

# Templates in the uiml.NET renderer

Jan Meskens

April 12, 2006

## Contents

<b>1</b>	<b>About</b>	<b>1</b>
<b>2</b>	<b>Implemented tags</b>	<b>2</b>
2.1	Behavior . . . . .	2
2.2	Constant . . . . .	3
2.3	Content . . . . .	3
2.4	DClass . . . . .	4
2.5	DComponent . . . . .	5
2.6	Logic . . . . .	6
2.7	Interface - Peers - Presentation . . . . .	6
2.8	Part . . . . .	7
2.9	Property . . . . .	7
2.10	Restructure . . . . .	9
2.11	Rule . . . . .	9
2.12	Script . . . . .	9
2.13	Structure . . . . .	10
2.14	Style . . . . .	11
<b>3</b>	<b>Comments</b>	<b>11</b>
3.1	References between elements . . . . .	11
3.1.1	Behavior-Structure references . . . . .	11
3.1.2	Style-Structure references . . . . .	12
3.1.3	Possible general solution . . . . .	13
3.2	DMethod's . . . . .	13
3.3	Attributes . . . . .	13
3.4	#PCDATA Children . . . . .	14
3.5	Finite number of children . . . . .	14
<b>4</b>	<b>To Do</b>	<b>14</b>
<b>5</b>	<b>Errors in the specification</b>	<b>14</b>

## 1 About

The purpose of this document is to give some information about the template support in the uiml.net renderer. An example of every supported tag is included.

During the implementation of the uiml.net template mechanism some comments about the specification were formed and appended to this document.

## 2 Implemented tags

This section is an overview of the different implemented tags - with tested examples - in the uiml.net renderer as described in the the uiml specification (listing 1). Notes about the different tags are included in the different sections.

Listing 1: the template specification

```
<!ELEMENT template (behavior | constant | content | d-class |
                    d-component | interface | logic | part | peers |
                    presentation | property | restructure | rule | script |
                    structure | style)>
<!ATTLIST template
    id NMTOKEN #IMPLIED>
```

### 2.1 Behavior

An example of the possibilities of the `behavior` tag in the `template` element is given in listing 2. The `behavior` templates are sensitive for the reference problem (explained in section 3.1.1).

Listing 2: the behavior element

```
<uiml>
  <interface>
    <structure>
      ...
    </structure>
    <style>
      ...
    </style>
    <behavior source="#behavior_template" how="cascade" />
  </interface>
  <peers>
    <presentation base=" ../swf-1.1.uiml" />
  </peers>
  <template id="behavior_template">
    <behavior>
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="copyleft" />
        </condition>
        <action>
          <property part-name="rightentry" name="text">
            <property part-name="leftentry" name="text" />
          </property>
        </action>
      </rule>
    </behavior>
  </template>
</uiml>
```

```

</rule>
<rule>
  <condition>
    <event class="ButtonPressed" part-name="copyright" />
  </condition>
  <action>
    <property part-name="leftentry" name="text">
      <property part-name="rightentry" name="text" />
    </property>
  </action>
</rule>
</behavior>
</template>
</uiml>

```

## 2.2 Constant

The `constant` template element is also present (see 3), although the current implementation doesn't support advanced `content` handling with the `reference` element. This element is tested by virtual tree inspection.

## 2.3 Content

The `content` element is not fully implemented by the uiml.net renderer at the moment. The `template` functionality is already present and tested by virtual part tree inspection. The `content template` is very functional to create eg. multilingual interfaces (listing 3).

Listing 3: the content element

```

<uiml>
  <interface>
    <structure>
      ...
    </structure>
    <style>
      ...
    </style>
    <content source="#dutch_language" how="replace" />
  </interface>
  <peers>
    ...
  </peers>
  <template id="content_template">
    <content>
      <constant id="bla" value="hihi" />
      <constant id="blabla" value="jajajajaaa" />
      <constant model="list" id="homp" source="#list_model"
        how="replace" />
    </content>
  </template>
</uiml>

```

```

</template>
<template id="list_model">
  <constant>
    <constant id="1" value="one" />
    <constant id="2" value="two" />
    <constant id="3" value="three" />
  </constant>
</template>
</uiml>

```

## 2.4 DClass

The use of DClass templates could be a solution for open issue 2.9 "Should UIML include inheritance within d-classes and templates". As proof of concept a swf-1.1 vocabulary is constructed and tested (listing 4).

Listing 4: the d-class element

```

<uiml>
<template id="baseProperties">
  <d-class>
    <d-property id="label" maps-type="setMethod"
      maps-to="Text">
      <d-param type="System.String" />
    </d-property>
    <d-property id="size" maps-type="setMethod"
      maps-to="Size">
      <d-param type="System.Drawing.Size" />
    </d-property>
    <d-property id="position" maps-type="setMethod"
      maps-to="Location">
      <d-param type="System.Drawing.Point" />
    </d-property>
    <d-property id="width" maps-type="setMethod"
      maps-to="Width">
      <d-param type="System.Int" />
    </d-property>
    <d-property id="height" maps-type="setMethod"
      maps-to="Height">
      <d-param type="System.Int" />
    </d-property>
    <d-property id="enabled" maps-type="setMethod"
      maps-to="Enabled">
      <d-param type="System.Boolean" />
    </d-property>
    <d-property id="visible" maps-type="setMethod"
      maps-to="Visible">
      <d-param type="System.Boolean" />
    </d-property>
    <d-property id="background" maps-type="setMethod"

```

```

    maps-to="BackColor">
    <d-param type="System.Drawing.Color" />
  </d-property>
  <d-property id="foreground" maps-type="setMethod"
    maps-to="ForeColor">
    <d-param type="System.Drawing.Color" />
  </d-property>
</d-class>
</template>
<presentation base="swf-1.1" id="SWF">
  <d-class id="Frame" used-in-tag="part"
    maps-type="class"
    maps-to="System.Windows.Forms.GroupBox"
    source="#baseProperties" how="cascade" />
  <d-class id="Container" used-in-tag="part"
    maps-type="class"
    maps-to="System.Windows.Forms.Panel"
    source="#baseProperties" how="cascade" />
  <d-class id="Button" used-in-tag="part"
    maps-type="class"
    maps-to="System.Windows.Forms.Button"
    source="#baseProperties" how="cascade">
    <d-property id="label" return-type="System.String"
      maps-type="getMethod" maps-to="Text" />
    <!-- events -->
    <d-property id="clicked" maps-type="event"
      maps-to="Click">
      <d-param
        type="System.Windows.Forms.Control.OnClick" />
    </d-property>
    <d-property id="entered" maps-type="event"
      maps-to="Entered">
      <d-param
        type="System.Windows.Forms.Control.OnEnter" />
    </d-property>
    ...
  </d-class>
  ...
</presentation>
</uiml>

```

## 2.5 DComponent

In listing 5 an example of the d-component tag is given. This element is fully supported by the uiml.net renderer.

Listing 5: the dcomponent and logic tag

```

<uiml>
  ...

```

```

<template id="logic-template">
  <logic source="#date-template" how="cascade">
    <d-component id="String" maps-to="System.String"
      source="#concatenate" how="cascade">
      <d-method id="compare" returns-value="int"
        maps-to="Compare">
        <d-param id="str0" type="System.String"/>
        <d-param id="str1" type="System.String"/>
      </d-method>
    </d-component>
  </logic>
</template>
<template id="concatenate">
  <d-component>
    <d-method id="concatenate"
      returns-value="string" maps-to="Concat">
      <d-param id="str0" type="System.String"/>
      <d-param id="str1" type="System.String"/>
    </d-method>
  </d-component>
</template>
<template id="date-template">
  <logic>
    <d-component id="Date" maps-to="System.DateTime">
      <d-method id="now" returns-value="DateTime"
        maps-to="Now"/>
      <d-method id="today" returns-value="DateTime"
        maps-to="Today"/>
      <d-method id="compare" returns-value="int"
        maps-to="Compare">
        <d-param id="date1" type="System.DateTime"/>
        <d-param id="date2" type="System.DateTime"/>
      </d-method>
    </d-component>
  </logic>
</template>
<logic source="#logic-template" how="cascade">
  ...
</logic>
</uiml>

```

## 2.6 Logic

The `logic` tag is also shown in listing 5.

## 2.7 Interface - Peers - Presentation

Multiple `Structures`, `Styles` and `Presentations` aren't supported in the current `uiml.net` implementation, some work on these points should be done before

the template mechanism could work.

## 2.8 Part

The `part` template is fully implemented in the uiml.net renderer. An example is given in listing 6. The `part` tag could also have references outside the `template` : this comment is discribed in section 3.1.2 and 3.1.1.

Listing 6: the part tag

```
<uiml>
  <interface>
    <structure>
      <part id="Body" class="Container">
        <part id="PizzaForm" class="Frame">
          <part id="Toppings" class="Frame">
            <part id="Ansjovis" class="CheckBox" />
            <part id="Mozarella" class="CheckBox" />
            <part id="Olives" class="CheckBox" />
          </part>
          <part id="Size" class="Frame" source="#tester"
            how="replace" />
          <part id="Order" class="Button" />
          <part id="Cancel" class="Button" />
        </part>
      </part>
    </structure>
    <style>
      ...
    </style>
  </interface>
  ...
  <template id="tester" >
    <part>
      <part id="Small" class="RadioButton" />
      <part id="Medium" class="RadioButton" />
      <part id="Large" class="RadioButton" />
    </part>
  </template>
</uiml>
```

## 2.9 Property

In the current uiml.net renderer, a `property` element could contain only one element of the `#PCDATA`, `constant`, `property`, `reference` or `call` type. Because a `property` can contain only one child not all the template strategies are possible here :

**Replace :** This strategy is straight forward and easy to use.

**Union and Cascade :** Only possible when the placeholder does not contain a child (because 1 child is the maximum).

An example of the `property` `template` is given in 7.

Listing 7: the `property` tag

```
<uiml>
<template id="artist_name">
  <property>Sex Pistols</property>
</template>
<interface>
  <structure>
    <part id="fr" class="Frame">
      <part id="left" class="Container">
        <part id="artist1" class="Label">
          <style>
            <property name="text" source="#artist_name"
              how="replace"/>
          </style>
        </part>
        <part id="articles" class="List"/>
      </part>
      ...
    </part>
  </structure>
  <style>
    <property part-name="l_title" name="text"
      source="#artist_name" how="replace"/>
    <property part-name="title" name="text"
      source="#artist_name" how="replace"/>
    <property part-name="update" name="label"
      source="#artist_name" how="replace"/>
    <property part-name="articles" name="content"
      source="#articles_list" how="replace"/>
  </style>
</interface>
<peers>
  ...
</peers>
<template id="articles_list">
  <property>
    <constant model="list">
      <constant value="Blog_title_1"/>
      <constant value="Entry_2"/>
      <constant value="Another_Title"/>
    </constant>
  </property>
</template>
</uiml>
```

## 2.10 Restructure

Is currently not supported in the uiml.net renderer.

## 2.11 Rule

The `rule` tag can contain 1 `condition` and `action` element or no elements. When it contains no elements cascade and union is possible; otherwise only replace is possible (the same system as explained in 2.9). An example is given in listing 8.

Listing 8: the rule tag

```
<uiml>
<structure>
<part class="Button" id="b8" />
</structure>
<behavior>
  <rule source="#b8_buttonPressed" how="replace" />
</behavior>
<peers>
  ...
</peers>
<template id="b8_buttonPressed">
<rule>
  <condition>
    <event part-name="b8" class="ButtonPressed" />
  </condition>
  <action>
    <property part-name="output" name="text">
      <call name="String.concatenate">
        <param>
          <property part-name="output" name="text" />
        </param>
        ...
      </call>
    </property>
  </action>
</rule>
</template>
</uiml>
```

## 2.12 Script

In the `script` element only `#PCDATA` is allowed. In the spec there is no mechanism described to merge `#PCDATA`. A comment about this is made in section 3.4. Listing 9 illustrates the script tag.

Listing 9: the scripted buttons example tag

```
<uiml>
...
<structure>
```

```

<part id="NemerleButton" class="Button">
  <style>
    <property name="label">Nemerle
  </property>
  </style>
</part>
</structure>
...
<behavior>
<rule>
  <condition>
    <event class="ButtonPressed"
      part-name="NemerleButton" />
  </condition>
  <action>
    <call>
      <script type="Nemerle"
        source="#nemerle" how="replace" />
    </call>
  </action>
</rule>
</behavior>
...
<template id="nemerle">
  <script>
    System.Console.WriteLine ("Nemerle says:
    .....\"Hello world!\");
  </script>
</template>
</uiml>

```

## 2.13 Structure

The **structure** element can contain at least one **part** tag in the uiml.net implementation. This constraint results in less template solution strategies. The cascade and union method is only possible with an empty structure element (this problem is as described in comment 3.5 and 2.9). An example of the **structure** element is given in listing 10. The reference problem, as described in section 3.1.2, occurs also here.

Listing 10: A structure example

```

<uiml>
  <interface>
    <structure source="#myStructure" how="cascade" />
    <peers>
      ...
    </peers>
    <template id="myStructure">
    <structure>

```

```

<part class="Frame" id="Frame">
  <part class="Entry" id="leftentry"/>
  <part class="Button" id="copyleft"/>
  <part class="Button" id="copyright"/>
  <part class="Entry" id="rightentry"/>
</part>
</structure>
</template>
</uiml>

```

## 2.14 Style

The `style` tag is demonstrated in listing 11. References between the `style` - and `structure` block occur frequently (see section 3.1.2).

Listing 11: The style tag

```

<uiml>
<template id="property">
  <style>
    <property part-name="b0" name="label">0</property>
    <property part-name="b1" name="label">1</property>
    <property part-name="b2" name="label">2</property>
    <property part-name="b3" name="label">3</property>
    <property part-name="b4" name="label">4</property>
    <property part-name="b5" name="label">5</property>
    <property part-name="b6" name="label">6</property>
  </style>
  <interface>
    <structure>...</structure>
    <style source="#property" how="cascade">
      ...
    </style>
  </interface>
</template>

```

## 3 Comments

### 3.1 References between elements

A general drawback of the templates is the vanishment of references between different parts of the uiml document. The major drawback of this problem is the lose of generality in the templates. In the next sections there are a few examples of the "reference problem".

#### 3.1.1 Behavior-Structure references

In listing 2 there are references from the `behavior` template to the `structure` part Eg. `<event class="ButtonPressed" part-name="copyleft"/>`. This

example references to somewhere in the **structure part-tree** (more specific to the **part** element with the **copyleft** identifier). So, the template designer must know the part-names when he is developing the template. The strong coupling between the **structure** and the **template-behavior** element results in less generality.

### 3.1.2 Style-Structure references

In listing 12 are references from the **structure** part to the **style** block eg : the property *'checked'* to the *'small'* part. To solve this there is already an *'inline'* mechanism in the uiml specification, it's shown in listing 13.

Listing 12: Structure and Style references

```

<uiml>
<interface>
<structure>
<part id="Body" class="Container">
  <part id="PizzaForm" class="Frame">
    <part id="Toppings" class="Frame">
      <part id="Ansjovis" class="CheckBox" />
      <part id="Mozarella" class="CheckBox" />
      <part id="Olives" class="CheckBox" />
    </part>
    <part id="Size" class="Frame" source="#tester"
      how="replace" />
    <part id="Order" class="Button" />
    <part id="Cancel" class="Button" />
  </part>
</part>
</structure>
<style>
  <property part-name="Body" name="size">
    400,400</property>

  <property part-name="PizzaForm" name="position">
    5,5</property>
  <property part-name="PizzaForm" name="size">
    310,200</property>
  <property part-name="PizzaForm" name="label">
    Pizza Form</property>
  ...
  <property part-name="Small" name="checked">
    false</property>
  <property part-name="Small" name="position">
    20,20</property>
  ...
  <property part-name="Medium" name="checked">
    true</property>
  ...
  <property part-name="Large" name="checked">

```

```

    false</property>
    ...
</style>
</interface>
<peers>
    ...
</peers>
<template id="tester" >
  <part>
    <part id="Small" class="RadioButton" />
    <part id="Medium" class="RadioButton" />
    <part id="Large" class="RadioButton" />
  </part>
</template>
</uiml>

```

Listing 13: Inline style definition

```

<template id="bla">
  <part class="Button">
    <style>
      <property name="label">Click me!</property>
    </style>
  </part>
</template>

```

### 3.1.3 Possible general solution

A possible solution to fix this is a manner to pass parameters to the template element. This system could be compared with functions in a programming language. The use of parameters introduces the scope terminology : inside the template is the local scope, passing elements to the local scope could be done with parameters.

## 3.2 DMethod's

The `dmethod` element contains a source and how attribute but cannot be sourced with a `template`, this is maybe an inconsistency in the current specification.

## 3.3 Attributes

The specification isn't clear about what to do with the attributes of the template top element and the placeholder (see listing 14, which of the two identifiers shall be chosen ?).

Listing 14: The identifier problem

```

<part id="original_identifier" source="#test"
  how="cascade" />
  ...

```

```
<template id="test">
  <part id="identifier" .../>
</template>
```

### 3.4 #PCDATA Children

How is it possible to resolve templates when the placeholder contains *#PCDATA* children? An example of such a situation is the `script` tag. The tags which can be inside a template and can contain *#PCDATA* children are :

- `script` : Only the replace method is supported; a warning is produced when the cascade or union strategy is used.
- `property` : The `property` tag can contain also other elements. Because a `property` tag can contain only 1 child (in the uiml.net implementation), no special union or cascade is needed (see section 2.9).

### 3.5 Finite number of children

When a template element can contain a finite number of children, not all strategies are always possible. This problem is explained in the `rule` (2.11) and `property` (2.9) sections.

## 4 To Do

- A cycle detecting algorithm
- Multiple nesting of the rule and behavior tag
- The correct template strategies
  - How to build correct names with the union and cascade strategy ? (not very clear in the current specification)
  - Add only unique children with the cascade strategy.

## 5 Errors in the specification

page 47 : At the top of the page, the `constant` tag is closed with the `property` tag.