
<AutoPlacement>
  <DBKey>Database Key</DBKey>
  <TableName>Destination Table</TableName>
  <MatchAttribute>Match Attribute Name</MatchAttribute> or
    <MatchProperty>Match Record Property</MatchProperty>
  <SearchAttribute>Search Attribute Name</SearchAttribute> or
    <SearchValue>Search Value</SearchAttribute>
  <IsChild>true|false</IsChild>
  <CreateIfNotFound>true|false</CreateIfNotFound>
  <FilterName>Subset Name</FilterName>
  <ForceToRoot>true|false</ForceToRoot>
  <FunctionalMergeOption>Replace|TryMergeThenReplace|TryMergeThenDoNotImport
        </FunctionalMergeOption>
  <ConflictResolution>DoNotImport|Replace|ReplaceData|Merge|Append
        </ConflictResolution>
</AutoPlacement>
```

6.3.1 Elements

Element	Required/ Optional	Description
`<DBKey>`	Required	The key of the database you want to import to. This is used for validation, and you must select the correct database before importing the files. The database key is specified in the MI:Server Manager tool.
`<TableName>`	Required	The destination table where the new record will be created. If the optional `isaliased` property is set to *true*, then `<TableName isaliased="true">` becomes the name of an element in the template from which the Text Importer should read the table name. If omitted, the default value for `isaliased` is *false*.

Element	Required/ Optional	Description
`<MatchAttribute>` or `<MatchProperty>`	Optional	Use one of these elements to specify the match keyword in the destination table. See *6.3.2*. Required (along with `SearchAttribute` or `SearchValue`) unless `ForceToRoot` is *true*.
`<SearchAttribute>` or `<SearchValue>`	Optional	Use one of these elements to specify the match keyword in the import template. See *6.3.2*. Required (along with `MatchAttribute` or `MatchProperty`) unless `ForceToRoot` is *true*.
`<IsChild>`	Optional	Specifies whether the record will be imported as a child of the located record or into the located record itself. If *true*, then the data will be imported into a new record below the record identified. If *false*, then the data will be imported directly into the matched record.
`<CreateIfNotFound>`	Optional	If a record is not found by auto-placement, and `CreateIfNotFound` is *true*, the record is created under any folders defined by `TreeHierarchy` (or directly in the table root if no hierarchy levels are defined). If `CreateIfNotFound` is *false*, no records will be created in new locations – only found records can be imported to, even if `IsChild` or `ForceToRoot` options are true if `CreateIfNotFound` is not defined, a new record is created when `IsChild` or `ForceToRoot` options are true.
`<FilterName>`	Optional	Specifies a subset to which the new record will be added. The subset must already exist; subsets are created in the MI:Admin application.

Element	Required/ Optional	Description
`<ForceToRoot>`	Optional	Specifies whether to force the records being imported into the root of the table. This overrides the auto-placement search options. `TreeHierarchy` (see *6.4*) still applies. If *true*, then the data will be imported into a new record directly below the table level (at the top or root of the tree). If *false*, then the option is ignored.
`<FunctionalMergeOption>`	Optional	Specifies how functional data will be handled when the conflict resolution strategy is Merge or Append; see Section *6.3.3*.
`<ConflictResolution>`	Optional	Specifies how duplicate record name conflicts will be resolved; see Section *6.3.4*.

When importing via MI:Toolbox, values in the template take precedence over the location set by the user at the time of import. If the elements are not used in the template, auto-placement will not be activated during import and the record will be imported into the location specified by the user at the time of import, provided it is in a valid table (see Section *5.2*).

6.3.2 Identifying records – searching and matching

Data may be placed into a specific location by searching and matching on existing records, which is done in a similar way to that of auto-linking (see Section *6.2*). The differences are that the search occurs in the same table in which the record will be created, and must only return a single match.

Use either the `<SearchAttribute>` or the `<SearchValue>` element to specify the search keyword in the import template:

- `<SearchAttribute>` is an attribute whose value will be uniquely matched in one record in the table.
- `<SearchValue>` is a value in the template.

Use either the `<MatchAttribute>` or the `<MatchProperty>` element to specify the match keyword in the destination table:

- `<MatchAttribute>` is an attribute in the destination table.
- `<MatchProperty>` is an intrinsic property of the record in the destination table, one of TreeName, FullName, RecordID, or GUID.

6.3.3 Conflict resolution strategy – <ConflictResolution>

A GRANTA MI database does not allow multiple records to have the same short name within a folder. During the import, if the Text Importer attempts to create a new record which would violate this constraint, a record name conflict will occur. The `<ConflictResolution>` and `<FunctionalMergeOption>` (see 6.3.4) auto-placement elements define the strategy for dealing with the incoming data in this situation.

There are five valid values for the `<ConflictResolution>` element in a Text Importer template:

Value	Description
DoNotImport	The record is not imported and the Text Importer will move on to the next record in the list, or finish the import if this was the final or only record being imported.
Replace	The existing record and all the links associated with the record are deleted. The new record is then created. No information from the existing record is preserved. This option is not available for generic records.
ReplaceData	The existing record is kept and the incoming data is added to the record. Where there is existing data, this will be replaced by the incoming data. If there is no incoming data for an attribute, existing data remains unchanged. There is no control over which data is replaced. If incoming links are included, then they will be added to the record. The existing links remain unchanged and duplicate links will not be created.
Merge	(*Functional and multi-value data types only*) The existing record and data is kept and the incoming data is added to the record. Where there is existing multi-value data, the incoming data is merged. Where there is existing functional data, the **FunctionalMergeOption** setting instructs the Text Importer how to treat functional data. If the data type does not support merging, the incoming data will replace the existing data. If there is no incoming data for an attribute, existing data remains unchanged. If incoming links are included, then they will be added to the record. The existing links remain unchanged and duplicate links will not be created.
Append	(*Multi-value data types only*) The existing record and data is kept and the incoming data is added to the record. Where there is existing multi-value data, the incoming data is appended.

Value	Description
	Where there is existing functional data, the **FunctionalMergeOption** setting instructs the Text Importer how to treat functional data (note that there is no option to append functional data, only to merge it). If there is no incoming data for an attribute, existing data remains unchanged. If incoming links are included, then they will be added to the record. The existing links remain unchanged and duplicate links will not be created. If the data type does not support appending, the incoming data will replace the existing data (ReplaceData).

If `ConflictResolution` is not set in the template, duplicate record names are treated as follows:

- In MI:Toolbox, the user will be offered a choice in the Duplicate Record Name dialog.
- In the command-line interface, the setting from the command line is used. If there is no setting on the command line the default behavior is **Merge**.
- In Remote Import the default behavior is **Merge**.

6.3.4 Functional data merge strategy – <FunctionalMergeOption>

If the conflict resolution strategies of **Merge** or **Append** are chosen, the Text Importer will keep the existing record and attempt to merge any incoming functional data with existing functional data as follows:

Values	Description
Replace	The existing functional data is deleted and the incoming functional data is added. No attempt is made to merge or append the incoming functional data with the existing data, even if the data is compatible.
TryMergeThenReplace	If the incoming functional data is compatible with the existing functional data, it will be merged with the existing data. If the incoming functional data is not compatible with the existing functional data, the existing data is deleted and the incoming data is added.
TryMergeThenDoNotImport	If the incoming functional data is compatible with the existing functional data, it will be merged with the existing data. If the incoming functional data is not compatible with the existing functional data, the incoming data is not added and the existing data is unchanged.

If `<FunctionalMergeOption>` is not set in the template, incoming functional data is treated as follows:

- In MI:Toolbox
  - If the record name conflict resolution strategy is **Merge** and the incoming functional data is compatible with the existing functional data, the default behavior is **Merge**.
  - If the record name conflict resolution strategy is **Append** and the incoming functional data is compatible with the existing functional data, the default behavior is **Merge**.
  - If the incoming functional data is not compatible with the existing functional data, the user will be offered a choice in the **Cannot Merge Data** dialog.
- In the command-line interface
  - The setting from the command line is used.
  - If there was no setting on the command line, the default behavior is **TryMergeThenReplace**.
- In Remote Import the default behavior is **TryMergeThenReplace.**

6.3.5 Examples

Example 1: match based on attribute

The code for matching is in the text file, and can be found in the text section as a regular expression called *B_Num*. It matches an attribute in the database called *Batch Number*.

```
<AutoPlacement>
  <DBKey>CESLab14</DBKey>
  <TableName>Tensile test</TableName>
  <MatchAttribute>Batch Number</MatchAttribute>
  <SearchAttribute>B_Num</SearchAttribute>
  <FilterName>Tensile test data</FilterName>
  <IsChild>true</IsChild>
  <CreateIfNotFound>false</CreateIfNotFound>
  <FunctionalMergeOption>Replace</FunctionalMergeOption>
  <ConflictResolution>Append</ConflictResolution>
</AutoPlacement>
...
<!-- the textreaders section of the import template has a regular expression
   named 'B_Num' -->
      <RegularExpression name="B_num" import="false">
            <FileCode>File1</FileCode>
            <Expression>§BatchNumberCode§\s*(?<ITEM1>[^§¶]+)¶</Expression>
            <Multiple>true</Multiple>
      </RegularExpression>
```

Example 2: match based on GUID

The match is based on GUID, which can be found in the text section as a regular expression called 'GUID_Code'.

```
<AutoPlacement>
  <DBKey>CESLab14</DBKey>
  <TableName>Tensile test</TableName>
  <MatchProperty>GUID</MatchProperty>
  <SearchAttribute>GUID_Code</SearchAttribute>
  <FilterName>Tensile test data</FilterName>
  <IsChild>true</IsChild>
  <CreateIfNotFound>false</CreateIfNotFound>
  <FunctionalMergeOption>Replace</FunctionalMergeOption>
  <ConflictResolution>Append</ConflictResolution>
</AutoPlacement>

...
<!-- the textreaders section of the import template has a regular expression
   named 'GUID_Code' -->
      <RegularExpression name="GUID_Code" import="false">
            <FileCode>File1</FileCode>
            <Expression>§GUID§\s*(?<ITEM1>[^§¶]+)¶</Expression>
            <Multiple>true</Multiple>
      </RegularExpression>
```

Example 3: match based on record name

Add data to the record (uniquely) named 'AMS 6520, 250 Grade Maraging'

```
<AutoPlacement>
  <DBKey>CESLab14</DBKey>
  <TableName>Tensile test</TableName>
  <MatchProperty>FullName</MatchValue>
  <SearchValue>AMS 6520, 250 Grade Maraging</SearchValue>
  <FilterName>Tensile test data</FilterName>
  <IsChild>false</IsChild>
  <CreateIfNotFound>false</CreateIfNotFound>
  <FunctionalMergeOption>Replace</FunctionalMergeOption>
  <ConflictResolution>Append</ConflictResolution>
</AutoPlacement>
```

6.4 Record hierarchy – <TreeHierarchy>

The Text Importer can place the records being imported into a hierarchy within the MI:Viewer record tree. The optional `<TreeHierarchy>` element defines where in the tree hierarchy the record data should be placed when it is created in the database. Creating a record hierarchy can be combined with using auto-placement to locate the 'base' folder. In order to do this, the placement `<IsChild>` element must be set to *true* (see Section *6.3.1*).

```
<TreeHierarchy>
  <TreeLevel>attribute_or_constant</TreeLevel>
  <TreeLevel>attribute_or_constant</TreeLevel>
  <TreeLevel>attribute_or_constant</TreeLevel>
</TreeHierarchy>
```

6.4.1 Example

The imported data is put in: "TheNameOfMyFolder" -> Category1 -> Category2 -> Record Name

```
<Automation>
  <AutoPlacement>
    <DBKey>MYDB_3.0.Master</DBKey>
    <TableName>Raw data</TableName>
    <MatchProperty>FullName</MatchProperty>
    <SearchAttribute>treelevel1</SearchAttribute>
    <IsChild>true</IsChild>
  </AutoPlacement>

  <TreeHierarchy>
    <TreeLevel>Category1</TreeLevel>
    <TreeLevel>Category2</TreeLevel>
  </TreeHierarchy>
</Automation>

<TextReaders>
  <RegularExpression name="Category1" import="false">
    <FileCode>FileCode0</FileCode>
   <Expression>Eco01Categories</classificationSystem>[^>]<classificationVal
ue[^>]>(?<ITEM1>[^/]+)/[^>]*></Expression>
  </RegularExpression>
  <RegularExpression name="Category2" import="false">
    <FileCode>FileCode0</FileCode>
    <Expression>Eco01Categories</classificationSystem>[^>]<classificationVal
ue[^>]>[^//]/(?<ITEM1>[^/])<[^>]*></Expression>
  </RegularExpression>
</TextReaders>

<TemplateProperties>
  <TemplateProperty name="treelevel1">
    <Property>Text</Property>
    <Value>TheNameOfMyFolder</Value>
  </TemplateProperty>
</TemplateProperties>
```

7 Attribute data – <Data>

In this section, *Attribute* is used to describe an attribute in the database and *Data element* is used to describe an element in the template that creates a database attribute value.

The `<Data>` element appears under the root `<Template>` element and is required. It includes information on how to create the data for attributes within the database. The types of `Data` element fall into three main categories:

- **TextReaders.** Read data directly from the text file using a regular expression.
- **DataBuilders.** Generate data from other data elements.
- **TemplateProperties**. Specify additional information that is not present within the input text files, such as the series or specimen number, or the import file name.

An additional data type, **NullValues**, can be used to specify a list of global null values, that is, values that will be treated as being equivalent to a null or missing value. If a value being imported is equivalent to one of the specified null values, this value will be set to null and not imported, or, for grid functional data, the value will be imported as a 'not applicable' data point.

The type of the `Data` element determines how the value is created. For example, if you were reading Long Text data from one location in a text file, you might use a `TextReaders` element. However, if you were constructing Long Text data by combining several sentences from different locations you might use a `DataBuilders` element.

For data with complex formatting, such as functional data, specific `DataBuilders` elements have been created (see Table 7-2).

Table 7-1. Available elements for each Data element type

Data element type	Description	Required/ Optional	Available elements

`<TextReaders>`	Reads data directly from the text file	Required	`<RegularExpression>`
			`<GridReader>`
			`<Table>`

`<DataBuilders>`	Generates data from other data elements, that is, TextReaders, DataBuilders and TemplateProperties
	The <DataBuilders> element can only appear once in the template, and is included in the general template information and not in the Data list (see Section 3.5)
		Optional	`<RecordName>`
			`<TreeName>`
			`<Table>`
			`<Definition>`
			`<Text>`
			`<Functional>`
			`<TextConverter>`
			`<GridFunctional>`

Data element type	Description	Required/ Optional	Available elements
	If a piece of data that a DataBuilder relies on was not read or generated from the text files, then this DataBuilder will not create any values.		`<MathFunctional >` `<FileReader >` `<HyperlinkReader>` `<MultivaluedPoint>`
`<TemplateProperties>`	Provides additional information that is not present in the text files	Optional	`<TemplateProperties>`
`<NullValues>`	Specifies words or numbers that should not be imported	Optional	`<NullValues>`

`DataBuilders` can use other `DataBuilders` to create the value, but ultimately they must be derived from a `RegularExpression` or `Table` i.e. `TextReaders` data.

The `RecordName` and `TreeName` are different to all the other `DataBuilders`. They can both only appear once in the template, and so are included in the general template information and not in the `Data` list (see Sections *3.4* and *3.5*).

`Data` elements do not need to return any data and any derived `Data` elements using this value will also have no data.

*Table 7-2. Suggestions on which Data element to use for each GRANTA MI data type**

To import this type of data…	use this Data element*	in this element type
Date	<RegularExpression>	<TextReaders>
Discrete	<RegularExpression>	<TextReaders>
Discrete Functional	<Functional>	<DataBuilders>
Equations and logic	<MathFunctional >	<DataBuilders>
File (Embedded Media)	<FileReader >	<DataBuilders>
Float Functional (series data)	<Functional>	<DataBuilders>
Float Functional (grid data)	<GridFunctional>	<DataBuilders>
Hyperlink	<HyperlinkReader>	<DataBuilders>
Integer	<RegularExpression>	<TextReaders>

To import this type of data…	use this Data element*	in this element type
Logical	<RegularExpression>	<TextReaders>
Long Text	<RegularExpression> or <Text>	<TextReaders> or <DataBuilders>
Multi-value Discrete	<RegularExpression>	<TextReaders>
Multi-value Point	<MultivaluedPoint>	<DataBuilders>
Picture	<RegularExpression>	<TextReaders>
Point	<RegularExpression>	<TextReaders>
Range	<RegularExpression>	<TextReaders>
Short Text	<RegularExpression>	<TextReaders>
Tabular	The Text Importer cannot import tabular data.	

* Please note, this table is for guidance only. The most suitable Data element to use for your data will depend on the format of the data in the file, and whether it needs any further manipulation.

7.1 General notes on importing data

7.1.1 Dynamically-named attributes

The Text Importer can name attributes dynamically. This functionality allows the attribute populated by an element to change based on information in the text file. The Text Importer allows the attribute name to be dynamically set during the import by using one of the elements in the template to read the attribute name. An example where this might be used is a testing machine that produces a consistent data format for each test type, and in the database we want each of these test types to be added to a separate attribute. Rather than using a separate template for each attribute, type information within the file can be used to determine the attribute to be used.

Example

```
<Functional name="FunctionalData" attributename="AttributeName" isaliased="true">
  <FileCode>Code1</FileCode>
  <FunctionalType>Grid</FunctionalType>
  <YAxis>Row2</YAxis>
  <XAxis name="Temperature">Row1</XAxis>
  <Parameters>
    <Parameter>Loading</Parameter>
    <Parameter>Humidity</Parameter>
    <Parameter>Basis</Parameter>
  </Parameters>
  <Interpolate>true</Interpolate>
```

```
</Functional>
```

In the above example, the `isaliased` property is used. If this property is set to *true*, then the value in the `attributename` is no longer the name of an attribute in the database, but should be the name of a data element within the template.

```
<RegularExpression name="AttributeName">
  <FileCode>Code1</FileCode>
  <Expression>Attribute Name:(?<item>[_a-z]+)</Expression>
</RegularExpression>
```

7.1.2 Markdown

When importing Long Text data, if the data to be imported starts with the text **#markdown** followed by a newline, the data will be interpreted as Markdown-formatted, and displayed as such in MI:Viewer.

(Note that the newline after #markdown is required, otherwise you'll just see the text "#markdown" at the beginning of your attribute value).

7.1.3 Trailing zeroes

The Text Importer can import trailing zeroes for point data (including multi-value point) and range data.

The optional `storetrailingzeroes` attribute is associated with the `<MultivaluedPoint>` element (see Section *7.13.3*) and the `<RegularExpression>` element (see Section *7.1.3*). If you do not set it, it is assumed to be 'false'.

When creating the value to be imported with a Data element, you should ensure that the regular expression can recognize trailing zeroes. For example, when using a regular expression to read a value from a text file, you should ensure that the syntax includes a rule to match the decimal separator.

For import, the decimal separator is expected to be a period (.). However, if your data uses a different separator, you could construct your Data elements to read your separator and then convert it to the full stop. (A different decimal separator may be used as a result of setting the `<Locale>` element in the `<General>` section of the template; see *Locale – <Locale>*.)

7.2 RegularExpression (TextReaders)

`<RegularExpression>` extracts data from the text file using a regular expression. The regular expression must be specific enough to only return one match from a text file but generic enough to apply to each instance of the text files. The regular expression forms the basis of all the information read into the database, so it is necessary to have some knowledge of regular expressions before attempting to create a Text Importer template.

Please see *Appendix A* for more examples of the use of regular expressions in the Text Importer.

```
<RegularExpression name="name" attributename="attributename"
isaliased="true|false" import="true|false" storetrailingzeroes="true|false">
  <FileCode>filecode</FileCode>
  <Expression>expression1</Expression>
  <Expression>expression2</Expression>
                  ...
  <Multiple>true|false</Multiple>
  <Unit>unit</Unit>
  <Series>true|false</Series>
  <ParentAttribute>parentAttribute</ParentAttribute>
  <DataQuality>data quality element</DataQuality>
</RegularExpression>
```

7.2.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for name will be used.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is *false*.
`import`	Optional	Indicates whether or not the data should be imported.

Attribute	Required/ Optional	Description
		If *true*, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If *false*, this data will never be directly imported but is available for use by the `DataBuilders`. If omitted, then data will only be imported if there is an exact match with an attribute name in the database. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.
`storetrailingzeroes`	Optional	If *true*, the precision of the data will be stored by reading the number of digits. If omitted, the default value is *false*.

7.2.2 Elements

The order of the child elements is enforced.

Element	Required/ Optional	Description
`<FileCode>`	Required	The text file where the information is to be read from. It must be a `FileCode` from the `Files` list; see Section *4*.
`<Expression>`	Required (at least once)	Defines a regular expression. This can be included one or more times. See *Appendix A* for more information on regular expressions.
`<Multiple>`	Optional	Specifies whether or not a value for this attribute appears more than once in the file. If your text file has a single value for this attribute which applies to all records you are creating, then set `Multiple=false`. If your text file has a different value for this attribute for every record, then set `Multiple=true`. Typically, most multi-record imports would have `Multiple=true` on most elements.

Element	Required/ Optional	Description
`<Unit>`	Optional	Specifies the units for numeric data types. Note that the specified unit must already be defined in the database; you cannot create new units with the Text Importer.
`<Series>`	Optional	This should be set to *true* when multiple series will be created for the same functional data graph. When set to *true*, the data can only be imported as functional data.
`<ParentAttribute>`	Optional	(Meta-attributes only) The parent Attribute name (case sensitive). This should exactly match the Attribute name in the database.
`<DataQuality>`	Optional	The name of the Data element that will be used as the value for the quality rating. If the data with which the quality rating is associated is used by another element in the template e.g. in a calculation, the original quality rating will be ignored, but a new data quality element can be specified separately. If a quality ratings system has not been assigned to the table into which the data is being imported, the data is imported without the quality rating. If no quality rating is found, the data is imported without the quality rating. If a continuous quality ratings system has been assigned to the table, the quality rating must be numeric; otherwise an error will occur and no data will be imported for this attribute. If a discrete quality ratings system has been assigned to the table, the quality rating must be an existing, valid value; otherwise an error will occur and no data will be imported for this attribute.

7.2.3 Example

```
<RegularExpression name="Specimen ID">
  <FileCode>File1</FileCode>
  <Expression>SPECIMEN LABEL:\s*\[(?<ITEM>.+)\]</Expression>
  <Multiple>true</Multiple>
</RegularExpression>
```

Multiple `<Expression>` elements can be used in the `<RegularExpression>` element. This is useful when users want to use a single template to import files that differ slightly in format, for example, the naming convention may differ between files that are otherwise identical.

The first expression to return a value is used, in the order they appear in the template, and all subsequent expressions are ignored. If no expressions return a value, then no data is created. If multiple expressions are used to import multiple specimens from a single text file, then the Text Importer will look for the first successful expression in each specimen. However, it is recommended that the text files should be as consistent as possible.

In the following example, the term 'Ramp Rate' and 'Crosshead speed' are used for the same value:

```
<RegularExpression name="Crosshead Speed">
  <FileCode>File1</FileCode>
  <Expression>Crosshead speed:\s*\[(?<ITEM>.+)\]</Expression>
  <Expression>Ramp Rate:\s*\[(?<ITEM>.+)\]</Expression>
</RegularExpression>
```

Note that when using more than one `<Expression>` element inside a `RegularExpression`, alternative naming conventions can be matched using a single expression with the `|` character (logical OR). For example, instead of using two expressions to match 'Crosshead speed' and 'Ramp Rate', you can simply have `(Crosshead speed)|(Ramp Rate)`.

In the following example, two values for a range attribute named 'Load Range' are extracted from the text line:

Load Range: 30 - 50

```
<RegularExpression name="Load Range">
  <FileCode>File1</FileCode>
  <Expression>Load Range: (?<ITEM>[0-9]+) - (?<ITEM>[0-9]+)</Expression>
</RegularExpression>
```

In the following example, a file name for a picture attribute named 'Figure' is extracted from the text line:

Figure: C:\DemoFiles\Light.gif

```
<RegularExpression name="Figure">
  <FileCode>File1</FileCode>
  <Expression>Figure:\s*(?<ITEM>.+)\r</Expression>
</RegularExpression>
```

7.3 GridReader (TextReaders)

Reads row and column data from the text file. This element cannot be used to create data for an attribute directly and must be used by a `DataBuilder` element.

```
<GridReader name="name">
  <FileCode>filecode</FileCode>
  <GridExpression>regular expression</GridExpression>
  <CellExpression>regular expression</CellExpression>
  <IgnoreRows>integer</IgnoreRows>
  <IgnoreColumns>integer</IgnoreColumns>
  <Unit>unit</Unit>
  <Multiple>true/false</Multiple>
  <Series>true/false</Series>
</GridReader>
```

7.3.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.

7.3.2 Elements

Element	Required/ Optional	Description
`<FileCode>`	Required	The text file where the information is to be read from. It must be a `FileCode` from the `Files` list; see Section *4*.
`<GridExpression>`	Required	The regular expression that will be used to match the entire data grid.
`<CellExpression>`	Required	A regular expression that can be used to match a cell within the data grid.
`<IgnoreRows>`	Optional	If specified, the Text Importer will ignore the specified number of rows in the data grid, and only start processing data after this number of rows.
`<IgnoreColumns>`	Optional	If specified, the Text Importer will ignore the specified number of columns in the data grid, and only start processing data after this number of columns.
`<Unit>`	Optional	Specifies the units for numeric data types.

Element	Required/ Optional	Description
		Note that the specified unit must already be defined in the database; you cannot create new units with the Text Importer.
`<Multiple>`	Optional	Specifies whether or not a value for this attribute appears more than once in the file. If your text file has a single value for this attribute which applies to all records you are creating, then set `Multiple=false`. If your text file has a different value for this attribute for every record, then set `Multiple=true`. Typically, most multi-record imports would have `Multiple=true` on most elements.
`<Series>`	Optional	This should be set to *true* when multiple series will be created for the same functional data graph. When set to *true*, the data can only be imported as functional data.

7.3.3 Example

The following is an example of some data that can be imported using the `GridReader`:

```
start
                1       2       3       4       1       2       3       4
                A       A       A       A       B       B       B       B
10      |       1.0     1.5     1.8     2.0     1.2     1.7     2.2     3.0
20      |       1.1     1.6     1.9     2.1     1.3     1.8     2.3     3.1
30      |       1.3     1.6     1.8     2.2     1.4     1.9     2.4     3.2
40      |       2.5     3.0     3.5     4.0     3.0     3.5     4.0     4.5
end
```

Figure 5. Data to be imported using GridReader

The `<GridReader>` element to import the data above would look like this:

```
<GridReader name="MyGrid">
  <FileCode>filecode</FileCode>
  <GridExpression>(?=start\s*)(\s+.+)+(?=\s+end)</GridExpression>
  <CellExpression>(?<item>[0-9\.]{1,3})</CellExpression>
  <IgnoreRows>2</IgnoreRows>
  <IgnoreColumns>1</IgnoreColumns>
</GridReader>
```

In the example above, thirty-two values will be read by the Text Importer, as follows:

1. The `<GridReader>` will use the `<GridExpression>` to find the entire grid. In this case, it will match the six lines of text between ‘start’ and ‘end’.

2. The Text Importer then delimits the data using the line break character to create the six rows of data from this text.
3. The `<IgnoreRows>` element is set to 2, therefore the first two rows of data will be ignored and the Text Importer will start processing the third row.
4. Next, the Text Importer uses the `<CellExpression>` to find the individual cells of data in the row. In this case it matches any numeric values. The `<IgnoreColumns>` element is set to 2, so the first column of data will be ignored.
5. The first set of data generated by the Text Importer will be the eight values '1.0', '1.5', '1.8', '2.0', '1.2', '1.7', '2.2', and '3.0'.
6. The Text Importer will then continue to process the remaining rows of data until an array of data is created which reads left to right, top to bottom through the grid. Table

7.4 Table (TextReaders)

The `<Table>` type is used to read the information from a text file where the data is formatted as a table. It uses a regular expression to read in the table headers and uses this information to dynamically create a set of regular expressions to read the content of the table. This provides a tool that allows the Text Importer to uniquely identify any cell in the table by column and row, and use this value to create an attribute.

The basic syntax for this type is this:

```
<Table name="name">
    <FileCode>filecode</FileCode>
    <Header>
        <Expression>regex</Expression>
        <HeaderItem>header</HeaderItem>
            <Headername>name</Headername>
            <Value>regex</Value>
            <Unit>unit</Unit>
    </Header>
    <Row>
        <Expression>regex</Expression>
        <RowIdentity>identity</RowIdentity>
    </Row>
</Table>
```

The structure of the `<Table>` element is fairly complex. An example that illustrates how to use it is given below in Section *7.4.3*.

7.4.1 Attributes

Attribute	Required/ Optional	Description
name	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.

7.4.2 Elements

Element	Required/ Optional	Description
`<FileCode>`	Required	The text file where the data is to be read from. It must be a `FileCode` from the `Files` list; see Section *4*.
`<Header>`	Required	Defines the table header row.
`<Expression>`	Required	A regular expression that matches the entire header
`<HeaderItem>`	Required	Defines a column header; may consist of a value alone, or a value plus a unit.
`<Headername>`	Required	Identifies the column header substring
`<Value>`	Required	A regular expression that returns the column header substring from the header
`<Unit>`	Optional	For numeric data, specifies the unit for the column
`<Row>`	Required	Extracts data, and adds it under the column headers identified above
`<Expression>`	Required	A regular expression template that will match, in turn, each row of data.
`<RowIdentity>`	Required	A regular expression that returns a name for the row, used by the TableReader > RowIndex tag to identify which row to pull data from.

7.4.3 Example data and code

Consider a data file that contains this table-like data:

```
" ","Specimen label","Diameter (in)","Ext. gauge len (in)","Spec gauge len (in)"
"1","Ti003-01-S1",0.12450,0.50000,0.62500
"2","Ti003-01-S2",0.12400,0.50000,0.62500
"3","Ti003-01-S3",0.12450,0.50000,0.62500
"4","Ti003-01-S4",0.12400,0.50000,0.62500
```

The `<Table>` tag can require the Text Importer to read this into memory as this table:

(row index)	Specimen label	Diameter (unit: in)	Ext. gauge len (unit: in)	Spec gauge len (unit: in)
1	Ti003-01-S1	0.12450	0.50000	0.62500
2	Ti003-01-S2	0.12400	0.50000	0.62500
3	Ti003-01-S3	0.12450	0.50000	0.62500
4	Ti003-01-S4	0.12400	0.50000	0.62500

The first line of data has become the table's *header*. All the other lines have become the *content*.

The `<Table>` tag reads data into an internal table in memory. It does not place the data into MI attributes; that work is done later with a `<TableReader>` tag (see section *7.5*).

The XML code below will read this table, if it is placed inside the `Data > TextReader` element in the text importer template file.

```
<Table name="Summary One">
      <FileCode>FileCode1</FileCode>
      <Header>
            <Expression>\r\s*"(?&lt;ITEM1&gt;[#/@%0-9a-z\^ '\.\-
]{1,})","(?&lt;ITEM2&gt;[#/@%0-9a-z\^ '\.\-]{1,})"(,"(?&lt;ITEMX&gt;[#/@%0-9a-z\^
'\.\-]{1,}(\([#/@%0-9a-z\^ '\.\-]{1,}\))+)"){1,}\r</Expression>
            <HeaderItem>
                  <HeaderName>ITEM1</HeaderName>
                  <Value>(?&lt;ITEM1&gt;[#/@%0-9a-z\^ '\.\-]{1,})</Value>
            </HeaderItem>
            <HeaderItem>
                  <HeaderName>ITEM2</HeaderName>
                  <Value>(?&lt;ITEM2&gt;[#/@%0-9a-z\^ '\.\-]{1,})</Value>
            </HeaderItem>
            <HeaderItem>
                  <HeaderName>ITEMX</HeaderName>
                  <Value>(?&lt;ITEMX&gt;[#/@%0-9a-z\^ '\.\-]{1,})(\([#/@%0-9a-
z\^ '\.\-]{1,}\))?</Value>
                  <Unit>\((?&lt;ITEMX&gt;[#/@%0-9a-z\^ '\.\-]{1,})\)</Unit>
            </HeaderItem>
      </Header>
      <Row>
            <Expression>"(?&lt;ITEM1&gt;[#/@%0-9\^ '\.\-
]{1,})","(?&lt;ITEM2&gt;[\s#/@%0-9a-z\^ '\.\:_!\-]{1,})"!{,(?&lt;ITEMX&gt;[#/%e0-
9 \.\-]{1,})!}\r\s*</Expression>
            <RowIdentity>(?&lt;ITEM1&gt;[#/@%0-9\^ '\.\-]{1,})</RowIdentity>
      </Row>
</Table>
```

<FileCode> tag

The `FileCode` tag identifies which file to read from. The name (*FileCode1* in this case) should have been defined as a `TextFile` within the `Files` element of the text importer template file.

<Header> tag

<Expression> tag

The regular expression inside the `Expression` tag matches the entire header, i.e. the yellow-highlighted string below.

```
" ","Specimen label","Diameter (in)","Ext. gauge len (in)","Spec gauge len (in)"
"1","Ti003-01-S1",0.12450,0.50000,0.62500
"2","Ti003-01-S2",0.12400,0.50000,0.62500
"3","Ti003-01-S3",0.12450,0.50000,0.62500
"4","Ti003-01-S4",0.12400,0.50000,0.62500
```

Within the matched string, five named substrings are captured:

ITEM1, ITEM2, ITEMX(*1st instance*), ITEMX(*2nd instance*), ITEMX(*3rd instance*)

```
" ","Specimen label","Diameter (in)","Ext. gauge len (in)","Spec gauge len (in)"
"1","Ti003-01-S1",0.12450,0.50000,0.62500
"2","Ti003-01-S2",0.12400,0.50000,0.62500
"3","Ti003-01-S3",0.12450,0.50000,0.62500
"4","Ti003-01-S4",0.12400,0.50000,0.62500
```

These named substrings will be used in the next section to create column headers.

<HeaderItem> tag (simple example)

The elements within a `<HeaderItem>` tag declare a column header. A column header may consist of a value alone, or a value plus a unit. We will look at the second `<HeaderItem>`tag in the code, i.e. the tag that works with the ITEM2 substring to create the **Specimen label** header.

HeaderName tag

The `<HeaderName>` tag is the name of the substring identified above (in this case ITEM2).

Value tag

Inside the `<Value>` tag is a regular expression. It will search *within* the ITEM2 substring. Here is the full ITEM2 substring for clarity:

```
Specimen label
```

The regular expression matches the entire substring, and captures it, again using the name ITEM2:

ITEM2

```
Specimen label
```

<HeaderItem> tag (complex example)

Now we look at the third `<HeaderItem>` tag in the code, i.e. the tag that works with the ITEMX substring. In fact, there are three instances of ITEMX, and so this `<HeaderItem>`will be executed three times and create three headers. Below we consider only the first instance – the other two instances are very similar.

<HeaderName> tag

The `<HeaderName>` tag is the name of the substring identified above (in this case ITEMX(1st instance). Here is the full ITEM2 substring for clarity:

<Value> tag

Inside the `<value>` tag is a regular expression. It will search *within* the ITEMX(*1st instance*) substring. Here is the full ITEMX(*1st instance*) substring for clarity:

```
Diameter (in)
```

The regular expression matches the sub-substring below, and captures it, again using the name ITEMX:

ITEMX

```
Diameter (in)
```

<Unit> tag

Inside the `<Unit>` tag is another regular expression that will search *within* the ITEMX(*1st instance*) substring. The regular expression matches the sub-substring below, and captures it, again using the name ITEMX:

ITEMX

```
Diameter (in)
```

It is clear that the value of the column header will be **Diameter**, and the unit of the column header will be **in** (which should be a real unit in your MI database).

7.4.4 Row tag

The `<Row>` tag will extract data, and add it under the column headers we have identified above.

<Expression> tag

The `<Expression>` tag contains a *regular expression template* that will match, in turn, each row of data. Here is a copy of the *regular expression template*:

```
"(?&lt;ITEM1&gt;[#/@%0-9\^ '\.\-]{1,})","(?&lt;ITEM2&gt;[\s#/@%0-9a-z\^ '\.\:_!\-
]{1,})"!{,(?&lt;ITEMX&gt;[#/%e0-9 \.\-]{1,})!}\r\s*
```

The green-highlighted region is a special (MI-specific) notation, and is not treated like standard regular expression code. The Text Importer will turn the template into a normal regular expression by repeating the section bracketed by `!{` and `!}` an appropriate number of times depending on how

many column headers exist in total. (In this case, 3 times). The `!{` and `!}` characters themselves are otherwise ignored.

After processing, the expression matches, in turn, each row of data. The first match is highlighted in yellow:

```
" ","Specimen label","Diameter (in)","Ext. gauge len (in)","Spec gauge len (in)"
"1","Ti003-01-S1",0.12450,0.50000,0.62500
"2","Ti003-01-S2",0.12400,0.50000,0.62500
"3","Ti003-01-S3",0.12450,0.50000,0.62500
"4","Ti003-01-S4",0.12400,0.50000,0.62500
```

Within the matched string, five named substrings are captured:

ITEM1, ITEM2, ITEMX(*1st instance*), ITEMX(*2nd instance*), ITEMX(*3rd instance*)

```
" ","Specimen label","Diameter (in)","Ext. gauge len (in)","Spec gauge len (in)"
"1","Ti003-01-S1",0.12450,0.50000,0.62500
"2","Ti003-01-S2",0.12400,0.50000,0.62500
"3","Ti003-01-S3",0.12450,0.50000,0.62500
"4","Ti003-01-S4",0.12400,0.50000,0.62500
```

These substrings are placed under the columns, in order, as data.

<RowIdentity> tag

The `<RowIdentity>` tag declares a name for the row. The name is later used by the `TableReader` > `RowIndex` tag to identify which row to pull data from.

Within the `RowIdentity` tag is a regular expression, which will search *within* the ITEM1 substring. Here is the full ITEM1 substring:

```
1
```

The `RowIdentity` regular expression matches the entire substring, and captures it, using again the name ITEM1:

```
1
```

The row will be called **1**.

Note that, if in the *regular expression template* in the `Expression` tag, the ITEM1 capture group is deleted and replaced with the `RowIdentity` literal text, then the new expression will be:

```
"1","(?&lt;ITEM2&gt;[\s#/@%0-9a-z\^ '\.\:_!\-]{1,})"!{,(?&lt;ITEMX&gt;[#/%e0-9
\.\-]{1,})!}\r\s*
```

which gives an expression that matches only the appropriate row.

7.5 TableReader (DataBuilders)

Returns data (rows, columns, column headers, units) from a `table` in the text file (see *Table (TextReaders)*.

```
<TableReader name="name" attributename="attributename" isaliased="true|false"
import="true|false">
  <RowIndex>RowIndexName</RowIndex>
  <TableName>Table Name</TableName>
  <ParentAttribute>attribute name</ParentAttribute>
  <ColumnHeaders>
    <ColumnHeader>Header Name</ColumnHeader>
    <ColumnHeader>Header Name</ColumnHeader>
  </ColumnHeaders>
  <DefaultUnit>unit</DefaultUnit>
</TableReader>
```

7.5.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	A name that uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is *false*.
`import`	Optional	This indicates whether the data should be imported. If *true*, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If *false*, this data will never be directly imported but is available for use by the DataBuilders.

Attribute	Required/ Optional	Description
		If it is omitted, then data will only be imported if there is an exact match with an attribute name in the database. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.

7.5.2 Elements

Element	Required/ Optional	Description
`<RowIndex>`	Required	The value that will be used to identify the row in the table. This is the name of one of the Data elements in the template.
`<TableName>`	Required	The name of one of the `<Table>` elements in the `<TextReader>` that will be used to read the table.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.
`<ColumnHeaders>`	Required	A list of headers that will be looked for in the table. Multiple column headers can be specified to handle variations in nomenclature. The first header in the list that is found in the table will be used.
`<DefaultUnit>`	Optional	The unit information can be read from the tables in the text file. If the unit is missing, then you can specify a default unit, this will only be used if the unit cannot be read from the file.

7.6 Definition (DataBuilders)

Allows simple calculations to be carried out on data.

```
<Definition name="name" attributename="attributename" isaliased="true|false"
import="true|false">
  <DefinitionProperty>
    <Parameter>value|attribute</Parameter>
    <Parameter>value|attribute</Parameter>
    <Modifier>modifier</Modifier>
  </DefinitionProperty>
  <DefinitionProperty>
    <Parameter>value|attribute</Parameter>
    <Parameter>value|attribute</Parameter>
    <Modifier>modifier</Modifier>
  </DefinitionProperty>
  <Unit>unit</Unit>
  <ParentAttribute>parentAttribute</ParentAttribute>
  <DataQuality>data quality element</DataQuality>
</Definition>
```

7.6.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is *false*.

Attribute	Required/ Optional	Description
`import`	Optional	This indicates whether the data should be imported. If *true*, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If *false*, this data will never be directly imported but is available for use by the DataBuilders. If it is omitted, then data will only be imported if there is an exact match with an attribute name in the database. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.

7.6.2 Elements

Element	Required/ Optional	Description
`<DefinitionProperty>`	Required	The information to construct the calculation. See section 7.5.3 for more information.
`<Unit>`	Optional	Specifies the units for numeric attribute data types. Note that the specified unit must already be defined in the database; you cannot create new units with the Text Importer.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.
`<DataQuality>`	Optional	The name of one of the `<Data>` elements in the template that will be used as the value for the quality rating. If the data with which the quality rating is associated is used by another element in the template e.g. in a calculation, the original quality rating will be ignored, but a new data quality element can be specified separately.

Element	Required/ Optional	Description
		If a quality ratings system has not been assigned to the table into which the data is being imported, the data is imported without the quality rating. If no quality rating is found, the data is imported without the quality rating. If a continuous quality ratings system has been assigned to the table, the quality rating must be numeric; otherwise an error will occur and no data will be imported for this attribute. If a discrete quality ratings system has been assigned to the table, the quality rating must be an existing, valid value; otherwise an error will occur and no data will be imported for this attribute.

7.6.3 <DefinitionProperty>

This is an element within the `<Definition>`. It specifies the parameters for constructing the calculation. A calculation consists of a list of parameters and modifiers.

In the case of more than one, the first successful calculation will be used. A calculation is successful if all the attributes in the parameter list `<Parameter>` return a value from the text file. If any fail to return a value, then the calculation will fail and the next definition in the list will be attempted until one succeeds. If all calculations fail, then no value is created. This allows you to specify two methods to calculate the same value e.g. calculating area from (i) the diameter of a circular sample or (ii) the width and thickness of a rectangular sample.

```
<DefinitionProperty>
    <Parameter>value|attribute</Parameter>  <!-- first parameter -->
    <Parameter>value|attribute</Parameter>  <!-- second parameter ->
    <Parameter>value|attribute</Parameter>  <!-- third parameter -->
    <Modifier>modifier</Modifier>  <!-- first modifier -->
    <Modifier>modifier</Modifier>  <!-- second modifier -->
</DefinitionProperty>
```

There must be one more parameter than modifier. You may list all the parameters first, or all the modifiers, or intersperse the two. The order of calculation is the order in which they appear in the template, alternating between parameters and modifiers. So, in the calculation, the first parameter is followed by the first modifier, then the second parameter, second modifier and so on.

Parameters can either be one of the `Data` elements in the list or a numeric value. Modifiers can be one of: Multiply, divide, power, add, minus. For calculations on two or more arrays, these additional modifiers can be used: `multiplypairs`, `dividepairs`, `powerpairs`, `addpairs`, `minuspairs` (see p57).

Example

This example calculates the area of a sample by one of two methods:

- the diameter of a circular sample
- the width and thickness of a rectangular sample

```
<Definition name="Area">
  <DefinitionProperty>
      <Parameter>Gauge Diameter</Parameter>
      <Modifier>divide</Modifier>
      <Parameter>2</Parameter>
      <Modifier>power</Modifier>
      <Parameter>2</Parameter>
      <Modifier>multiply</Modifier>
      <Parameter>3.14159265358897932</Parameter>
  </DefinitionProperty>
  <DefinitionProperty>
      <Parameter>Gauge Thickness</Parameter>
      <Parameter>Gauge Width</Parameter>
      <Modifier>multiply</Modifier>
  </DefinitionProperty>
  <Unit>in^2</Unit>
 </Definition>
```

Working with arrays

When working with arrays, the resulting output will be a single array. If the arrays have different sizes of array, then the resulting array will be the size of the smallest array. If one of the arrays has no values, then the resulting array will have no values. Essentially the 'modifier' acts on the pair of numbers found at the corresponding offset of each array, incrementing the index from 0 until one runs out of data.

Example of the template XML:

```
<Definition name="Strain">
  <DefinitionProperty>
      <Parameter>Gage 1</Parameter>
      <Parameter>Gage 2</Parameter>
      <Modifier>addpairs</Modifier>
  </DefinitionProperty>
  <Unit>% strain</Unit>
</Definition>
```

Result: two arrays added together

Gage 1	Gage 2	Output
1	2	3
3	4	7
6	-	-

7.7 Text (DataBuilders)

Allows you to combine attributes to create a Long Text attribute. (Note that Short Text attributes are not created from this type, they can be created by a regular expression, see Section *7.1.3*.)

```
<Text name="name" attributename="attributename" import="true|false"
isaliased="true|false" items="all|any|first">
  <Item>attribute one</Item>
  <Item>attribute two</Item>
  <Item>attribute three</Item>
  <ParentAttribute>parentAttribute</ParentAttribute>
</Text>
```

7.7.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for 'name' will be used.
`import`	Optional	This indicates whether the data should be imported. If *true*, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If *false*, this data will never be directly imported but is available for use by the `DataBuilders`. If it is omitted, then data will only be imported if there is an exact match with an attribute name in the database. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name.

Attribute	Required/ Optional	Description
		If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is *false*.
`items`	Optional	Possible values are: `all` – the value for each `<Item>` is added to the `<Text>` element's return value, in the order specified in the template. If any `<Item>` is null, no output will be produced. `any` – any constituent `<Item>` elements yielding a value will be included in the output; null values will be ignored, and output will only be null if none of the `<Item>` elements return a value. `first` – the `<Text>` element's output will be that of the first `<Item>` in the template yielding a value. Subsequent `<Item>` elements will be ignored. Output will only be null if none of the `<Item>` elements produce a value. The default value is `all`.

7.7.2 Elements

The order of the child elements is enforced.

Element	Required/ Optional	Description
`<Item>`	Required	This is a list of all the template attributes that could be used by this attribute. The attribute name must match those used in the template. Each attribute will be added in order along the row. If the text file contains more than one row of data, then each row will form a new line in the database attribute value (see example below) and each attribute must have the same number of rows.

Element	Required/ Optional	Description
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.

7.7.3 Example

```
<Text name="Additional Test Notes">
  <Item>UDF Index</Item>
  <Item>Text: </Item>
  <Item>UDF Label</Item>
  <Item>Test : </Item>
  <Item>UDF Data</Item>
</Text>
```

When the above example XML is applied to the text file shown on page 60 in Figure 6, the text was read as three separate template attributes, one for each column. Each template attribute was itself a regular expression e.g. the text for UDF Label was read in as `[Test Cell:  ]` and converted to
"`Test Cell`".

The values for each template attribute were then combined to give a single value for the database attribute (Figure 7).

```
User Defined Fields:
    1:[Test Cell:          ][Instron 1              ]
    2:[Material:           ][Ti-6-4 Plate           ]
    3:[Extensometer:       ][LADT 01                ]
    4:[Work#               ][2005-03-01             ]
    5:[Batch:              ][456                    ]
```

Figure 6. Text in the file

Additional Information

Additional Test Notes

1: Test Cell : Instron 1
2: Material : Ti-6-4 Plate
3: Extensometer : LADT 01
4: Work# : 2005-03-01
5: Batch : 456

Figure 7. Example of imported text

7.8 Functional (DataBuilders)

Allows you to create a functional attribute corresponding to an existing attribute in the database.

```
<Functional name="name" attributename="attributename" isaliased="true|false">
  <FileCode>extension</FileCode>
  <FunctionalType>Grid|Series</FunctionalType>
  <YAxis>y-Axis data name</YAxis>
  <XAxis name="Name" defaultvalue="value|Name" isaliased="true|false"
parameteralias="Name">
    x-Axis data name
  </XAxis>
  <Log>true|false</Log>
  <Interpolate>true|false</Interpolate>
  <LineType>lines|markers|both</LineType>
  <Parameters>
    <Parameter name="Name" defaultvalue="value|Name" isaliased="true|false"
parameteralias="Name">
      parameter 1
    </Parameter>
    <Parameter name="Name" defaultvalue="value|Name"
      isaliased="true|false" parameteralias="Name">
      parameter 2
    </Parameter>
    <Parameter name="Name" defaultvalue="value|Name" isaliased="true|false"
parameteralias="Name">
      parameter 3
    </Parameter>
    <Parameter name="Name" defaultvalue="value|Name" isaliased="true|false"
parameteralias="Name">
      parameter 4
    </Parameter>
  </Parameters>
  <ParentAttribute>attribute name</ParentAttribute>
  <Series>true|false</Series>
  <ShowAsTable>true|false</ShowAsTable>
  <DataQuality>data quality element</DataQuality>
</Functional>
```

7.8.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.

Attribute	Required/ Optional	Description
`attributename`	Optional	The attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is *false*.

7.8.2 Elements

<Functional> child Element	Required/ Optional	Description
`<FileCode>`	Required	This is the FileCode where the functional data can be found. This is used to determine the number of series before the Text Importer attempts to read the data.
`<FunctionalType>`	Required	The type of functional data that will be created, `Grid` or `Series`.
`<YAxis>`	Required	Specifies the `<Data>` element that provides the data for the functional graph Y-axis. See page 64.
`<XAxis>`	Required	Specifies the `<Data>` element that provides the data for the functional graph X-axis. See page 64.
`<Log>`	Optional	If *true*, the functional graph will be displayed with a logarithmic scale. If *false*, the functional graph will be displayed with a linear scale. The default is *false*.
`<Interpolate>`	Optional	Determines if the functional graph is interpolated. The default is *false*, which means that no interpolation will take place.

<Functional> child Element	Required/ Optional	Description
		For Series data, if Interpolate=true, the interpolation method for the X-axis parameter is set to 'Automatic'. This means that the interpolation method will be inherited from the attribute. (Note that if the inherited interpolation method is `None`, then interpolation will not take place, despite the Interpolate value being set to *true*.) For parameters other than the x-axis parameter, interpolation will not occur. *For Grid data*, if `Interpolate=true`, the interpolation method for all parameters is set to 'Automatic'. This means that the interpolation method for each parameter will be inherited from the attribute. (Note that if the inherited interpolation method of a parameter is `None`, then interpolation will not take place for that parameter, despite the Interpolate value being *true*.)
`<LineType>`	Optional	Determines the line type for the functional graph, one of `lines`, `markers`, or `both`. The default setting is `lines`.
`<Parameters>`	Optional	A list of additional parameters that can be used for the functional data. Each `Parameter` should be the name of one of the `<Data>` elements in the template.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.
`<Series>`	Optional	If *true*, there are multiple series. If *false*, there is a single series. The default setting is *true*.
`<ShowAsTable>`	Optional	If *true*, the functional data will be shown as a table of data in the datasheets. If *false*, the functional data will be shown as a graph. The default setting is *false*.
`<DataQuality>`	Optional	The name of one of the `<Data>` elements in the template that will be used as the value for the quality rating on the functional data points. The number of quality ratings returned by the named element must be zero or exactly equal to the number of

<Functional> child Element	Required/ Optional	Description
		data points in the functional data; otherwise an error will occur and no data will be imported for this attribute. If a quality ratings system has not been assigned to the table into which the data is being imported, the data is imported without the quality rating. If no quality rating is found, the data is imported without the quality rating. If a continuous quality ratings system has been assigned to the table, the quality rating must be numeric; otherwise an error will occur and no data will be imported for this attribute. If a discrete quality ratings system has been assigned to the table, the quality rating must be an existing, valid value; otherwise an error will occur and no data will be imported for this attribute.

Attributes for <Xaxis> and< Parameter> child elements

Name	Required/ Optional	Description
`name`	Required	The name of the parameter in the database.
`defaultvalue`	Optional	The default parameter value or the name of a `<Data>` element in the template from which the Text Importer should read the default parameter value. For a numeric parameter, the value must be a number. Even if the parameter has named values, you must still use the numeric value. For a discrete parameter, the value must be a text string.
`isaliased`	Optional	Specifies whether or not to use an alias for the parameter name. If *true*, the name specified in `parameteralias` is used as the parameter name If *false*, the parameter name is specified by the `name` attribute. If omitted, the default value is *false*.

Name	Required/ Optional	Description
`parameteralias`	Optional	If `isaliased=true`, the `parameteralias` is the name of a `<Data>` element in the template from which the Text Importer should read the parameter name.

7.8.3 Grid and Series functional data

The behavior of the parameters used in the functional data changes depending on whether the functional type is set to **Grid** or **Series**.

- In the **Series** functional data type, each parameter should only contain a single value.
- In the **Grid** functional data type, parameters can contain a single value or multiple values.
- In the **Grid** data type, if multiple values are used this must be equal to the number of data points.

The functional data should be sorted so that the values for the x-axis parameter 1 are in ascending or descending order.

The following example of data can be imported as either series or grid functional data.

- If the **Series** data type is used, one series will be created.
- If the **Grid** data type is used, then each data point will be assigned the same value for 'Test type'.

```
Test type: Type A

Data    Temperature
10      0
20      50
30      100
40      150
50      200
```

Figure 8. Example of series functional data

The following example can only be imported correctly using the **Grid** functional data type.

If the **Series** data type is used, then a single series will be created. The **Series** data type can only use a single parameter value and it will use the first in the list if several are available. This is because the **Series** data type expects each set of data to correspond to an individual series. In this case, the first parameter value for the *Test type* parameter is 'Type A', so a single series of Test type A will be created. The series will only contain four values at the four unique temperatures and these will actually be the values for 'Type B' as the Text Importer processes the data from top to bottom and will overwrite the 'Type A' values. To import this data correctly using the **Series** data type, the text file needs to be altered and individual series split using a delimiter.

This data can easily be imported as **Grid** functional data. In the **Grid** data type, each data point will be assigned the correct parameter value.

Data	Temperature	Test type
10	0	Type A
20	50	Type A
30	100	Type A
40	150	Type A
15	0	Type B
25	50	Type B
35	100	Type B
45	150	Type B

Figure 9. Example of grid functional data

7.8.4 Discrete functional data

The data for discrete functional attributes is discrete text values rather than numeric values, and the data should be treated as series of point data. The `<Functional>` element for a discrete functional attribute should be formatted in the same way as series functional data, with the `FunctionalType` set to `Series`. The optional elements `<Interpolate>`, `<LineType>`, `<Log>`, and `<ShowAsTable>` do not apply.

7.8.5 Parameter ordering in functional data

The order of parameters associated with functional data is important in some cases. Specifically, if you are using grid functional data where one or more of the parameters use cubic spline interpolation, then parameter order may have an effect on the interpolation results. Note that parameter order has no effect if only linear interpolation is being used, or in series functional data.

What does parameter order mean? We interpolate gridded graphs one parameter at a time. So if we have a graph with 3 parameters (strain, cycles and time, say) to interpolate to a single point we first constrain a parameter (strain = 0.1% say) and create a new graph that only has the parameters cycles and time. Then we constrain another parameter (cycles = 1,000,000 say) to get a graph that only has one parameter. Finally, we constrain the last parameter (time) to get a point value. If we use a non-linear interpolation strategy for one of those steps, the value of the interpolated point can vary depending upon the order in which we interpolate the parameters.

Note that each piece of functional data can have the parameters stored in a different order. If version control is being used, then each version of the data can have a different order.

It is possible to check the order of parameters by using the **View All Data** option in Graph Tools in MI:Viewer. The order that the parameters appear in the columns is the order they are stored in the functional data.

The order of parameters on functional data is controlled when the data is loaded into MI, either via MI:Viewer, or by the Excel or Text importer. If the order of the parameters needs to be changed, then the data needs to be loaded again. If the data is version controlled, then a new version needs to be created.

When the data is loaded via the Text Importer using the `<Functional>` element in your template, then the order of the parameters will be the X-parameter followed by the parameters that appear in the `<Parameters>` element (in the same order that the parameters appear in the `<Parameters>` element)

If the data is imported using the `<GridFunctional>` template element, then the order of the parameters is determined by the order of the `<GridParameter>` elements.

Note: Merging new series into existing functional data will not change the parameter order of the data.

7.9 TextConverter (DataBuilders)

The `<TextConverter>` element allows you to change the value of some data before it is imported. This is used where you may not have a consistent naming convention in the data files but you want a consistent name in the database.

```
<TextConverter name="name" attributename="attributename" >
  <Entity>data name</Entity>
  <Strict>true|false</Strict>
  <Conversion>
    <NewValue>new 1</NewValue>
    <ExistingValue>existing value 1</ExistingValue>
    <ExistingValue>existing value 2</ExistingValue>
    <ExistingValue>existing value 3</ExistingValue>
  </Conversion>
</ TextConverter >
```

7.9.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.

7.9.2 Elements

The order of the child elements of the `<TextConverter>` element is enforced.

Element	Required/ Optional	Description
`<Entity>`	Required	Specifies the `<Data>` element used to find the existing value.
`<Strict>`	Optional	Determines whether or not values that do not match one of the allowed values are created.

Element	Required/ Optional	Description
		If *true*, then, if the file contains one of the allowed values, the data is created; if the value is not one of the listed values, it is not created. If *false* and the file does not contain one of the allowed values, the value is still created as the original value. The default value is *true*.
`<Conversion>`	Required (at least once)	This is a conversion associated with this element. There can be one or more conversions specified in the `<TextConverter>`.
`<NewValue>`	Required	The new value that will be created for the named database attribute.
`<ExistingValue>`	Required	The list of values that if found will cause the creation of the new data with the new value.

7.9.3 Example

```
<TextConverter name="Specimen Geometry" attributename="Specimen Geometry">
  <Entity>SpecimenGeometry-File</Entity>
  <Strict>true</Strict>
  <Conversion>
    <NewValue>Cylindrical</NewValue>
    <ExistingValue>Circular</ExistingValue>
    <ExistingValue>Round</ExistingValue>
    <ExistingValue>Bar</ExistingValue>
  </Conversion>
  <Conversion>
    <NewValue>square</NewValue>
    <ExistingValue>beam</ExistingValue>
    <ExistingValue>quadrilateral</ExistingValue>
    <ExistingValue>cube</ExistingValue>
  </Conversion>
</TextConverter>
```

In this example, if the *SpecimenGeometry-File* Data element has the value *Circular*, *Round*, or *Bar*, then the **Specimen Geometry** data will be created for the new value with value *Cylindrical*. The **Specimen Geometry** will always contain the same value.

7.10 GridFunctional (DataBuilders)

The `<GridFunctional>` element allows you to have more control over how grid functional data is imported, and allows the importing of more complicated grid functional data than is possible using only the `<Functional>` type. This element is optional and appears under the `<DataBuilders>` element in the template. The order of the child elements of the `<GridFunctional>` element is enforced.

```
<GridFunctional name="name" attributename="attributename" isaliased="true|false"
>
  <DefaultXParameter>parameter</DefaultXParameter>
  <DataGrid>data</DataGrid>
  <ParentAttribute>attribute name</ParentAttribute>
  <GridParameter name="parametername" defaultvalue="value|Name" >
    <Parameter>parameterdata</Parameter>
    <ParameterType>Single|Row|Column|Grid</ParameterType>
  </GridParameter>
  <GridParameter>
    <Parameter>parameterdata</Parameter>
    <ParameterType>Single|Row|Column|Grid</ParameterType>
  </GridParameter>
  <DataQuality>data quality element</DataQuality>
</GridFunctional>
```

7.10.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`isaliased`	Optional	Specifies whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. The default value is *false*.

7.10.2 Elements

Element	Required/ Optional	Description
`<DefaultXParameter>`	Required	The parameter that should be used for the X-axis by default when viewed in MI:Viewer.
`<DataGrid>`	Required	The name of the `<Data>` element in the template that will provide the data points used to create the grid functional data.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent Attribute name. This should exactly match the Attribute name in the database. Note it is case sensitive.
`<GridParameter>`	Required (at least once)	The parameters that will be used to create the grid functional data. There must be at least one parameter. Attributes: `name` (optional) The name of the parameter as it appears in the database. If omitted, the name of the element is used as the parameter name. `defaultvalue` (optional) The default parameter value or the name of a `<Data>` element in the template from which the Text Importer should read the default parameter value. For a numeric parameter, the value must be a number. Even if the parameter has named values, you must still use the numeric value. For a discrete parameter, the value must be a text string.
<DataQuality>	Optional	The name of one of the `<Data>` elements in the template that will provide the quality rating value for the functional data points. The number of quality ratings returned by the named element must be zero or exactly equal to the number of data points in the functional data; otherwise an error will occur and no data will be imported for this attribute. If a quality ratings system has not been assigned to the table into which the data is being imported, the data is imported without the quality rating.

Element	Required/ Optional	Description
		If no quality rating is found, the data is imported without the quality rating. If a continuous quality ratings system has been assigned to the table, the quality rating must be numeric; otherwise an error will occur and no data will be imported for this attribute. If a discrete quality ratings system has been assigned to the table, the quality rating must be an existing, valid value; otherwise an error will occur and no data will be imported for this attribute.

<GridParameter> elements

Element	Required/ Optional	Description
`<Parameter>`	Required	The name of the `<Data>` element in the template that will be used to create the grid functional data.
`<ParameterType>`	Required	The parameter can be one of four parameter types: `Single` The parameter data will only contain a single value and this should be applied to every point in the grid. `Row` The grid of data will have a unique parameter value for every row in the grid. Every data point in the row will be assigned the same parameter value. `Column` The grid of data will have a unique parameter value for every column in the grid. Every data point in the column will be assigned the same parameter value. `Grid` Every point in the grid will be assigned a unique parameter value for this parameter. See *Appendix B* for examples of the four data types.

7.11 MathFunctional (DataBuilders)

Allows you to import equation-based (*Equations and Logic)* data. This element is optional and appears under the `<DataBuilders>` element in the template. The order of the child elements of the `<MathFunctional>` element is not enforced.

```
<MathFunctional name="name" attributename="attributename" isaliased="true|false">
  <FileCode>Code1</FileCode>
  <Log>false</Log>
  <AxesAreInverted>false</AxesAreInverted>
  <Expression>expression</Expression>
  <ParentAttribute isaliased="true|false">attribute name</ParentAttribute>
  <DefaultFreeParameter>
    <Parameter name="name" isaliased="true|false" parameteralias="parametername"
/>
  </DefaultFreeParameter>
  <Extents>
    <Extent>
      <Parameter name="name" isaliased="true|false"
parameteralias="parametername" />
      <Value isaliased="true|false" valuealias="valuename" value="value"
type="min|max" />
    </Extent>
  </Extents>
  <Curves>
    <Curve>
      <Constraints>
        <Constraint>
          <Parameter name="name" isaliased="true|false"
parameteralias="parametername" />
          <Value isaliased="true|false" valuealias="valuename" value="value"
type="min/max" />
        </Constraint>
      </Constraints>
      <Label isaliased="true|false" labelaliased="labelname" value="value" />
    </Curve>
  </Curves>
</MathFunctional>
```

7.11.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute

Attribute	Required/ Optional	Description
		name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`isaliased`	Optional	Whether or not to use an alias for the attribute name. If *false*, the `attributename` is the name of the attribute in the database. If *true*, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. The default value is *false*.

7.11.2 Elements

Element	Required/ Optional	Description
`<FileCode>`	Required	The text file where the information is to be read from. It must be a `FileCode` from the `Files` list. See Section *4.1*.
`<Log>`	Optional	Determines if the curve data graph axis for the attribute value is displayed with a logarithmic scale. Set to *true* for logarithmic. The default is *false*.
`<AxesAreInverted>`	Optional	Determines if the curve data graph axes are transposed. Set to *true* for the parameter to be displayed on the Y-axis and the attribute value on the X-axis. The default is *false*.
`<Expression>`	Required	This should be the name of another `<Data>` element in the template, containing the name of the expression.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive. Attributes: `<isaliased>` (optional). If *false,* the name of the attribute in the database is used. If

Element	Required/ Optional	Description
		true, the Text Importer reads the attribute name from an element in the template. The default is *false*.
`<DefaultFreeParameter>`	Required	The parameter that should be used for the curve X-axis by default when viewed in MI:Viewer. Child elements: `<Parameter>`, see below.
`<Extents>`	Optional	The extents of each of the parameters. Child elements: `<Extent>`, see below.
`<Curves>`	Optional	The curve parameter values, labels and which curve is the default. There can be several curves. Child elements: `<Curve>`, see below.

<DefaultFreeParameter> child elements

Element	Required/ Optional	Description
`<Parameter>`	Required	The name of the parameter. Attributes are: `name` (optional) The name of the parameter in the database. `isaliased` (optional) If *false*, the name is the name of the parameter in the database. If *true*, the `parameteralias` is used. If omitted, the default is *false*. `parameteralias` (optional) If `isaliased=true`, this is the name of a `<Data>` element in the template from which the Text Importer should read the parameter name.

<Extents> child elements

Element	Required/ Optional	Description
`<Extent>`	Required	The extents of a single parameter.

Element	Required/ Optional	Description
`<Parameter>`	Optional	The name of the parameter. Attributes are the same as for the `<Parameter>` child element of `<DefaultFreeParameter>` (see above).
`<Value>`	Optional	The maximum and minimum value of the parameter. Attributes are: `isaliased` (optional) If *false*, the value is the value of the parameter. If *true*, the `valuealias` is used. If omitted, the default value is *false*. `valuealias` (optional) If `isaliased=true`, this is the name of a `<Data>` element in the template from which the Text Importer should read the value. `value` (Optional) The value of the parameter. The units for the value are the database units of the parameter `type` (Optional) Determines if the value is the minimum extent *min* or the maximum extent *max*. This is only valid when used for the Extent/Value.

<Curves> child elements

Element	Required/ Optional	Description
`<Curve>`	Optional	The information for a single curve.
`<Constraints>`	Required	The values of any constrained parameters.
`<Constraint>`	Optional	The value of a constrained parameter.
`<Parameter>`	Required	The name of the parameter. Attributes are the same as for the `<Parameter>` child element of `<DefaultFreeParameter>` (see above).
`<Value>`	Required	The value of the parameter. Attributes are the same as for the `<Value>` child element of `<Extent>` (see above).

Element	Required/ Optional	Description
`<Label>`	Optional	The label for the curve. Attributes are: `isaliased` (optional) If false, the value is the text for the label. If true, the `labelaliased` is used. If omitted, the default value is false. `labelaliased` (optional) If `isaliased=true`, this the name of a `<Data>` element in the template from which the Text Importer should read the text label. `value` (optional) The value for text label.

7.11.3 Example

The following is an example of the `<MathFunctional>` element for an equations and logic attribute named 'Fatigue model' with three parameters; Number of Cycles, Stress Ratio, and Temperature.

```
<MathFunctional name="mathFunc" attributename="Fatigue Model" isaliased="false">
    <FileCode>File1</FileCode>
    <Log>false</Log>
    <AxesAreInverted>false</AxesAreInverted>
    <Expression>ExpressionRegEx</Expression>
    <DefaultFreeParameter>
      <Parameter name="Number of Cycles" isaliased="false" />
    </DefaultFreeParameter>
    <Extents>
      <Extent>
        <Parameter name="Number of Cycles" isaliased="false" />
        <Value isaliased="true" valuealias="Parameter1ExtentLowRegEx" type="min" />
        <Value isaliased="true" valuealias="Parameter1ExtentHighRegEx" type="max" />
      </Extent>
      <Extent>
        <Parameter name="Stress Ratio" isaliased="false" />
        <Value isaliased="true" valuealias="Parameter2ExtentLowRegEx" type="min" />
        <Value isaliased="true" valuealias="Parameter2ExtentHighRegEx" type="max" />
      </Extent>
      <Extent>
        <Parameter name="Temperature" isaliased="false" />
        <Value isaliased="false" value="273.15" type="min" />
        <Value isaliased="false" value="1033.15" type="max" />
      </Extent>
    </Extents>
    <Curves>
      <Curve>
        <Constraints>
          <Constraint>
            <Parameter name="Number of Cycles" isaliased="false" />
            <Value isaliased="true" valuealias="Curve1Parameter1Value" />
```

```
          </Constraint>
          <Constraint>
            <Parameter name="Stress Ratio" isaliased="false" />
            <Value isaliased="true" valuealias="Curve1Parameter2Value" />
          </Constraint>
          <Constraint>
            <Parameter name="Temperature" isaliased="false" />
            <Value isaliased="false" value="317.15" />
          </Constraint>
        </Constraints>
        <Label isaliased="true" labelaliased="Curve1LabelRegEx" />
      </Curve>
      <Curve>
        <Constraints>
          <Constraint>
            <Parameter name="Number of Cycles" isaliased="false" />
            <Value isaliased="true" valuealias="Curve2Parameter1Value" />
          </Constraint>
          <Constraint>
            <Parameter name="Stress Ratio" isaliased="false" />
            <Value isaliased="true" valuealias="Curve2Parameter2Value" />
          </Constraint>
          <Constraint>
            <Parameter name="Temperature" isaliased="false" />
            <Value isaliased="false" value="373.15" />
          </Constraint>
        </Constraints>
        <Label isaliased="false" value="Curve 2 at 100 C" />
      </Curve>
    </Curves>
  </MathFunctional>
```

7.12 FileReader (DataBuilders)

The `<FileReader>` element appears under the `<DataBuilders>` element in the template and allows the Text Importer to import additional files into the new record.

It reads the file path from the text file and then imports this file into the new record. The file path of the data files used in the import can be found using the `<TemplateProperties>` element; see Section 7.15.

```
<FileReader name="file attribute" attributename="attributename"
import="true|false">
  <FileName>filename</FileName>
  <Description>description</Description>
  <AllowIndex>true|false</AllowIndex>
  <DisplayAsRecord>true|false</DisplayAsRecord>
  <ParentAttribute>attributename</ParentAttribute>
  <Target>targetpane</Target>
</FileReader>
```

7.12.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for 'name' will be used.
`import`	Optional	Indicates whether or not the data should be imported. If `import=true`, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If `import=false`, this data will never be directly imported but is available for use by the `<DataBuilders>`. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.

7.12.2 Elements

Name	Required/ Optional	Description
`<FileName>`	Required	The name of one of the `Data` elements in the template. This should return the full path to the file that you want to import.
`<Description>`	Required	The name of one of the `Data` elements in the template. This should contain a description of the file that is being imported.
`<AllowIndex>`	Optional	If *true*, the file will be indexed on import, allowing text within the file to be included in searches in MI:Viewer.

Name	Required/ Optional	Description
		If *false*, the file will not be indexed. The default value is *false*.
`<DisplayAsRecord>`	Optional	If *true*, when the record is opened in MI:Viewer, this file will be displayed to the user instead of the datasheet. The default value is *false*.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.
`<Target>`	Optional	The target pane in MI:Viewer where the imported file will be displayed. The target can be one of the following values: `CURRENT_PANE` – Opens the file in the current pane. `NEW_WINDOW` – Opens the file in a new window. `RIGHT_PANE` – Opens the file in the right (main content) pane of MI:Viewer. `LEFT_PANE` – Opens the file in the left (browse) pane of MI:Viewer. `BOTH_PANES_BELOW_TOOLBAR` – Replaces the MI:Viewer left and right pane with a single pane and loads the file into this pane. The MI:Viewer toolbar will remain. `CURRENT_WINDOW` – Opens the file in the current window. If no value is supplied, the default is `CURRENT_PANE`.

Contact the administrator of your GRANTA MI system for information on which files types for embedded media can be searched and displayed in MI:Viewer. Note that a file can still be stored, even if it cannot be searched or displayed by default.

7.13 HyperlinkReader (DataBuilders)

The `<HyperlinkReader>` element allows the Text Importer to import a URL, description, and target for a hyperlink. This element is optional and appears under the `<DataBuilders>` element in the template.

```
<HyperlinkReader name="hyperlink attribute" attributename="attributename"
import="true|false">
  <URL>element name</URL>
  <Description>element name</Description>
  <Target>target</Target>
</HyperlinkReader>
```

7.13.1 Attributes

Attribute	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for `name` will be used.
`import`	Optional	This indicates whether the data should be imported. If *true*, the data will be imported directly as a simple data type, for example a single point value, a range, or a short text. If *false*, this data will never be directly imported but is available for use by the DataBuilders. If it is omitted, then data will only be imported if there is an exact match with an attribute name in the database. We recommend setting `import=true` as, without it, data will only be imported if the attribute name exactly matches an attribute in the database. If you make a mistake in the attribute name, you will not get any errors unless you explicitly set `import=true`; you will simply get no data imported.

7.13.2 Elements

Name	Required/ Optional	Description
`<URL>`	Required	The name of one of the `Data` elements in the template. This element should return the URL of the hyperlink. The URL may include tokens that are replaced with strings when the URL is resolved; see *7.13.3*.
`<Description>`	Optional	The name of one of the `Data` elements in the template. This element should contain a description of the hyperlink. If no value is supplied, the default value is that of the URL.
`<Target>`	Optional	The target pane in MI:Viewer for the link : `CURRENT_PANE` – Opens the hyperlink in the current pane. `NEW_WINDOW` – Opens the hyperlink in a new window. `RIGHT_PANE` – Opens the hyperlink in the right (main content) pane. `LEFT_PANE` – Opens the hyperlink in the Contents pane. `BOTH_PANES_BELOW_TOOLBAR` – Replaces the MI:Viewer left and right pane with a single pane and loads the hyperlink into this pane. The MI:Viewer toolbar will remain. `CURRENT_WINDOW` – Opens the hyperlink in the current window. If no value is supplied, the default is `CURRENT_PANE`.

7.13.3 Tokens in URLs

A URL may include tokens that are replaced with values defined in the database when the URL is resolved. Two different types of token can be included:

- **Replacement string**. This allows a replacement string defined in the database schema to be inserted into the URL.
- **Short text attribute**. This allows the value of a short text attribute in the record to be inserted into a URL within the same record. For example, a project code, a testing series ID, or a batch number.

Tokens have the following syntax:

{s:*name-of-replacement-string*}

{a:*name-of-attribute*}

For example:

{s:webURL}/resources/note1.html

https://acme.com/**{a:Project Code}**/notes.htm

{s:webURL}/{a:Material Type}/{a:Batch Number}.html

See the MI:Admin help for more information about defining Replacement Strings in GRANTA MI.

7.14 MultivaluedPoint (DataBuilders)

The `<MultivaluedPoint>` element allows the Text Importer to import multi-value point data. This element is optional and appears under the `<DataBuilders>` element in the template.

```
<MultivaluedPoint name="point attribute" attributename="attributename"
isaliased="true|false" storetrailingzeroes="true|false">
  <FileCode>filecode</FileCode>
  <PointValuesElement>element name</PointValuesElement>
  <Parameters>
    <Parameter name="Name" isaliased="true|false" parameteralias="Name">
      parameter 1
    </Parameter>
    <Parameter name="Name" isaliased="true|false" parameteralias="Name">
      parameter 2
    </Parameter>
  </Parameters>
  <ParentAttribute isaliased="true|false">attribute name</ParentAttribute>
  <DataQuality>data quality element</DataQuality>
</MultivaluedPoint>
```

7.14.1 Attributes

Name	Required/ Optional	Description
`name`	Required	This name uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.
`attributename`	Optional	The name of the attribute where this data will be created. This must exactly match the attribute name in the database. Note the name is case sensitive. If omitted, the value for **name** will be used.
`isaliased`	Optional	If **false**, the `attributename` is the name of the attribute in the database.

Name	Required/ Optional	Description
		If **true**, the `attributename` is the name of an element in the template from which the Text Importer should read the attribute name. If omitted, the default value is **false**.
`storetrailingzeroes`	Optional	If **true**, the precision of the data will be stored by reading the number of digits. If omitted, the default value is **false**.

<MultivaluedPoint> child elements

Name	Required/ Optional	Description
`<FileCode>`	Required	This is the 'FileCode' where the multi-value data can be found.
`<PointValuesElement>`	Required	The `Data` element name which will match the point attribute values.
`<Parameters>`	Optional	A list of additional parameters that can be used for the multi-value data. Each Parameter should be the name of one of the Data elements in the template and may take the following attributes: `name` (optional) The name of the parameter in the database, if different from the Data element name `isaliased` (optional) If *false*, the `name` is the name of the parameter in the database. If *true*, the `parameteralias` is used. If omitted, the default value is *false*. `parameteralias` (optional) If `isaliased=true`, `parameteralias` is the name of the Data element in the template from which the Text Importer should read the parameter name.
`<ParentAttribute>`	Optional	If this data is to be imported as metadata, this is the parent Attribute name. This should exactly match the attribute name in the database. Note it is case sensitive.

Name	Required/ Optional	Description
`<DataQuality>`	Optional	The Data element in the template that will be used as the value for the quality rating on the multi-value data. The number of quality ratings returned by the named element must be zero or one; otherwise an error will occur and no data will be imported for this attribute. If a quality ratings system has not been assigned to the table into which the data is being imported, the data is imported without the quality rating. If no quality rating is found, the data is imported without the quality rating. If a continuous quality ratings system has been assigned to the table, the quality rating must be numeric; otherwise an error will occur and no data will be imported for this attribute. If a discrete quality ratings system has been assigned to the table, the quality rating must be an existing, valid value; otherwise an error will occur and no data will be imported for this attribute.

7.15 TemplateProperties

This section exists under the `<Data>` element.

It formally describes some of the additional information that can be imported. The information cannot be imported directly into the database, but can be used by other elements in the template. The order of the child elements is enforced.

```
<TemplateProperties>
  <TemplateProperty name="PropertyName">
    <Property> Series|Specimen|FileName|FilePath|NewLine|Text </Property>
    <Value>StringValue</Value>
    <FileCode>FileCode</FileCode>
  </TemplateProperty>
</TemplateProperties>
```

7.15.1 Attributes

Name	Required/ Optional	Description
`name`	Required	Uniquely identifies this element of the template. If other elements use the data returned by this element, they should refer to this name.

7.15.2 Elements

Child element	Required/ Optional	Description
`<Property>`	Required	One of these values: `Series` The number of the current series being read. For example, if the text file contains data on three series, this value will be 1, 2, or 3 `Specimen` The number of the current specimen being read. For example, if the text file contains data on four specimens, this will be 1, 2, 3, or 4. `FileName` The original file name of the selected file, excluding the extension, for the selected `FileCode`. `FilePath` The file path of an embedded media file, including the filename and extension. `NewLine` Returns the new line character. `Text` Returns the value in the `<Value>` element.
`<Value>`	Optional	Only required if `Property=Text`. The value within this element will be returned.
`<FileCode>`	Optional	Only required if `Property=FileName` or `Property=FilePath`. The file name for this FileCode will be returned.

7.16 NullValues

The Text Importer template allows you to specify a list of global null values, that is, values that will be treated as being equivalent to a null or missing value. During the import, every value read in by the Text Importer is checked. If the value is equivalent to one of the specified null values, this value will be set to null.

Grid functional data is the *only* attribute type where the null values are imported; for all other attribute types, null values are not imported. For grid data, if a value is explicitly set to be equivalent

to null in the template, this value will be imported as a 'not applicable' data point. This will replace any existing data points in the grid with a 'not applicable' data point. No other attribute type or data will replace existing data in the database with null values.

```
<NullValues>
  <NullValue>-99999</NullValue>
  <NullValue>null</NullValue>
  <NullValue>0.00000000</NullValue>
</NullValues>
```

This element is optional and appears under the `<Data>` element in the template.

In the following example, a null value has been set for the point attribute 'Elastic limit' (Figure 10). In this case, the null value will not be imported. If there is an existing value for Elastic limit, this will not be overwritten.

In the same file, the values for a grid functional data attribute (between the 'Start' and 'End' delimiters) contain two grid points with the values '-99999' that will be imported as 'not applicable' data points, overwriting any existing data points in the database.

```
Elastic limit: Null

Grid
Start
                    1       2       3       4       1       2       3       4
                    A       A       A       A       B       B       B       B
        10      |   1.0     1.5     1.8     2.0     1.2     1.7     2.2     3.0
        20      |   1.1     1.6     -99999  2.1     1.3     1.8     2.3     3.1
        30      |   1.3     1.6     1.8     2.2     1.4     1.9     2.4     3.2
        40      |   2.5     3.0     3.5     -99999  3.0     3.5     4.0     4.5
End
```

Figure 10. Example file with null values for a point attribute and a grid functional data attribute

The null values are intended primarily for use with functional data. For other data types, users should try to write regular expressions that only match valid data types. In the above example, if the `<NullValues>` element is not included, the Text Importer would try to import a temperature value of 'null'. This will cause a format exception, as 'null' is not a valid value for Temperature. In the template, the regular expression for temperature should be designed to only match valid numeric values and not to rely on null values.

8 Advanced features

8.1 Merging and appending

The Text Importer can merge functional data and multi-value data, and append multi-value data; see Sections *6.3.1*. You can also use the merge/append capability to delete existing data values for some data types (see Section *8.2*).

8.1.1 Merging functional data

If the incoming functional data is compatible with the existing functional data, then it will be merged.

Merging series float functional data and discrete functional data

- Incoming series with different parameter values (for the non-varying parameter) to the existing data are added.
- Existing series with the same parameter values (for the non-varying parameter) as the incoming data are replaced.
- Existing series with different parameter values (for the non-varying parameter) to the incoming data are kept.

If an existing chart has two stress-strain curves at 300K and 400K, in this case the non-varying parameter is temperature. When a chart with stress-strain curves at 273K and 300K is imported, then the old 300K curve will be replaced with the incoming 300K curve and the 273K curve will be added to the existing curves. The existing 400K curve is kept. The result of the merge is a chart with three stress-strain curves, at 273K, 300K and 400K (see Figure 11).

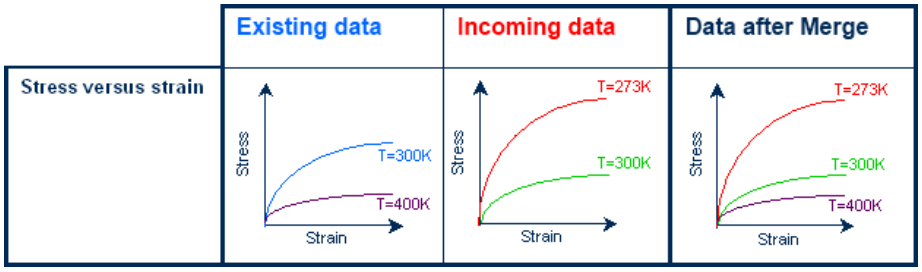


Figure 11 . Example of merging series functional data

When merging series data, the incoming data must have the same parameters as the existing data. In the above example where the existing chart has two stress-strain curves at 300K and 400K, it is not possible to add a series without a temperature value, or add a series with temperature and test-type values.

Merging grid float functional data

- Incoming grid values with different parameter values to the existing data are added.
- Existing grid values with the same parameter values as the incoming data are replaced.
- Existing grid points with different parameter values to the incoming data are kept.

Grid example:

- The example two-dimensional grid has grid values for parameter X=1, X=2, X=3 and parameter Y=55, Y=57, Y=59.
- Grid values for parameter X=4 are added.
- Grid values for (X=3, Y=55) and (X=2, Y=57) are replaced.
- All other grid values are kept (see Figure 12).

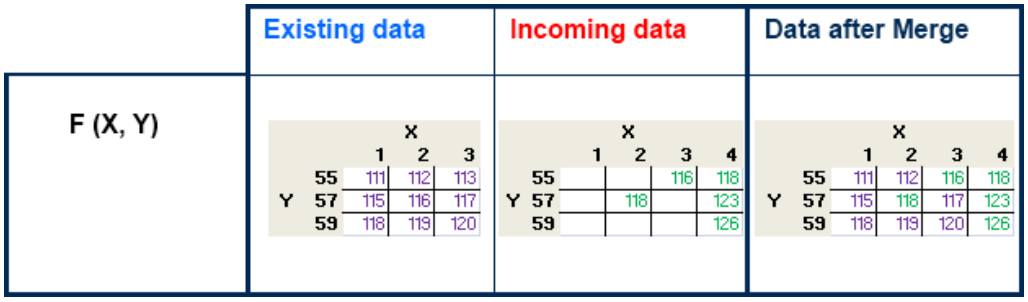


Figure 12. Example of merging grid functional data

When merging grid data, the incoming data must have the same parameters as the existing data. In the above example, it is not possible to add a grid value without a X parameter value, or add a grid value with X, Y and Z parameter values.

Merging equations and logic data

Incoming equations and logic data replaces existing data.

Incompatible functional data

If the incoming functional data is **not** compatible with the existing functional data, then there are two possible actions.

- **Replace** – The existing data is deleted. The incoming data is then imported.
- **Do Not Import** – The incoming data is not imported and the Text Importer will move on to the next data item to be imported.

If the `FunctionalMergeOption` (see Section *6.3.4*) has been set (either in the template or on the command line), this will determine which action is taken. Otherwise

- In MI:Toolbox the user will be offered a choice in the **Cannot Merge Data** dialog.
- In the command-line interface the default behavior is **TryMergeThenReplace.**
- In the Remote Import the default behavior is **TryMergeThenReplace.**

8.1.2 Merging multi-value data

Merging multi-value discrete data

- Existing values are kept.
- Incoming values that are different from the existing values are added.
- Duplicate values will not result from a merge.

Data status	Multi-value Data for the Product Form attribute
Existing data	Bar, Extrusion
Incoming data	Extrusion, Rod, Shapes
Resultant data after merge	Bar, Extrusion, Rod, Shapes

Figure 13. Multi-value discrete data for the 'Product Form' attribute after Merge, in MI:Viewer

Merging multi-value point data

- Existing values are kept.
- Incoming values that are different from the existing values are added. The combination of attribute value plus parameter values is considered as 'one' multi-value.
- Duplicate values will not result from a merge.

Data status	Multi-value Data for the 'Tension strength' attribute	
	Attribute value	Value for Basis parameter
Existing data	456.4 MPa	Mean
	381.3 MPa	A Basis
	411.6 MPa	B Basis
Incoming data	456.4 MPa	Mean
	390.4 MPa	A Basis
	411.6 MPa	B Basis
Resultant data after merge	456.4 MPa	Mean
	381.3 MPa	A Basis
	411.6 MPa	B Basis
	390.4 MPa	A Basis

Tension strength	456.4	MPa (Mean)
	381.3	MPa (A-basis)
	411.6	MPa (B-basis)
	390.4	MPa (A-basis)

Figure 14. Multi-value point data for 'Tension strength' after Merge, in MI:Viewer datasheet

8.1.3 Appending multi-value data

Appending multi-value discrete data

Existing values are kept; incoming values are added; duplicate values can result from an append. For example:

Data status	Multi-value Data for the Product Form attribute
Existing data	Bar, Extrusion
Incoming data	Extrusion, Rod, Shapes
Resultant data after merge	Bar, Extrusion, Extrusion, Rod, Shapes

Product Form	Bar, Extrusion, Extrusion, Rod, Shapes

Figure 15. Multi-value discrete data for the 'Product Form' attribute after Append, in MI:Viewer

Appending multi-value point data

Existing values are kept; incoming values are added; the combination of attribute value plus parameter values is considered as 'one' multi-value; duplicate values can result from an append. For example:

Data status	Multi-value Data for the 'Tension strength' attribute	
	Attribute value	Value for Basis parameter
Existing data	456.4 MPa	Mean
	381.3 MPa	A Basis
	411.6 MPa	B Basis
Incoming data	456.4 MPa	Mean
	390.4 MPa	A Basis
	411.6 MPa	B Basis
Resultant data after merge	456.4 MPa	Mean
	381.3 MPa	A Basis
	411.6 MPa	B Basis

Data status	Multi-value Data for the 'Tension strength' attribute	
	Attribute value	Value for Basis parameter
	456.4 MPa	Mean
	390.4 MPa	A Basis
	411.6 MPa	B Basis

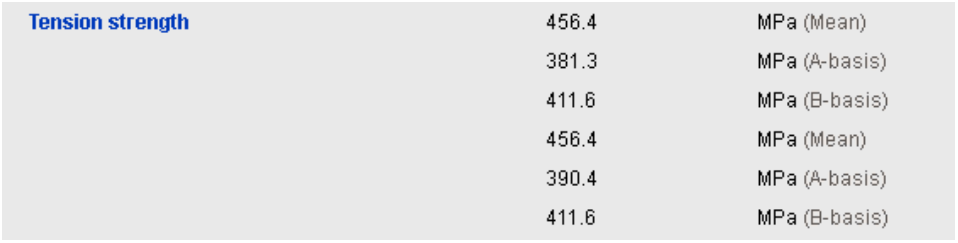


Figure 16. Multi-value point data for 'Tension strength' after Append, in MI:Viewer datasheet

8.2 Deleting existing data

When merging or appending data, it is possible to delete existing data for the following data types: range, single-value point, short text, long text, single-value discrete, picture, integer, date and logical.

This is done by setting the data value being imported to:[-DELETE-]

You must also modify any regular expressions in the template that match data to also recognize the delete string.

For range values, if the maximum or minimum value is set to the delete value, then both values are deleted.

Please note that the data value is deleted. This is not the same as setting the data to 'Not Applicable'.

This feature is not supported for hyperlink, file, multi-value point or multi-value discrete data types. For functional data (float, discrete, equations and logic) the setting does not apply, as functional data can be deleted by omitting it from the incoming data (see Section *6.3.4*).

Suppose the following data is imported:

```
RCSN          Density     RefCode
Record 1      1500        10
Record 2      1200        15
Record 3      2000        25
```

A mistake is discovered and the following data is re-imported:

```
RCSN          Density     RefCode
Record 1
Record 2                  20
Record 3                  [-DELETE-]
```

When asked in the Duplicate Record Name dialog, the 'Merge' option is selected and the resultant data in the records is:

```
RCSN          Density     RefCode
Record 1      1500        10
Record 2      1200        20
Record 3      2000
```

In the Text Importer template, a regular expression that matches a positive integer e.g.

```
Data: (?<group>[0-9]+)
```

must be modified to recognize [-DELETE-] as well e.g.

```
Data: (?<group>([0-9]+)|(\[-DELETE-\]))
```

9 Text Importer templates

In a default installation, the expected location of templates for the Text Importer plug-in is a *Templates* folder under the MI:Toolbox installation folder, typically:

C:\Program Files\Granta\GRANTA MI\Toolbox\plugins\Importers\Text\Templates

Users can add folders in which to look for templates in the **Template Locations** dialog in the MI:Toolbox Text Importer plug-in (opened from the **Edit Template Folders** button in the **Select Templates** dialog). The folder may be a network folder. Folders added with this method are stored with the user's *Documents and Settings* and are specific to the user.

9.1 Configuration of template locations

A configuration file can set the expected location of Text Importer templates for all users on a computer. The default folder can be changed or added to. You will require write permissions to the folder to which MI:Toolbox was installed, typically C:\Program Files\Granta\GRANTA MI\Toolbox.

1. Ensure that the Text Importer plug-in is closed.
2. Create a new folder named *config* in the Text Importer installation folder; typically:
   C:\Program Files\Granta\GRANTA MI\Toolbox\plugins\Importers\Text\config
3. In the config folder, create a new file named TextImporter.Config.xml and open it in an editor.
4. Add the following text to the file.

```
<?xml version="1.0" encoding="utf-8" ?>
  <TextImporterConfig>
    <TemplateLocation>
      <DirectoryPath>\Templates\</DirectoryPath>
    </TemplateLocation>
  </TextImporterConfig>
```

   The `\Templates\` value for the `<DirectoryPath>` element is the default location.
5. To add a folder location, add an additional `<DirectoryPath>` element inside the `<TemplateLocation>` element. The location may be a relative path or a full path. A full path can be a network folder.
6. To remove the original location, delete the element: `DirectoryPath>\Templates\</DirectoryPath>`
7. Save the file.

To check your changes, open the Text Importer plug-in and click **Select Template**. In the **Select Template** dialog, click **Edit Template Folders**. The **Template Location** dialog will list the folders from the configuration file, and any user-defined folders.

Notes

- The configuration change only applies to the individual computer.
- If the configuration file is removed, the default location will return to its single, original location.

Appendix A. Use of regular expressions in the Text Importer

A regular expression (abbreviated as regexp, regex or regxp) is a sequence of characters that define a search pattern used to match strings within a longer piece of text. Further information about regular expressions can be found in a number of books and websites, including the MSDN library.

The regular expressions used by the Text Importer are case insensitive. The import relies on the groups in the regular expression to read the data. The groups in the regular expression are labeled and each regular expression attribute in the template must specify the label of the group where the data is found.

Please note that in the text of this document, the `<` and `>` characters have been used to make the regular expressions easier to read. However, inside an XML element, the characters should be replaced with an entity reference, for example, `<` is replaced by `<`

A.1 Examples

Considering the following text:

```
Test Time: 20 Seconds
```

The following regular expression:

```
Test Time: (?<item>[0-9]+)
```

will match *Test Time: 20* from the text, but not the unit information. In this match, *<item>* will be equal to '20'.

A.1.1 Explicit capture in regular expressions

When using the regular expression attributes, it is necessary to include the name of the group used. This is called an *explicit capture*:

```
<RegularExpression name="Test time">
  <FileCode>file</FileCode>
  <Expression>Test Time: (?<item>[0-9]+)</Expression>
  <Unit>s</Unit>
</RegularExpression>
```

This will import the value '20' into the attribute '`Test time`'. Declaring *<item>* in parentheses in the following example will import 'seconds' into the '`Test time units`' attribute.

```
<RegularExpression name="Test time units">
  <FileCode>file</FileCode>
  <Expression>Test Time: [0-9]+ (?<item>[a-z]+)</Expression>
</RegularExpression>
```

> **Note:** That the ability to match more than one item of data in a regular expression is used by the Text, Functional and Range type attributes. All other data types only expect one item of data, so multiple matches will cause an error.

A.1.2 Range Data examples

Range attributes can store a range of data, a minimum or maximum value only, or a single value (equivalent to point data).

The following example shows text data to be imported into a range attribute called **Test Temperature**.

```
     Test Temperature: 305 - 308
```

The following expression will match both values of the range:

```
Test\sTemperature:\s*(?<ITEM1>([0-9\.\-]+))\s*-\s*(?<ITEM1>([0-9\.\-]+))
```

A `<RegularExpression>` element in a template would look as follows, and would be used within a `<TextReaders>` element:

```
<RegularExpression name="Test Temperature">
  <FileCode>txt</FileCode>
  <Expression>Test\sTemperature:\s*(?<ITEM1>([0-9\.\-]+))\s*-\s*(?<ITEM1>([0-
9\.\-]+))</Expression>
  <Unit>K</Unit>
</RegularExpression>
```

To import an open-ended range (a minimum or maximum value only), the same expression would be used but a null value must be defined in the template and used in the data. For example, the following would be imported as '305 (minimum)'.

Data:

```
     Test Temperature: 305 - 9999
```

Null Value definition:

```
<NullValues>
  <NullValue>9999</NullValue>
</NullValues>
```

If there is only a single value for the data, but it is to be stored in the same range attribute, a more complex expression is required.

Data:

```
     Test Temperature: 305°F;
```

Regular Expression:

```
 (?=Test\sTemperature:\s*(?<ITEM1>([0-9\.\-
]+)))Test\sTemperature:\s*(?<ITEM1>([0-9\.\-]+))
```

A.1.3 Functional data example 1

The following example is a typical stream of text (only the first 10 rows are represented here), generated by a tensile test machine. In this case we would be interested in capturing the second column, which is the measure of Displacement and the third, the Load.

```
     1,     0.000001,    -0.151250
     2,    -0.000001,    -0.265845
     3,     0.000001,     0.077920
     4,    -0.000002,     0.421684
     5,    -0.000001,    -0.609609
     6,    -0.000000,     0.994618
     7,     0.000001,    -0.495014
     8,    -0.000002,    -0.495014
     9,    -0.000002,    -0.380419
    10,    -0.000001,    -0.495014
```

The following expression will match all the rows that constitute the table of values:

```
(\s*[0-9]+,\s*[0-9\.\-]+,\s*[0-9\.\-]\r\n)+
```

In this table, we would like to capture displacement and load values, respectively referenced as Item(2) and Item(3) in the table below. This can be done by adding explicit captures around the second and third numbers.

The following regular expression will return the series of Displacement values:

```
(\s*[0-9]+,\s*(?<ITEM2>[0-9\.\-]+),\s*[0-9\.\-]+\r\n)+
```

The following regular expression will return the series of Load values:

```
(\s*[0-9]+,\s*[0-9\.\-]+,\s*(?<ITEM3>[0-9\.\-]+)\r\n)+
```

In this case, the captures have been named ITEM2 and ITEM3 to be consistent with the following table.

Match	Item(1)	Item(2)	Item(3)
1, 0.000001, -0.151250	1	-0.000001	-0.151250
2, -0.000001, -0.265845	2	-0.000001	-0.265845
3, 0.000001, 0.077920	3	-0.000001	0.077920
4, -0.000002, 0.421684	4	-0.000002	0.421684
5, -0.000001, -0.609609	5	-0.000001	-0.609609
6, -0.000000, 0.994618	6	-0.000000	0.994618
7, 0.000001, -0.495014	7	-0.000001	-0.495014
8, -0.000002, -0.495014	8	-0.000002	-0.495014
9, -0.000002, -0.380419	9	-0.000002	-0.380419
10, -0.000001, -0.495014	10	-0.000001	-0.495014

We now simply need to add the expressions in the template, one for each series of values:

```
<RegularExpression name="Displacement">
  <FileCode>txt</FileCode>
  <Expression>(\s*[0-9]+,\s*(?<ITEM2>[0-9\.\-]+),\s*[0-9\.\-]+\r\n)+
</Expression>
  <Unit>in</Unit>
</RegularExpression>
<RegularExpression name="Test Load">
  <FileCode>txt</FileCode>
  <Expression(\s*[0-9]+,\s*[0-9\.\-]+,\s*(?<ITEM3>[0-9\.\-]+)\r\n)+</Expression>
  <Unit>lbf</Unit>
</RegularExpression>
```

These expressions can be now used in a Functional data builder. The `Test Load` RegularExpression element will be used for Y-axis values of the `Load vs Displacement` attribute whereas the `Displacement` element will be used to feed the X-axis parameter.

Note: For the Functional data builder to be valid:

- Test Load and Displacement must match the same number of points;
- Their units must be compatible (i.e. same dimension) with the attribute and parameter units respectively.

```
<Functional name="Load vs Displacement">
  <FileCode>txt</FileCode>
  <FunctionalType>Series</FunctionalType>
  <YAxis>Test Load</YAxis>
  <XAxis name="Displacement">Displacement</XAxis>
  <Log>false</Log>
  <Interpolate>true</Interpolate>
  <LineType>lines</LineType>
  <Series>false</Series>
</Functional>
```

A.1.4 Functional data example 2

Another more complicated regular expression is needed to read this fatigue data, with the first column as Time, the second as Stress, and the third as Strain:

```
Time Data Start=   0      3.35708e-006 0.0897762
      0.018311     4.86771e-005 0.326187
      0.036622     0.000120853  0.733173
      0.054933     0.000198065  1.25986
      0.073244     0.000303812  1.79852
      0.091555     0.000344096  2.19952
      0.109866     0.000433058  2.71124
      0.128177     0.000523698  3.33369
          …        …       …
      0.622574     0.00387277   23.5393
      0.640885     0.00400679   24.2067
      0.659196     0.00412406   24.7872
```

```
      0.677507     0.00425809   25.3977
      0.695818     0.00438207   26.0022
      0.714129     0.00450604   26.5498
Data End=   0.73244      0.00462499   27.0526
```

The following regular expression is used to match the whole table.

```
Time Data Start=(\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\r)+\s*Data
End=\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+
```

The first part of the expression:

```
Time Data Start=(\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\r)+\s*
```

matches nearly all the data we are interested in. The last part:

```
Data End=\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+
```

is only needed to match the final row. It is also necessary to capture the information we are interested in, for example using an <item> variable.

The first column would be captured by the following expression:

```
Time Data Start=(\s*(?<ITEM>[#/%e0-9\.\-]+)\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-
]+\r)+\s*Data End=\s*(?<ITEM>[#/%e0-9\.\-]+)\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+
```

<ITEM> for this expression will have the following values:

0
0.018311
0.036622
0.054933
0.073244
0.091555
0.109866
0.128177
…
0.622574
0.640885
0.659196
0.677507
0.695818
0.714129
0.73244

The second column would be captured in the same way by:

```
Time Data Start=(\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-]+)\s*[#/%e0-9\.\-
]+\r)+\s*Data End=\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-]+)\s*[#/%e0-9\.\-]+
```

Finally, the third column would be captured by:

```
Time Data Start=(\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-
]+)\r)+\s*Data End=\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-]+)
```

Hence in the same manner as in the previous example, functional data would be imported by first adding the following expressions to the template:

```
<RegularExpression name="Stress">
      <FileCode>FileCode0</FileCode>
      <Expression>Time Data Start=(\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-
]+)\s*[#/%e0-9\.\-]+\r)+\s*Data End=\s*[#/%e0-9\.\-]+\s*(?<ITEM>[#/%e0-9\.\-
]+)\s*[#/%e0-9\.\-]+</Expression>
      <Multiple>true</Multiple>
      <Unit>psi</Unit>
</RegularExpression>

<RegularExpression name="Strain">
      <FileCode>FileCode0</FileCode>
      <Expression>Time Data Start=(\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-
]+\s*(?<ITEM>[#/%e0-9\.\-]+)\r)+\s*Data End=\s*[#/%e0-9\.\-]+\s*[#/%e0-9\.\-
]+\s*(?<ITEM>[#/%e0-9\.\-]+)</Expression>
      <Multiple>true</Multiple>
      <Unit>% strain</Unit>
</RegularExpression>
```

Then using them in a functional data builder:

```
<Functional name="Tensile Response (11 axis)">
      <FunctionalType>Series</FunctionalType>
      <FileCode>txt</FileCode>
      <YAxis>Stress</YAxis>
      <XAxis name="Strain">Strain</XAxis>
      <Interpolate>true</Interpolate>
      <LineType>lines</LineType>
</Functional>
```

A.2 More information on Regular Expressions

For more information on regular expressions, try the following:

- Regular Expression Library at http://regexlib.com.
- The Cheat Sheet (http://regexlib.com/CheatSheet.aspx) contains a list of matches for .Net.
- To test your regular expression, go to http://regexlib.com/RETester.aspx, enabling the options: Case Insensitive, Multiline, and .Net Regex Engine.

Appendix B. Grid functional data example

Figure 17 shows an example of grid functional data with five parameters:

- Single
- Row
- Column 1
- Column 2
- Grid Parameter

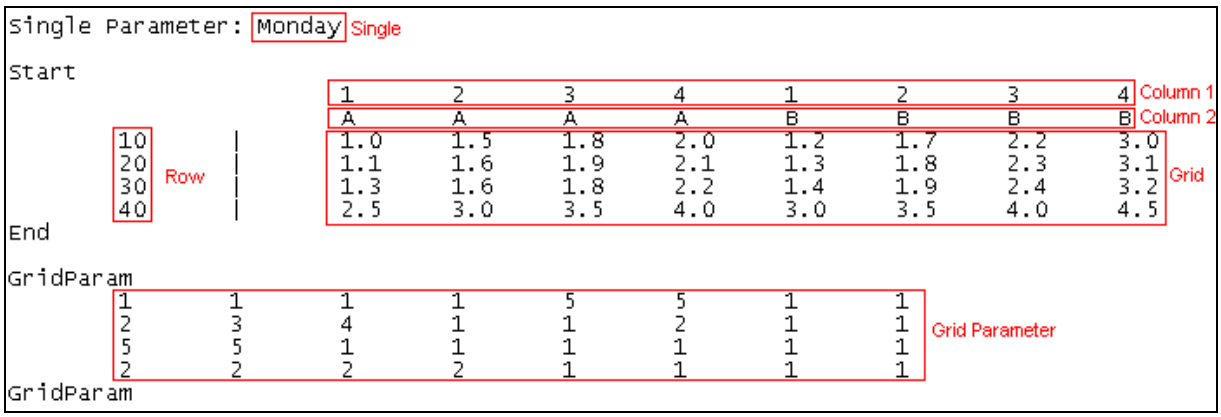


Figure 17. Example of grid functional data

The red outlines and text are labels for clarity, and are not part of the data. There are four types of parameter values for Grid data: Single, Row, Column, and Grid.

The **single** type is used when there is only one value for this parameter that is used for all data points. The value “Monday” is an example of a single parameter type.

Row parameter types are used when every row in the grid is to be assigned the same parameter value.

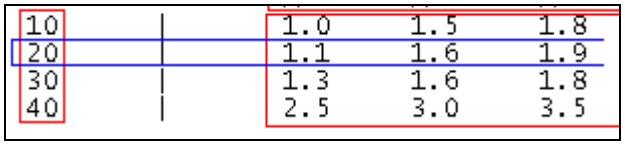


Figure 18. Example of Row parameter type

Here, every data point in the row highlighted in blue is assigned the value “20” for the parameter.

The **column** type is used when every data point in a column is assigned the same value.

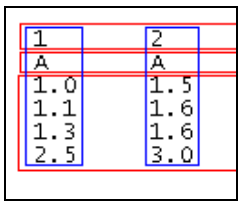


Figure 19. Example of Column parameter type

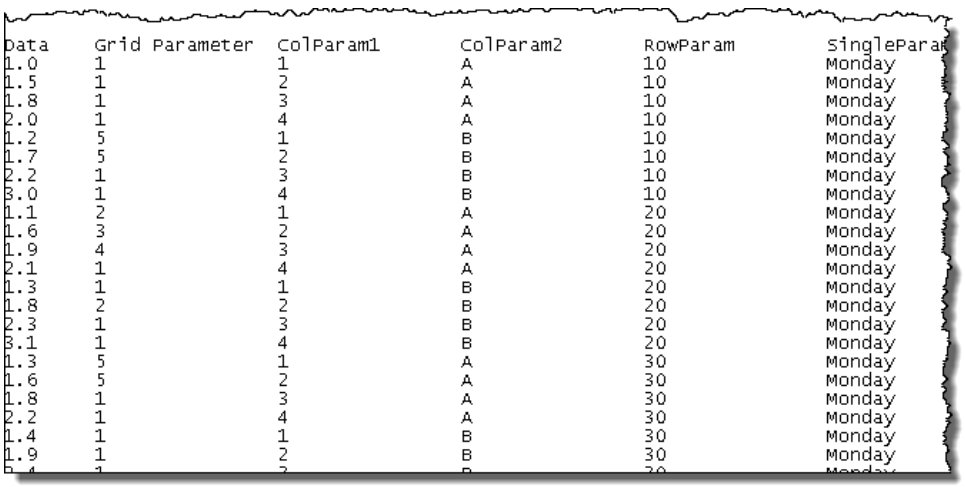
In the example shown in Figure 19, every data point in the first row will have value “1” for parameter Column 1 and value “A” for parameter Column 2. Every value in the second row will have value “2” for parameter Column 1 and value “A” for parameter Column 2.

The **grid** parameter type is used when there is an individual parameter value for every data point. In Figure 18 you can see that there are an equal number of values in the Grid parameter as there are in the Grid data values.

The template element that would be used to build this grid functional data is as shown:

```
<GridFunctional name="Grid Functional Data">
  <DefaultXParameter>Column 1</DefaultXParameter>
  <DataGrid>Grid</DataGrid>
  <GridParameter>
    <Parameter>Single Parameter</Parameter>
    <ParameterType>Single</ParameterType>
  </GridParameter>
  <GridParameter>
    <Parameter>Row</Parameter>
    <ParameterType>Row</ParameterType>
  </GridParameter>
  <GridParameter>
    <Parameter>Column 1</Parameter>
    <ParameterType>Column</ParameterType>
  </GridParameter>
  <GridParameter>
    <Parameter>Column 2</Parameter>
    <ParameterType>Column</ParameterType>
  </GridParameter>
  <GridParameter>
    <Parameter>Grid Parameter</Parameter>
    <ParameterType>Grid</ParameterType>
  </GridParameter>
</GridFunctional>
```

The grid created in the database from the data in Figure 18 would look like this:



Data	Grid Parameter	ColParam1	ColParam2	RowParam	SinglePara
1.0	1	1	A	10	Monday
1.5	1	2	A	10	Monday
1.8	1	3	A	10	Monday
2.0	1	4	A	10	Monday
1.2	5	1	B	10	Monday
1.7	5	2	B	10	Monday
2.2	1	3	B	10	Monday
3.0	1	4	B	10	Monday
1.1	2	1	A	20	Monday
1.6	3	2	A	20	Monday
1.9	4	3	A	20	Monday
2.1	1	4	A	20	Monday
1.3	1	1	B	20	Monday
1.8	2	2	B	20	Monday
2.3	1	3	B	20	Monday
3.1	1	4	B	20	Monday
1.3	5	1	A	30	Monday
1.6	5	2	A	30	Monday
1.8	1	3	A	30	Monday
2.2	1	4	A	30	Monday
1.4	1	1	B	30	Monday
1.9	1	2	B	30	Monday

Figure 20. The data grid generated by the Text Importer

Appendix C. Text Importer command-line interface (CLI)

The Text Importer can import data from text files into a GRANTA MI database without the need for user interaction. This allows the Text Importer to run as a scheduled task.

C.1 Running the Text Importer plug-in from the command line

The Text Importer plug-in executable is located in the *bin* subfolder in the MI:Toolbox installation folder, typically:

C:\Program Files\Granta\GRANTA MI\Toolbox\bin\

To run the Text Importer from the command line: at the command prompt, type the following:

MIToolbox.Console.exe -plugin Granta.TextImporter

The Text Importer requires several command-line arguments to provide the information that is supplied by the user when using the plug-in inside MI:Toolbox. These include:

- The plug-in name, Granta.TextImporter
- The name of the server where the GRANTA MI service is running.
- The authentication details used to connect to the GRANTA MI Service (if System authentication is not used). The user will require at least write privileges to the database.
- The database into which the new records will be imported, specified with the database key.
- The location of the data source (text) files.
- The path to a template that will be used to import the data files into the database. The template must have auto-placement options set.

The MI:Toolbox application should be closed before running a plug-in from the command line.

C.2 Command-line arguments

Arguments containing spaces, colons, or equals should be double quoted e.g.

-username "Peter Parker"
-inputfile "C:\Users\ali.kim\Documents\data imports\SS import.xls

Argument	Required?	Description
-plugin Granta.TextImporter	Yes	Specifies the name of the plug-in.
-server <*servername*>	Yes	Specifies the hostname of the server where the GRANTA MI Service is running.

Argument	Required?	Description
-dbkey <*key*>	Yes*	*(*Required when the template does not include auto-placement)* The database key of the database into which the data will be imported. If the importer template includes auto-placement options, the database key does not need to be specified on the command line, and if it is specified, it will be ignored.
-table <*tablename*>	Yes	The name of the table into which the data will be imported.
-sourcedata <*folder*>	Yes*	(**Required if importing data from multiple files*) The location of a folder containing all the source data to be imported. When importing data from a single file, you can use the -sourcefile argument instead.
-sourcefile <*filepath*>	Yes*	(**Required if importing data from a single file*) The location of a file containing all the source data to be imported. When importing data from multiple files, use the -sourcedata argument instead to specify the folder containing the source files.
-template	Yes	The template to use for the import.
-conflictresolution <*value*>	No	Specifies how record name conflicts on the tree should be dealt with. The values are not case sensitive. The default behavior is Merge. Permitted values: • DoNotImport • Replace • Merge • Append See the table in section *6.3.3* for details of the behavior of each option (where the table refers to `FunctionalMergeOption`, -functionaldatamerge is the command-line equivalent).
-functionaldatamerge <*value*>	No	Specifies how functional data is handled when there is existing data. This is only applicable if the conflict

Argument	Required?	Description
		resolution value is *Merge* or *Append*. The values are not case sensitive. The default behavior is *TryMergeThenReplace*. Permitted values: • Replace • MergeReplace • MergeDoNotImport See the table in section *6.3.4* for details of the behavior of each option
-help	No	Displays usage information for the application. If the plug-in is also specified, it will include the usage information of the specified plug-in
-logfile <*filepath*>	No	The name and location of the log file. If omitted, no log file will be generated.
-loglevel <*level*>	No	Sets the verbosity of logging sent to the command window console. (Note that this setting has no effect on the contents of the log file). Permitted values (in order of increasing verbosity) are: ERROR, WARN, INFO, DEBUG, TRACE, ALL (case-insensitive). If omitted, the default value is INFO.
-nologo	No	Prevents the display of application information on startup.
-username <*name*> -domain <*domain*> -password <*password*>	No*	(*Optional only when using Windows authentication) The authentication details used to connect to the GRANTA MI Service. A user account with at least write privileges to the database is required. These may be omitted from the command line if Windows authentication is being used; in this case, the import is performed with the user account used to log on to the MI:Toolbox host. Alternatively, you can perform the import under a different Windows user account by specifying the credentials of that account.

C.3 Usage example

```
C:\Program Files\Granta\Granta MI\Toolbox\bin\MIToolbox.Console.exe
-plugin Granta.TextImporter -logfile "C:\Users\kim.smith\Documents\Console.log"
-nologo -server localhost -username "kim.smith" -password MyPa55w0rd
-domain AcmeCorp -dbkey MI_Lab -conflictresolution Replace
-template "C:\Program Files\Granta\Granta MI\Toolbox\plugins\Importers\Text\
Templates\Tensile-Instron-SeriesIX.xml" -sourcedata "C:\TestFiles\Instron Tensile"
```

C.4 Troubleshooting

The following error codes may help you to troubleshoot problems that arise when running the Text Importer from the command line.

Code	Description
0	Success - Indicates successful execution of the MI:Toolbox console application.
1	Unspecified error - Indicates that an unspecified error was encountered.
2	Unhandled exception - Indicates that an unhandled exception was caught.
3	Invalid command-line option value - Indicates a problem with one of the specified command-line options (e.g. an option specifies a table or record that does not exist).
4	Syntax error in command-line option - Indicates a basic syntax error with one of the specified command-line options.
5	Could not load plugin - Indicates that the named plugin could not be loaded.
6	Plugin initialization error - Indicates an error during the initialization phase of the plugin, typically caused when trying to validate an option against data in a database (e.g. a database specified by a particular dbkey may not exist).
7	Cannot open the log file - Indicates that the log file could not be opened.
8	Plugin found no work to do - Indicates that the plugin initialization found nothing to import/export
1100	Failed to establish a connection to an MI:Server - Indicates that a connection to an MI:Server instance could not be established.
1110	No default database available - Indicates that the server does not have a default database, or the default database is unavailable.
1111	No such database - Indicates that the specified database does not exist on the server, or is otherwise unavailable (e.g. due to access control).

Code	Description
1112	Failed to retrieve database - Indicates that the specified database could not be retrieved from the server.
1120	No such record - Indicates that the record does not exist on the server, or is otherwise unavailable (e.g. due to access control).
1121	Failed to retrieve record - Indicates that the specified record could not be obtained from the server.
1122	Multiple records matched - Indicates that multiple matching records have been found that satisfy criteria intended to identify a single record.
1130	No such subset - Indicates that the one or more specified subsets do not exist or are otherwise unavailable on the server (e.g. due to access control).
1140	No default table - Indicates that the database does not contain a default table, or the default table is otherwise unavailable (e.g. due to access control).
1141	No such table - Indicates that the database does not contain a default table, or the default table is otherwise unavailable (e.g. due to access control).