

X2TL - X2A2 Templating Language - HTML Templating Language and Engine (Reflection Class)

[Print](#)

» [Home](#) » [X2TL - X2A2 Templating Language - HTML Templating Language and Engine \(Reflection Class\)](#)

X2TL - HTML Templating Language and Engine (Reflection Class)

Use simple HTML Templating to apply standard formatting / macro expansion

Table of Contents [[Hide/Show](#)]

[X2TL - HTML Templating Language and Engine \(Reflection Class\)](#)

[Overview](#)

[Requirements](#)

[Additional Files](#)

[Understanding HTML Templating](#)

[TransformNode and its Parameters](#)

[pSourceNode](#)

[pTemplateNode](#)

[pSupportingTemplatesNode](#)

[X2TL Delimiters](#)

[Escaped Delimiters](#)

[Many Escaped Delimiters](#)

[X2TL Commands](#)

[Command Delimiters](#)

[X2TL Command Set](#)

[Value-Replacement Commands](#)

[Result \(=\)](#)

[RawResult \(==, raw\)](#)

[Format \(?\)](#)

[Format Date \(> Date\)](#)

[Format Number \(> Number\)](#)

[Format String \(> String\)](#)

[.Net Framework Format Strings](#)

[Node-Replacement Commands](#)

[Copy \(*\)](#)

[CopyEncoded \(*=\)](#)

[Conditional Commands](#)

[If ... Else ... EndIf \(/If\)](#)

[Looping Commands](#)

[Each ... EndEach \(/Each\)](#)

[Template Commands](#)

[Apply \(%\)](#)

[ParamApply \(%%\)](#)

[Variable Commands](#)

[Variable \(var, :=\)](#)

[MultiVar \(mvar, ::=\) ... EndMultiVar \(endmvar, /MultiVar, /mvar, /::=\)](#)

[Manipulation Commands](#)

[Replace \(~\)](#)
[Miscellaneous Commands](#)
[Version \(ver\)](#)
[See Also](#)

Overview

This Class/DLL provides a single public method [TransformNode](#) which takes a **contextNode** and a **templateNode** and optionally a **supportingTemplatesNode**. The inner xml of the template is then parsed as a string, based on starting and ending delimiters. Text found inside of the delimiters are considered **Command Fragments**, and text outside of the delimiters is considered **Text Fragments**. The list of fragments are then sequentially processed, with **Text Fragments** simply being copied as is, and the **Command Fragments** being analyzed individually and processed. Each of commands are applied against the context node, unless the command itself changes the context. Templates are also allowed to reference other templates, to allow for building of complex templates out of simple templates. The **supportingTemplatesNode** is expected to be a flat list containing a single child node for each **named template**. Templates can be nested as much as the stack space will allow.

[Top](#)

Requirements

This Class library requires the DLL to be loaded as a Reflection Class.

```

1  <connection name="X2TL" description="X2 Templating Language">C:\OneRecord\FrontEnd\CoreComponents\
2
3  ...
4
5  <group name="X2TL" description="desc">
6    <mapper name="TransformNode" connection="X2TL" type="RelWare.X2TL" method="TransformNode" descript
7      <description>Transforms a Node using a Primary Template and any Supporting Templates</descripti
8      <params>
9        <param name="pSourceNode" datatype="System.Xml.XmlNode" type="xpath" format="xml" description=
10       <param name="pTemplateNode" datatype="System.Xml.XmlNode" type="xpath" format="xml" descripti
11       <param name="pSupportingTemplatesNode" datatype="System.Xml.XmlNode" type="xpath" format="xml"
12     </params>
13   </mapper>
14 </group>


```

[Top](#)

Additional Files

To fully test the capabilities of this control, the Solution contains a unit-test application which runs as a console application, dumping its results to standard output for verification.

Source code for this can be found in VSS at the following paths:

1	\$\Supporting Components\RelWare\X2TL	- Main branch for entire solution	
2	\$\Supporting Components\RelWare\X2TL\X2TL	- Code for the actual X2TL Class	
3	\$\Supporting Components\RelWare\X2TL\X2TL_UnitTesting	- Code branch for unit testing application	

*NOTE: While the unit testing still needs to be converted to true **NUnit**, there are several working examples contained within those test cases. Additionally, all of the code samples shown on this page are contained as*

samples as well, validating that the documented examples are unit tested as well.

[Top](#)

Understanding HTML Templating

HTML Templating is used to expand out data in a standardized formatting and styling manner. Generalized information on HTML Templating can be found at [Wikipedia - Web Template System](#). The templates are made of either [Text Fragments](#) or [Command Fragments](#).

Command Fragments are determined by finding blocks wrapped in a start and end delimiter. By default these are `{{` and `}}` respectively, referred to in the templating community as [Mustaches](#). Within the "mustaches" are commands that are evaluated and processed against the context node. Anything outside of the "mustaches" is considered a **Text Fragment** and is copied as-is.

A simple example of an HTML Template is as follows:

```
1 <span>{{ = row/@lastName}}, {{ = row/@firstName }} ({{ = row/@userName }})</span>
```

When given an XML context like the following:

```
1 <Patient><row lastName="Fine" firstName="Howard" userName="HFINE"/></Patient>
```

it results in the following output:

```
1 <span>Fine, Howard (HFINE)</span>
```

[Top](#)

TransformNode and its Parameters

As noted above, the primary method for this class is `TransformNode`, which is used to apply a given template against an XML node, optionally using other supporting templates, and return the resulting string. The method takes three parameters as its inputs: **pSourceNode**, **pTemplateNode**, and **pSupportingTemplatesNode**, each being an `XmlNode` datatype.

[Top](#)

pSourceNode

This is the context node that the template will be applied against. In our examples, this will most likely be against some subset of the DataCloud. It has no set structure and is purely dependant upon the template and possibly any supporting templates that are referenced.

pSourceNode Example

```
1 <Common.ADM_LoginMessagesGet>
2 <row MessageNo="1001" Subject="Sample bulletin board message" isAdmin="true" />
```

```
3 | </Common.ADM_LoginMessagesGet>
```

[Top](#)

pTemplateNode

This node contains the template to be applied. This contains a root node that wrappers the XML/HTML contents of the template. This should be an entire HTML fragment that you wish to have manipulated, with the delimited commands contained anywhere within the HTML. As you can see below, this is an entirely valid block of HTML. However, once the engine processes it, the commands will be replaced with their representative data.

pTemplateNode Example

```
1 | <template>
2 |   <div>
3 |     {{ IF (count(row)!=0) }}
4 |       <table>
5 |       <tbody>
6 |         {{ APPLY BulletinBoardRow row }}
7 |       </tbody>
8 |     </table>
9 |     {{ ELSE }}
10 |      No bulletin messages
11 |    {{ /IF }}
12 |   </div>
13 | </template>
```

*NOTE: The name of the root node is ignored within all logic of the **X2TL**. It is ONLY referred to as **<template>** for readability.*

[Top](#)

pSupportingTemplatesNode

If the primary template, or any subsequent templates, being applied incorporates the APPLY command, then this parameter will be expected to point to a valid nodeset. Otherwise, it will be ignored. If supporting templates are referenced, then the name of the template being applied will be searched within the

pSupportingTemplatesNode Example

```
1 | <templates>
2 |
3 |   <template name="BulletinBoardRow">
4 |     <tr>
5 |       <td id="Bulletin_{{ = @MessageNo }}">{{ = @Subject }}

```

NOTE: The name of the root node as well as the name of the child nodes below it are ignored within all logic of the

[Top](#)

By default, the **X2TL** engine uses the `{{` and `}}` as start and end delimiter respectively. These are referred to as **mustaches**. These can be overridden within the Class object by calling instantiating the object with a different delimiter set. However, at this time, this can NOT be achieved within the Reflection Class itself. It is still being debated upon whether to provide a separate method that allows specifying a different start and end delimiter, or providing a command to change delimiters mid-stream. However, at the time of this writing, we have limited ourselves to the mustaches approach.

Page 5 of 20

X2TL Commands

The **X2TL** has a limited command set, however, its flexibility allow for countless variations. The **X2TL** currently supports the following [commands](#), with the syntax for each command being described in detail below.

Please also note that some commands provide for one or more "shorthand" alternatives. These are shown in parenthesis following the command itself (i.e. Result (=)). Shorthand versions of commands are treated identical to the engine, and it does not distinguish between either writing of the command. It should be noted that **X2TL** command words themselves are NOT case-sensitive, however, of course, their XPath parameters are. With that, the following are multiple ways of writing the same command, each of which will provide identical output.

```

1  {{ RESULT node/@attrib }}
2  {{ Result node/@attrib }}
3  {{ rRESULT node/@attrib }}
4  {{ result node/@attrib }}
5  {{ = node/@attrib }}
```



[Top](#)

Command Delimiters

A command itself may be broken down into **parameters**, the first of which is always the command name. Some commands do not require any parameters, and some can take any number of parameters. To properly determine the parameters within the command fragment, **X2TL** understands the following **command delimiters**: double-quotes ("), parenthesis ((and)), and whitespace (space, newline, tab). Any text contained within a quoted-string, or within an outer set of parenthesis, is considered its own parameter. Any other text is broken down via whitespace.

[Top](#)

X2TL Command Set

- [Value-Replacement Commands](#)
 - [Result \(=\)](#)
 - [RawResult \(==, raw\)](#)
 - [Format \(?\)](#)
 - [Date](#)
 - [Number](#)
 - [String](#)
- [Node-Replacement Commands](#)
 - [Copy \(*\)](#)
 - [CopyEncoded \(*=\)](#)
- [Conditional Commands](#)
 - [If ... Else ... EndIf \(/If\)](#)
- [Looping Commands](#)
 - [Each ... EndEach \(/Each\)](#)
- [Template Commands](#)
 - [Apply \(%\)](#)
 - [ParamApply \(%%\)](#)
- [Variable Commands](#)
 - [Variable \(var, :=\)](#)

- [MultiVar \(mvar, ::=\) ... /MultiVar \(/mvar, /::=, endmultivar, endmvar\)](#)
- [Manipulation Commands](#)
 - [Replace \(~\)](#)
- [Miscellaneous Commands](#)
 - [Version \(ver\)](#)

[Top](#)

Value-Replacement Commands

- [Result \(=\)](#)
- [RawResult \(==, raw\)](#)
- [Format \(?\)](#)
 - [Date](#)
 - [Number](#)
 - [String](#)

[Top](#)

Result (=)

The **Result** command, which has a shorthand equivalent of `=`, is the most commonly used command, as value-replacement is primarily the point of a templating language. It expects a single parameter being an XPath within the current node context to locate the value that you wish resulted. It should be noted that the resulting value of the XML node will be XML-escaped before it is added to the resulting string. ([see RawResult below for more information](#))

NOTE: This is "value-replacement" only. For full "node-replacement", please refer to the [Copy \(\)](#) command.*

Result Example: Context Node

```
1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18"/>
3 </User>
```

?

Result Example: Template Sample

```
1 <template>
2   <div id="User_{{ = @userNo }}">
3     <span>{{ result @lastName }}, {{ result @firstName }} ({{ = @userName }})</span>
4     <span>User is {{ = Check-In/@status }} - ( {{ = /User/Check-In/@status }} ) as of {{ Result "*"@1
5   </div>
6 </template>
```

?

Result Example: Output

```
1 <div id="User_1001">
2   <span>Fine, Howard (hfine)</span>
3   <span>User is CHECKED-IN - ( Busy ) as of 8/31/2010 16:18</span>
4 </div>
```

?

[Top](#)

RawResult (==, raw)

The **RawResult** command, which has two shorthand equivalents of **==** and **raw**, will be lesser used than its counterpart **Result**, as RawResult provides back its resulting data as-is in string form, without any XML escaping performed. To better illustrate the use of this command, it will be compared to the **Result** command in the example below.

RawResult Example: Context Node

```
1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Pref favURL="http://www.google.com/#hl=en&source=hp&q=RelWare&aq=f&aqi=g1g-s1"/>
3 </User>
```

RawResult Example: Template Sample

```
1 <template>
2   <div>
3     <span>User's favorite URL is: {{ Result Pref/@favURL }}</span>
4     <span>This is Invalid XML:    {{ RawResult Pref/@favURL }}</span>
5
6     <hr />
7     <span>User's favorite URL is: {{ = Pref/@favURL }}</span>
8     <span>This is Invalid XML:    {{ == Pref/@favURL }}</span>
9   </div>
10 </template>
```

Result Example: Output

```
1 <div>
2   <span>User's favorite URL is: http://www.google.com/#hl=en&source=hp&q=RelWare&aq=f&aqi=g1g-s1</span>
3   <span>This is Invalid XML:    http://www.google.com/#hl=en&source=hp&q=RelWare&aq=f&aqi=g1g-s1</span>
4
5   <hr />
6   <span>User's favorite URL is: http://www.google.com/#hl=en&source=hp&q=RelWare&aq=f&aqi=g1g-s1</span>
7   <span>This is Invalid XML:    http://www.google.com/#hl=en&source=hp&q=RelWare&aq=f&aqi=g1g-s1</span>
8 </div>
```

[Top](#)

Format (?)

The **Format** command, which has a shorthand equivalent of **?**, is similar to the **Result** command, except that it is used to specifically change the output format of a given value, or in the case of a **String** format, multiple variables. The **Format** command expects (3) parameters: datatype (**date**, **number**, **string**), format ([see .Net Framework Format Strings](#)), and value (XPath reference to node value).

Unlike other commands, **Format** also has three 'shortcuts', being that you can omit the Format (or ?) portion of the command, and simply use the commands: **Date**, **Number** or **String**, each of which are described in more detail below. Other than the obvious lack of initial parameter, they operate identically.

Format Example: Context Node

```
1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18" phoneNum="1234567890">
3 </User>
```


Format Example: Template Sample

```

1 <template>
2   <span>{{ Each Check-In }}User is {{ = @status }} as of {{ Format Date MM/dd/yyyy @lastUpdate }}
3   and can be reached at {{ ? Number "#(###)###-#### x.####" @phoneNum }}{{ /Each }}</span>
4 </template>
```

Format Example: Output

```

1 <span>User is CHECKED-IN as of 08/31/2010
2 and can be reached at 1(234)567-8901 x.2345</span>
```

[Top](#)

Format Date (--> Date)

The **Date** command is a shortcut to the fully qualified **Format Date** command. It expects to receive a valid date format string, and an XPath value that points to a date value. At the time of this writing, there is no error handling for this, nor "invalid date message". Please use with care.

Note: While the command name only specifies Date, it is truly a DateTime format, and supports any .Net supported date/time string.

(see [Supported Date Format Strings](#) and [Custom Date Format Strings](#))

Date Example: Context Node

```

1 <SystemTime utc="8/26/2010 5:44:47 PM" local24="08/26/2010 13:44:47" />
```

Date Example: Template Sample

```

1 <template>
2   <span>UTC: {{ Format Date MM/dd/yyyy @utc }} - Local: {{ ? Date MM/dd/yyyy @local24 }} - English: {
3 </template>
```

Date Example: Output

```

1 <span>UTC: 08/26/2010 - Local: 08/26/2010 - English: August 26, 2010 - Time: 1:44 PM</span>
```

[Top](#)

Format Number (--> Number)

The **Number** command is a shortcut to the fully qualified **Format Number** command. It expects to receive a valid number formatted string, and an XPath value that points to a numeric value (either Int64 or Double). At the time of this writing, there is no error handling for this, nor "invalid numeric message". Please use with care.

(see [.Net Standard Number Format strings](#) and [Custom Numerical Format Strings](#))

Number Example: Context Node

```

1 <node>
2   <data n0="0" n1="123" n2="1234567890" n3="-765.4321" n4="12345678901.2345"/>
3 </node>
```

Number Example: Template Sample

```

1  <template>
2    <div>
3      <span>Original: {{ = data/@n1 }} Zero-padded: {{ format Number 000000 data/@n1 }}</span>
4      <span>Comma-Separated: {{ ? Number "#, #" data/@n2 }}</span>
5      <span>Negative currency: {{ Number "$#, #.##" data/@n3 }}</span>
6      <span>Telephone Ext: {{ Number "#(###)###-#### x.####" data/@n4 }}</span>
7      <span>Pos: {{ number "##.##; (##.##); **Zero**" data/@n1 }} Neg: {{ number "##.##; (##.##); **Zero**" data/@n1 }}
8      <span>Dec: {{ = data/@n1 }} Hex: {{ ? number X data/@n1 }}</span>
9    </div>
10 </template>

```

Number Example: Output

```

1  <div>
2    <span>Original: 123 Zero-padded: 000123</span>
3    <span>Comma-Separated: 1,234,567,890</span>
4    <span>Negative currency: -$765.43</span>
5    <span>Telephone Ext: 1(234)567-8901 x.2345</span>
6    <span>Pos: 123 Neg: (765.4) Zero: **Zero**</span>
7    <span>Dec: 123 Hex: 7B</span>
8  </div>

```

[Top](#)

Format String (--> String)

The **String** command is a shortcut to the fully qualified **Format String** command. It expects to receive a format string, and unlike the other commands, any number of XPath values, each pointing to their own string value nodes. This command uses the .Net String.Format() method for its internal resulting. At the time of this writing, there is no error handling for invalid format strings. Please use with care.

String Example: Context Node

```

1  <node>
2    <data user="HFINE" first="Howard" last="Fine" d1="8/26/2010 2:30:51 PM" n0="0" n1="123" n2="1234567">
3  </node>

```

String Example: Template Sample

```

1  <template>
2    <span>{{ format string "Name: {0}, {1} ({2})" data/@last data/@first data/@user }}</span>
3  </template>

```

String Example: Output

```

1  <span>Name: Fine, Howard (HFINE)</span>

```

[Top](#)

.Net Framework Format Strings

- [.Net Standard Date/Time Format strings](#)

- [.Net Custom Date/Time Format strings](#)
- [.Net Standard Number Format strings](#)
- [.Net Custom Number Format strings](#)
- [.Net Composite String Formatting](#)

Node-Replacement Commands

- [Copy \(*\)](#)
- [CopyEncoded \(*=\)](#)

[Top](#)

Copy (*)

The **Copy** command, which has a shorthand equivalent of *****, is used to bring a copy of the actual XML of a selected node into the result. Unlike the above Value-Replacement commands, which work only on the requested node's inner text / value, this actually brings forward full "outer-XML" of a requested node. With that, no additional formatting options apply.

Result Example: Context Node

```

1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18" />
3 </User>
```



Result Example: Template Sample

```

1 <template>
2   <xml id="User_{{ = @userNo }}">
3     {{ copy Check-In }}
4   </xml>
5 </template>
```



Result Example: Output

```

1 <xml id="User_1001">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18" />
3 </xml>
```



[Top](#)

CopyEncoded (*=)

The **CopyEncoded** command, which has a shorthand equivalent of ***=**, is used to bring a XML-encoded copy of the

actual XML of a selected node into the result. Unlike the above Copy command, the inner XML node or nodeset is encoded to be able to be dealt with as a string (< becomes **<**).

Result Example: Context Node

```
1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18" />
3 </User>
```

Result Example: Template Sample

```
1 <template>
2   <xml id="User_{{ = @userNo }}">
3     {{ copyencoded Check-In }}
4   </xml>
5 </template>
```

Result Example: Output

```
1 <xml id="User_1001">
2   &lt;Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010
3 </xml>
```

[Top](#)

Conditional Commands

- [If ... Else ... EndIf \(/If\)](#)

[Top](#)

If ... Else ... EndIf (/If)

Any language worth its salt has some form of the If//Else/EndIf logic, and **X2TL** is no different. Much like the **@test** attribute of an X2A2 <dataio>, the **If** command takes an XPath parameter that is used to perform a "does-node-exist" test. The test can consist of a simple relative XPath, a root-node based XPath, or a logic-based XPath, each of which will be shown below. Like most languages, the **Else** command is optional, however, the **EndIf** (or its shorthand equivalent **/If**) is required. In the future, an error message will most likely result in this situation, however at present, it may simply provide unpredictable results.

If / EndIf Example: Context Node

```
1 <User userNo="1001" lastName="Fine" firstName="Howard" userName="hfine">
2   <Check-In status="CHECKED-IN" availability="Busy" lastUpdate="8/31/2010 16:18">
3     <message>In meetings until lunch</message>
4   </Check-In>
5 </User>
```

Result Example: Template Sample

```
1 <template>
2   <div id="User_{{ = @userNo }}">
3     <span>{{ result @lastName }}, {{ result @firstName }} ({{ = @userName }})</span>
4     {{ IF /User/Check-In }}
```

```

5      <span>User is {{ = Check-In/@status }}
6      {{ If Check-In/@facility }}at the {{ = Check-In/@facility }}{{ Else }}but whereabouts are ur
7      </span>
8      {{ if (count(*/*message)!=0) }}
9      <div>{{ = /*message }}</div>
10     {{ /if }}
11     {{ ENDIF }}
12 </div>
13 </template>

```

*NOTE: In the above example, we see each of the three types of XPath tests: **root-node based** (*IF /User/Check-In*), **relative** (*If Check-In/@facility*), and **logical** (*if (count(*/*message)!=0)*).*

Result Example: Output

```

1 <div id="User_1001">
2   <span>Fine, Howard (hfine)</span>
3   <span>User is CHECKED-IN
4     but whereabouts are unknown
5   </span>
6   <div>In meetings until lunch</div>
7 </div>

```

[Top](#)

Looping Commands

- [Each](#) ... [EndEach \(/Each\)](#)

[Top](#)

Each ... EndEach (/Each)

As with the simple conditional logic, one cannot get far in templating without providing a simple looping logic. More complex conditional looping commands such as Do...Until and While...EndWhile may come later, but until then, the simple **Each** command is provided. **Each** acts almost identically to XSL's for-each command, taking an XPath as its parameter, which can either be a relative XPath against the context node, or a root-based XPath which will be applied against the context node's document root element. Each XmlNode matching the given XPath will then be "looped", with any **Text** or **Command Fragments** contained within the **Each** ... **EndEach** block (or its shorthand equivalent of **/Each**) being repeated against the looped node as their context node. As with If/EndIf above, there is presently no error checking on un-matched Each/EndEach. While nesting of Each commands is valid, if you miss an EndEach command, the results may be unpredictable.

Each / EndEach Example: Context Node

```

1 <Patient lastName="Brown" firstName="Charlie">
2   <Allergies>
3     <row AllergyNo="1001" description="PEANUTS" reaction="Hives"/>
4     <row AllergyNo="1002" description="LUCYVANPELT" reaction="Irritation"/>
5   </Allergies>
6 </Patient>

```

Each / EndEach Example: Template Sample

```

1 <template>

```

```

2      <div>{{ = @lastName}}, {{ = @firstName }}
3      <table>
4      <tbody>
5          {{ EACH Allergies/row }}
6          <tr><td>{{ = @description }}</td><td>{{ = @reaction }}</td></tr>
7          {{ ENDEACH }}
8      </tbody>
9      </table>
10     </div>
11 </template>

```

(Alternate Writing - Nested Each commands)

```

1  <template>
2      <div>{{ = @lastName}}, {{ = @firstName }}
3      <table>
4      <tbody>
5          {{ EACH Allergies/row }}
6          <tr>{{ Each (@*[name()!='AllergyNo']) }}<td>{{ = . }}</td>{{ /Each }}</tr>
7          {{ /EACH }}
8      </tbody>
9      </table>
10     </div>
11 </template>

```

Each / EndEach Example: Output (both examples above provide identical output)

```

1  <div>Brown, Charlie
2      <table>
3      <tbody>
4          <tr><td>PEANUTS</td><td>Hives</td></tr>
5          <tr><td>LUCYVANPELT</td><td>Irritation</td></tr>
6      </tbody>
7      </table>
8  </div>

```

[Top](#)

Template Commands

- [Apply \(%\)](#)
- [ParamApply \(%%\)](#)

[Top](#)

Apply (%)

While templates as described so far, provide for powerful formatting options, the true potential is unlocked when allowing them to **Apply** (with shorthand equivalent of **%**) other templates, either to the current context node, or against a different yet relative context. **Apply** always expects the **name** of a supporting template to be provided. Optionally, you can also provide an XPath to apply the template against a different node context. Otherwise, the template will be applied against the current node context.

Apply Example: Context Node

```

1  <Patient lastName="Brown" firstName="Charlie">

```

```

2      <Diagnoses>
3          <row DiagnosisNo="1001" description="Depression"/>
4      </Diagnoses>
5      <Allergies>
6          <row AllergyNo="1001" description="PEANUTS" reaction="Hives"/>
7          <row AllergyNo="1002" description="LUCYVANPELT" reaction="Irritation"/>
8      </Allergies>
9      <Medications/>
10 </Patient>

```

Apply Example: Template Sample

```

1 <template>
2     <div><h1>{{ % FullName }}</h1>
3
4     {{ Apply PatientDiagnoses Diagnoses }}
5
6     {{ Apply PatientAllergies Allergies }}
7
8     {{ Apply PatientMedications Medications }}
9
10    </div>
11 </template>

```

Apply Example: Supporting Templates Sample

```

1 <templates>
2
3     <t name="FullName">{{ = @lastName}}, {{ = @firstName }}</t>
4
5     <t name="PatientDiagnoses">
6         {{ if (count(row)!=0) }}
7             <table>
8                 <tbody>
9                     {{ % PatientDiagnosisRow row }}
10                </tbody>
11            </table>
12        {{ else }}
13            <span>No active diagnoses</span>
14        {{ /if }}
15    </t>
16
17    <t name="PatientDiagnosisRow">
18        <tr><td>{{ = @description }}</td></tr>
19    </t>
20
21    <t name="PatientAllergies">
22        {{ if (count(row)!=0) }}
23            <table>
24                <tbody>
25                    {{ % PatientAllergyRow row }}
26                </tbody>
27            </table>
28        {{ else }}
29            <span>No documented allergies</span>
30        {{ /if }}
31    </t>
32
33    <t name="PatientAllergyRow">
34        <tr><td>{{ = @description }}</td><td>{{ = @reaction }}</td></tr>
35    </t>
36
37    <t name="PatientMedications">

```

```

38     {{ if (count(row)!=0) }}
39         <table>
40         <tbody>
41             {{ % PatientMedicationRow row }}
42         </tbody>
43     </table>
44     {{ else }}
45         <span>No active medications</span>
46     {{ /if }}
47 </t>
48
49 <t name="PatientMedicationRow">
50     <tr><td>{{ = @description }}</td><td>{{ = @strength }}</td></tr>
51 </t>
52
53 </templates>

```

NOTE: The actual nodeName of the child nodes is irrelevant and ignored by the engine. We have used the child tag <t> for illustration purposes only.

Apply Example: Output

```

1  <div><h1>Brown, Charlie</h1>
2
3      <table>
4      <tbody>
5          <tr><td>Depression</td></tr>
6      </tbody>
7      </table>
8
9      <table>
10     <tbody>
11         <tr><td>PEANUTS</td><td>Hives</td></tr>
12         <tr><td>LUCYVANPELT</td><td>Irritation</td></tr>
13     </tbody>
14     </table>
15
16     <span>No active medications</span>
17
18 </div>

```

[Top](#)

ParamApply (%%)

In some cases, you will find a desire to pass parameters to supporting templates, which can be accomplished with the **ParamApply** command (with shorthand equivalent of **%%**). Unlike the **Apply** command, the **ParamApply** command requires an end command, being **EndParamApply** (or its shorthand options **/ParamApply** or **/%%**). Within the begin and end wrapper commands, you can specify either [Variable \(:=\)](#) or [MultiVar \(::=\)](#) commands, which will be scoped to within this template application. These variable declarations will act exactly like standard variables, however they will fall out of scope immediately after the template call.

ParamApply Example: Context Node

```

1  <Root>
2  <input click="this.focus();" desc="text field"/>

```



```

3   <emptyEl/>
4   <button click="alert(this.desc);" desc="stuff"/>
5 </Root>

```

ParamApply Example: Template Sample

```

1 <template>
2   {{ := onclick "'return void();'" }}{{ := description "'Howdy!'" }}
3   {{ apply BuildElement }}
4   {{ EACH * }}
5     {{ paramapply BuildElement . }}
6     {{ := element "name(.)" }}{{ := onclick @click }}{{ := description @desc }}
7     {{ /paramapply }}
8   {{ /EACH }}
9   {{ apply BuildElement }}
10 </template>

```

ParamApply Example: Supporting Templates Sample

```

1 <templates>
2   <t name="BuildElement">
3     <span onclick="{{ = $onclick }}" sourceEl="{{ = $element }}">{{ = $description }}</span>
4   </t>
5 </templates>

```

NOTE: The actual nodeName of the child nodes is irrelevant and ignored by the engine. We have used the child tag <t> for illustration purposes only.

ParamApply Example: Output

```

1 <span onclick="return void();" sourceEl="">Howdy!</span>
2 <span onclick="this.focus();" sourceEl="input">text field</span>
3 <span onclick="" sourceEl="emptyEl"></span>
4 <span onclick="alert(this.desc);" sourceEl="button">stuff</span>
5 <span onclick="return void();" sourceEl="">Howdy!</span>

```

[Top](#)

Variable Commands

- [Variable \(var, :=\)](#)
- [MultiVar \(mvar, ::=\) ... /MultiVar \(/mvar, /::=, endmultivar, endmvar\)](#)

[Top](#)

Variable (var, :=)

The **Variable** command, which has the shorthand equivalents of **var** and **:=** is used to store a textual value as a (global) variable that can be used later. This command takes (2) parameters: **name**, **xpath**. The first parameter

gives it a name to be used, and the second indicates what value should be stored.

A named variable can then be referenced at any point that a single xpath expression is expected, by simply providing the variable name prefixed by the variable delimiter (default being \$).

Variable Example: Context Node

```
1 <node>
2   <data d1="08/26/2010" />
3 </node>
```



Variable Example: Template Sample

```
1 <template>
2   {{ := myVar data/@d1 }}<span>From Xpath: {{ = data/@d1 }} From Variable: {{ = $myVar }}</span>
3 </template>
```



Variable Example: Output

```
1 <span>From Xpath: 08/26/2010 From Variable: 08/26/2010</span>
```



NOTE: As these are global variables, they can also be used in supporting templates.

Variable Example: Context Node

```
1 <node>
2   <data d1="08/26/2010" />
3 </node>
```



Variable Example: Template Sample

```
1 <template>
2   {{ := InputDate data/@d1 }}<span>Original: {{ = $InputDate }} EuroDate: {{ % EuroDate }}</span>
3 </template>
```



Variable Example: Supporting Templates Sample

```
1 <templates>
2
3   <template name="EuroDate">
4     {{ ~ "\b(?&lt;month&gt;\d{1,2})/(?&lt;day&gt;\d{1,2})/(?&lt;year&gt;\d{2,4})\b" "$ {day} - $ {month} - $ {year}" }}
5   </template>
6
7 </templates>
```



Variable Example: Output

```
1 <span>Original: 08/26/2010 EuroDate: 26-08-2010</span>
```



[Top](#)

MultiVar (mvar, ::=) ... EndMultiVar (endmvar, /MultiVar, /mvar, /::=)

The **MultiVar** command is used to start creating a variable out of multiple pieces, and is completed with an

EndMultiVar command. Any data that would have been resulted that is contained within a multivar start and end command will be stored as the contents of the variable. The start command has two shorthand equivalents: **mvar**, **::=**, and several shorthand versions of the end command: **endmvar**, **/MultiVar**, **/mvar**, **/::=**. This command takes a single parameters of **name**, as the contents will be the value. Once created, Multipart Variables act identically to standard variables.

MultiVar Example: Context Node

```
1 <node>
2   <data att1=""some"" att2=""more"" att3=""other"" att4=""weird"">stuff</data>
3 </node>
```

MultiVar Example: Template Sample

```
1 <template>
2   Multipart: {{ mvar totalstuff }}{{ each data/@* }}{{ = . }} {{ /each }}{{ = data }}{{ /mvar }}<spar
3 </template>
```

MultiVar Example: Output

```
1 Multipart: <span>some more other weird stuff</span>
```

[Top](#)

Manipulation Commands

- [Replace \(~\)](#)

[Top](#)

Replace (~)

The **Replace** command, which has a shorthand of **~** is used to perform a regular-expression based string replacement on either an Xpath or variable value. This command takes (3) parameters: **regExp**, **replaceText**, **input**. The first two of these are as defined by Microsoft .Net framework relating to the [RegEx Replace](#) method. (see also [.NET Framework Regular Expressions](#))

Replace Example: Context Node

```
1 <node>
2   <data d1="08/26/2010" />
3 </node>
```

Replace Example: Template Sample

```
1 <template>
2   <span>Original: {{ = data/@d1 }} Updated: {{ ~ "\b(?&lt;month&gt;\d{1,2})/(?&lt;day&gt;\d{1,2})/(?"
3 </template>
```

Replace Example: Output (both examples above provide identical output)

```
1 <span>Original: 08/26/2010 Updated: 26-08-2010</span>
```

[Top](#)

Miscellaneous Commands

- [Version \(ver\)](#)

[Top](#)

Version (ver)

The Version command is used to simply display the current version information of the **X2TL** engine.

Version Example: Context Node

```
1 | <dontCare />
```

[?](#)

Version Example: Template Sample

```
1 | <template>
2 |   <span>{{ ver }}</span>
3 | </template>
```

[?](#)

Version Example: Output

```
1 | <span>X2TL v1.1.0 rev.0</span>
```

[?](#)[Top](#)

See Also

[X2TL Default Supporting Templates](#)