Rational Publishing Engine Template Definition

Table of Contents

1 An introduction to Rational Publishing Engine Templates

A Rational Publishing Engine template describes *what* to render and *how* to render it. The templates are built by Rational Publishing Engine Document Studio which uses information and knowledge stored in a number of configuration files:

- ***TemplateXMLSchema.xsd*** – defines the schema of the template. Rational Publishing Engine will reject any template that does not perfectly match the schema. This file is bundled in the RRDG Core JAR file.
- **RPEElementsDefinitions.xml** –defines the formatting properties available for each Rational Publishing Engine Template element and where applicable, the domains of the values accepted for these properties. This file is located in **%RPE_HOME%/config.**
- ***RPEElementsRules.xml*** – defines the nesting rules of a template, which element can include what other elements and so on. This file is located in **%RPE_HOME%/config.**

1.1 Template versions

Each new Rational Publishing Engine version makes changes to the properties supported for each element and in some cases adds new elements. Rules might also change between Rational Publishing Engine versions. Whenever possible these changes are implemented in a backwards compatible manner. When that is not possible a new template language version is introduced. The template versions are integers and each new version increments the previous version by 1.

All Rational Publishing Engine versions are able to load templates with equal or lower template language versions but they cannot load templates with higher language version.

To date there are 2 template versions in circulation:

- version 1 – Rational Publishing Engine 1.1.2.1 and older
- version 2 – Rational Publishing Engine 1.2 and newer

1.2 Building a template

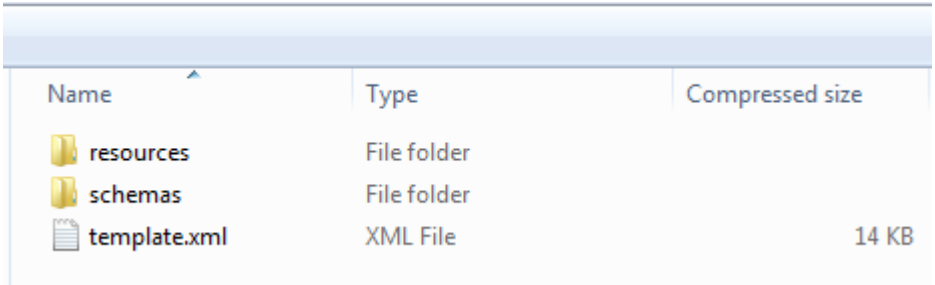
You must use Rational Publishing Engine Document Studio to build a Rational Publishing Engine template. At this time there is no Java API that can be used for this purpose. You could build a template manually or programmatically using the information in this document in which case you must use extreme caution when doing it.

2 Template Package

The Rational Publishing Engine templates are packaged as **zip** archive files. A Rational Publishing Engine zip contains the template itself and supporting resources (schemas, images, etc). Opening a template reveals the following structure:

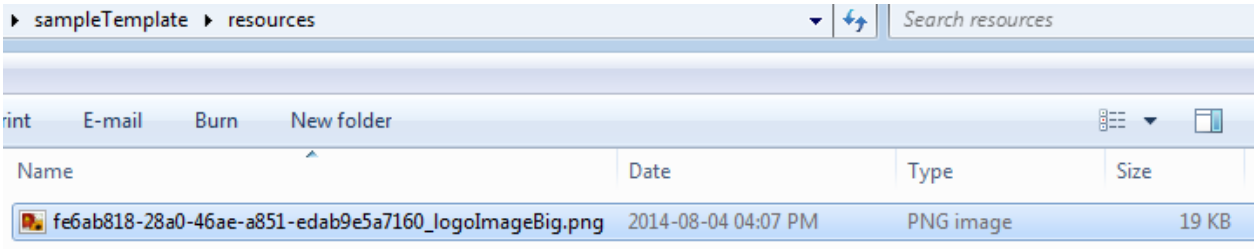
- *resources*
- *schemas*

- template.xml



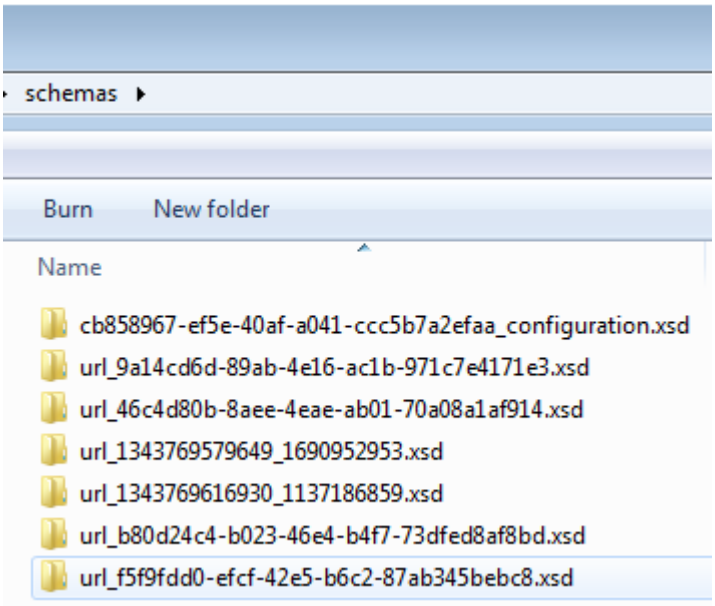
2.1 Resources

This folder contains images used in the template. The images are copied as is, but they are renamed to ensure there are no naming collisions in the template, including dynamically referenced templates. The algorithm used by Rational Publishing Engine is to add a GUID prefix to the name of the file.

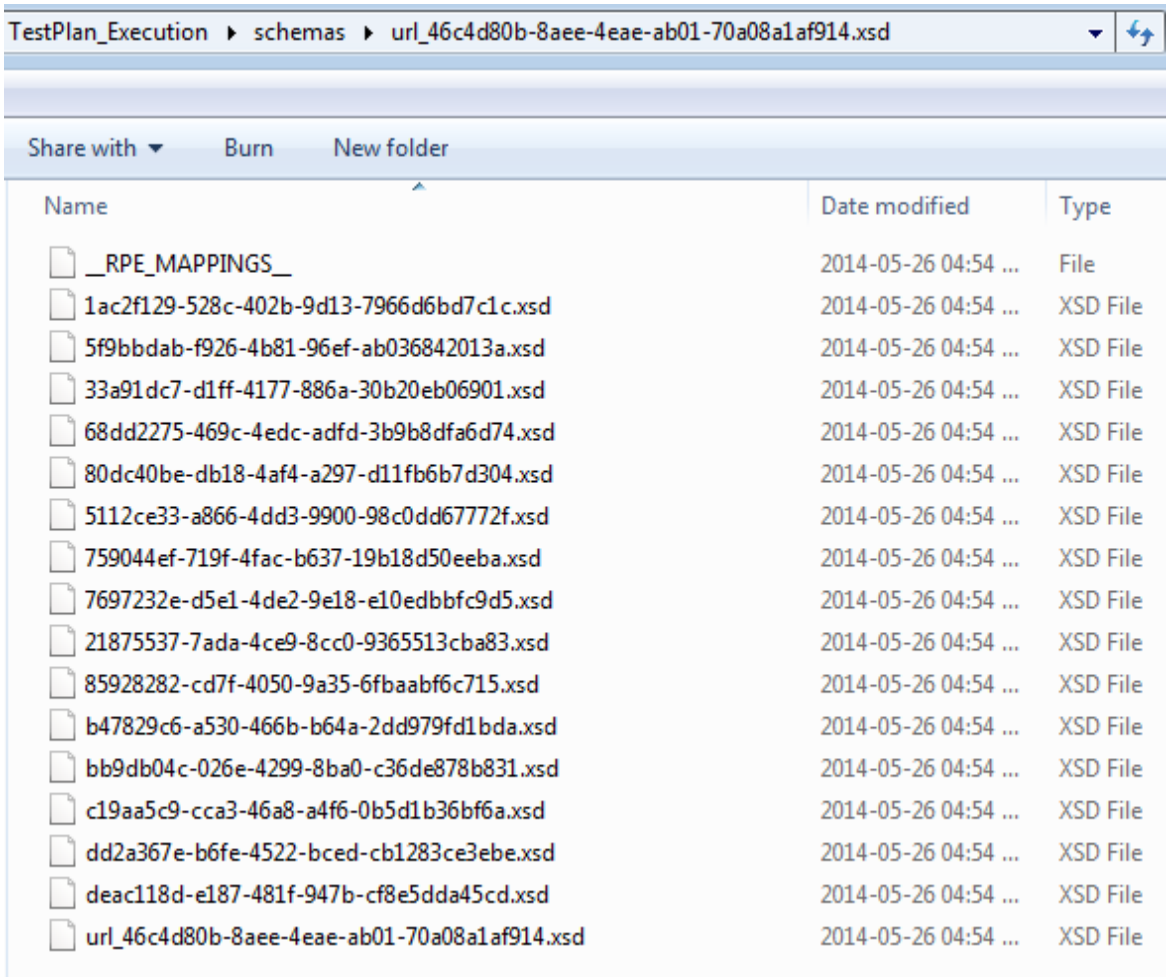


2.2 Schemas

This folder contains the schemas of all the data sources used in the template. Each data source is contained by a subfolder typically using GUIDs for names. For each data source a separate folder exists even if the files are identical. Rational Publishing Engine prevents naming collisions by using GUID to name the folders.



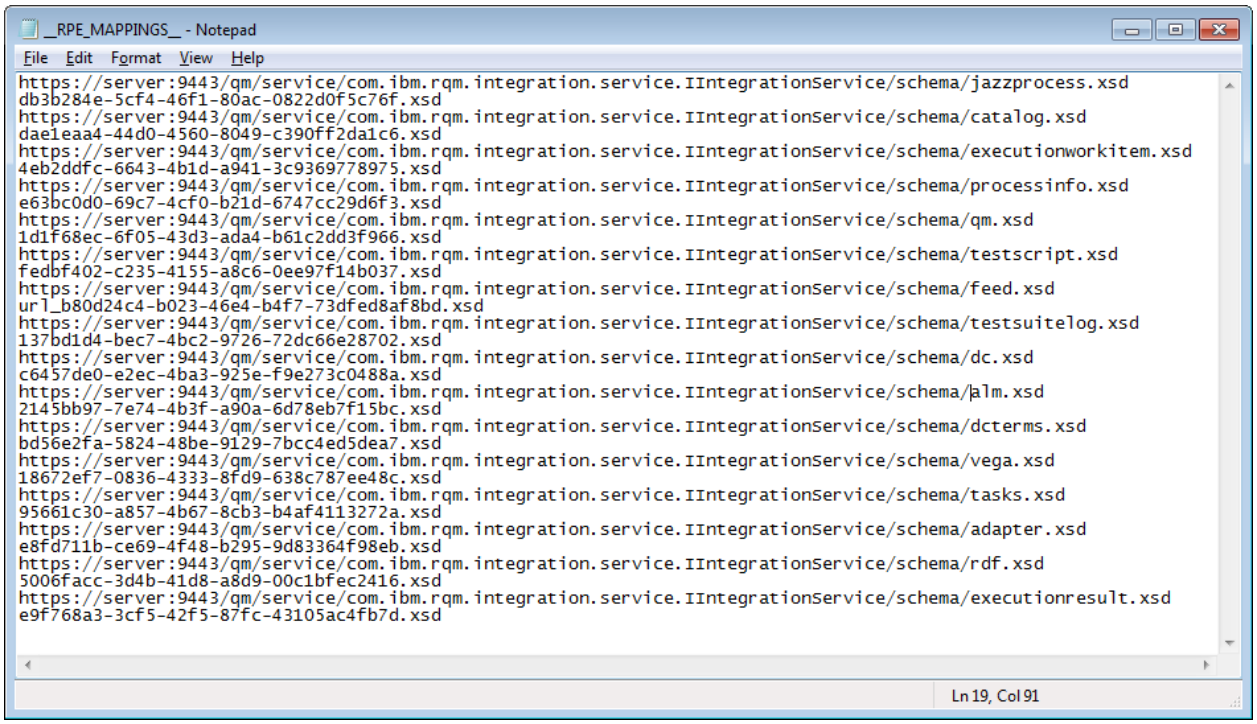
Most real world schemas are composed of multiple XSD files, usually by each file defining a different namespace. To speed up the template load process and make templates portable, Rational Publishing Engine downloads all the components of the schema and puts them in this folder.



The XSD file refers to the other XSD files using a “live” URL like https://server:9443/qm/service/com.ibm.rqm.integration.service.IIntegrationService/schema/qm.xsd and these live URLs must be mapped to the files cached in the schemas folder. This mapping is done in the __RPE_MAPPINGS__ file. The file structure is:

- Line 1: Source URL
- Line 2: Local file path
- Line 3: Source URL
- Line 4: Local file path
- …

Example:



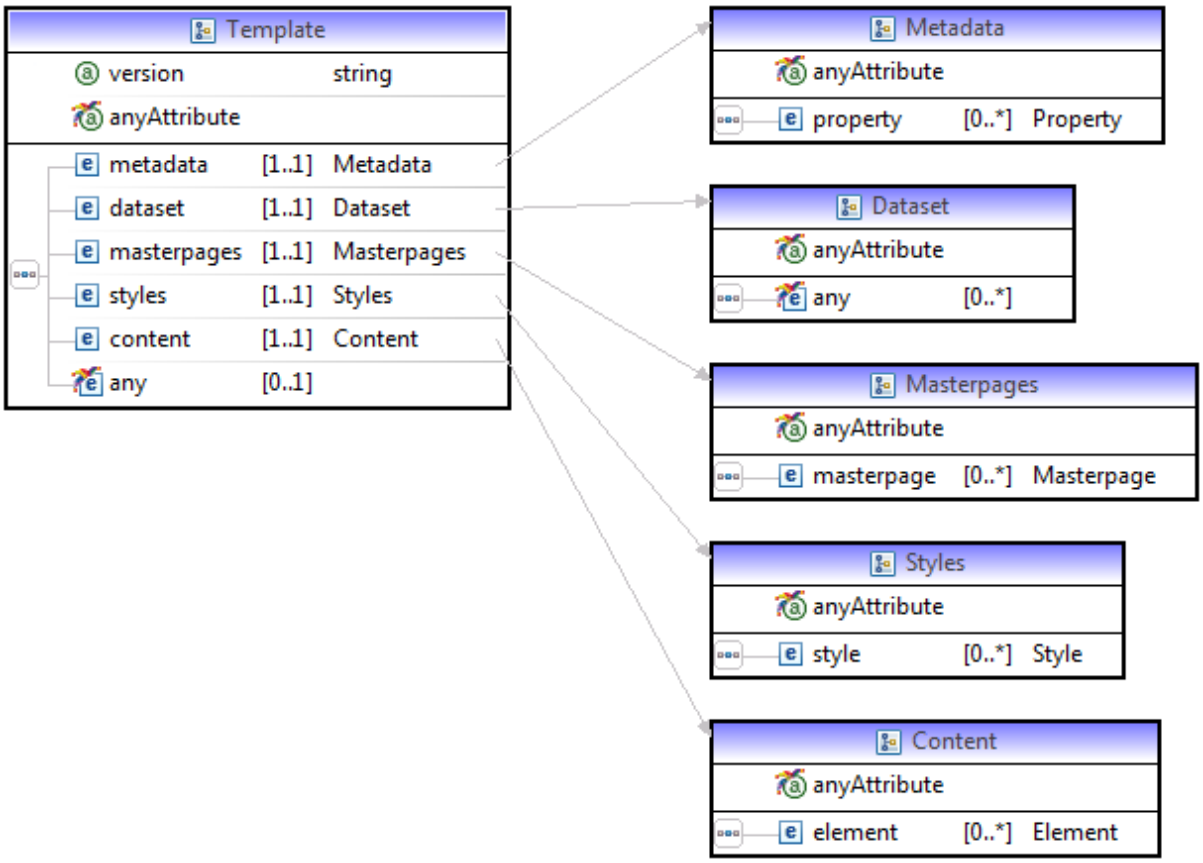
2.3 template.xml

This is the actual template content, referred to hereafter as the 'template'. The template defines the data it uses, the structure of the output and the detailed formatting for each element.

- The data can be either static or dynamic.
- Static information could include any hard coded text (for example standard legal disclaimers).
- Dynamic information is retrieved from data sources. A data source defines the structure of the data and not a specific instance of the data.
- The formatting consists of a set of properties that define the various aspects of the information in the output document.

3 Template Content

This section describes the semantics of the most important constructs in the template.xml file. For a formal description of the rules and the syntax of this file please consult the attached XSD file.



The high level structure of a template is:

1. metadata
2. dataset
3. master pages
4. styles
5. content

```
− <template>
  + <metadata></metadata>
  + <dataset></dataset>
  + <masterpages></masterpages>
  + <styles></styles>
  + <content></content>
  </template>
```

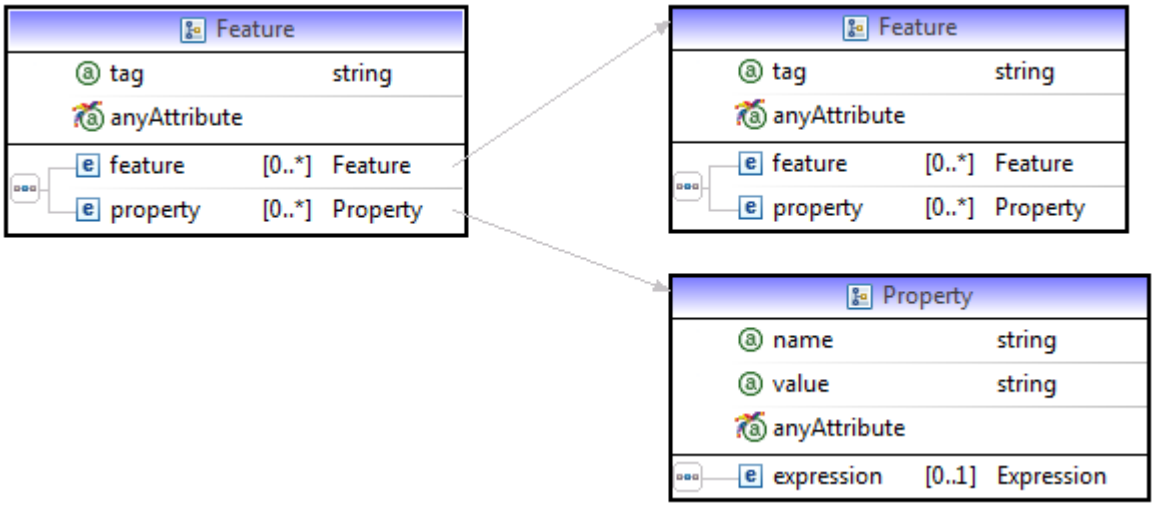
3.1 Basic structures

Most of the constructs used in the template language are described using property elements which in essence are name-value pairs. These properties are organized in features which are mere containers.

```
- <element id="4" tag="text" recursive="0" recursive_segments="1">
    <assignments/>
    <expression tag="content" type="data" context="$1">FullTag</expression>
  - <feature tag="format">
    - <feature tag="common">
        <property name="style name" value=""/>
        <property name="masterpage" value=""/>
        <property name="force page change" value=""/>
        <property name="heading level offset" value=""/>
        <property name="target region" value=""/>
      </feature>
```

3.1.1 Feature

Containers used to group properties. They can contain other features.



Attribute	Description	Mandatory
tag	the name of the feature.	yes

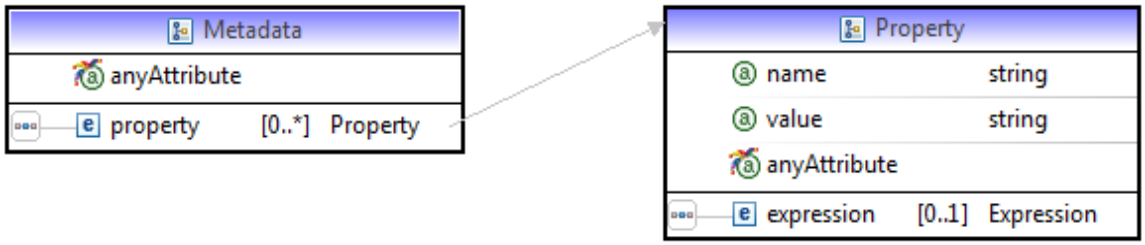
3.1.2 Property

A name-value pair. The value can be static, defined in the “value” attribute or dynamic, calculated at runtime.

Attribute	Description	Mandatory
name	the name of the property	yes
value	The static value of the property. Is not present if the property has a dynamic value defined by an expression. See the Expressions section.	no

3.2 Metadata

A set of properties for the template itself



Unlike formatting properties, the properties in the metadata section are fixed and have special meaning. However the schema does not enforce this explicitly so any number of properties can be added. But they will be ignored by Rational Publishing Engine and lost if Rational Publishing Engine is used to save the template.

```
−<metadata>
    <property name="version" value="2"/>
    <property name="name" value=""/>
  −<property name="description">
      <expression type="constant"/>
    </property>
    <property name="Query Separator" value="/"/>
  </metadata>
```

Property	Description	Mandatory
version	The template language version. Accepted values: <empty>, 1 , 2 If this property is missing the value is assumed to be 1	Yes
name	An optional name given to the template by the template designer for documenting the intended usage of this template. This name is displayed in various places by the Rational Publishing Engine UIs	No
description	An optional name given to the template by the template designer for documenting the intended usage of this template. This name is displayed in various places by the Rational Publishing Engine UIs	No
Query Separator	The separation used by Rational Publishing Engine in the XPath queries. This value is "/" Rational Publishing Engine 1.1.1.1 and older were using "." (dot) as the separator which was problematic since XML element names are allowed to contain ".".	Yes

3.3 Dataset

Defines the data sources and variables of a template.

```
- <dataset>
  - <sources>
    - <source>
      - <feature tag="definition">
          <property name="name" value="XML"/>
          <property name="schema" value="XML_schema"/>
        </feature>
      </source>
    </sources>
  - <schemas>
    - <schema>
      - <feature tag="definition">
          <property name="id" value="XML_schema"/>
          <property name="type" value="Generic XML"/>
          <property name="URI" value="schemas/url_3903eeae-da8b-483d-a0
        </feature>
      </schema>
    </schemas>
  - <variables>
    - <variable>
      - <feature tag="definition">
          <property name="name" value="title"/>
        - <property name="description">
            <expression type="constant"/>
          </property>
          <property name="type" value="user"/>
          <property name="access" value="external"/>
          <property name="default" value=""/>
          <property name="central variable" value=""/>
        </feature>
      </variable>
    </variables>
```

3.3.1 Sources

This element is used to group *source* elements.

3.3.2 Source

This element defines a data source. In the UI a single entity is visible for a data source but in the template xml, 2 entities map to it: a data source and a schema. For each data source there is one and only one schema and a schema cannot exist without a data source using it.

NOTE: the 2 entities exist for historical reasons and it's possible they will be merged in future versions.

```
- <source>
  - <feature tag="definition">
      <property name="name" value="XML"/>
      <property name="schema" value="XML_schema"/>
    </feature>
  </source>
</sources>
```

Properties:

Property	Description	Mandatory
name	The unique identifier of the data source. This identifier is used everywhere the data source needs to be referred. *Value must be unique in the set of source ids.*	Yes
schema	The ID of the data source's schema. MUST match the ID of one of the schema elements.	Yes

3.3.3 Schemas

This element is used to group *schema* elements.

3.3.4 Schema

This element defines a data source schema. See the discussion on Source.

```
- <schema>
  - <feature tag="definition">
      <property name="id" value="XML_schema"/>
      <property name="type" value="Generic XML"/>
      <property name="URI" value="schemas/url_3903eeae-da8b-483d-a(
    </feature>
  </schema>
```

Properties:

Property	Description	Mandatory
id	The identifier of the schema. This identifier is used by the data source definitions. *Value must be unique in the set of schemas ids.*	yes
type	The type of the data source. The valid values for this property come from the list of data sources known to Rational Publishing Engine. At the time of writing this document the list consists of: - Generic XML - DOORS - REST - REST v2	yes
URI	The relative path of the root XSD file for the schema. This file is stored in the template archive in the *schemas* folder.	yes

3.3.5 Variables

This is a *mandatory* element; contains zero or more *variable* elements.

```
- <variable>
  - <feature tag="definition">
      <property name="name" value="title"/>
    - <property name="description">
        <expression type="constant"/>
      </property>
      <property name="type" value="user"/>
      <property name="access" value="external"/>
      <property name="default" value=""/>
      <property name="central variable" value=""/>
    </feature>
  </variable>
```

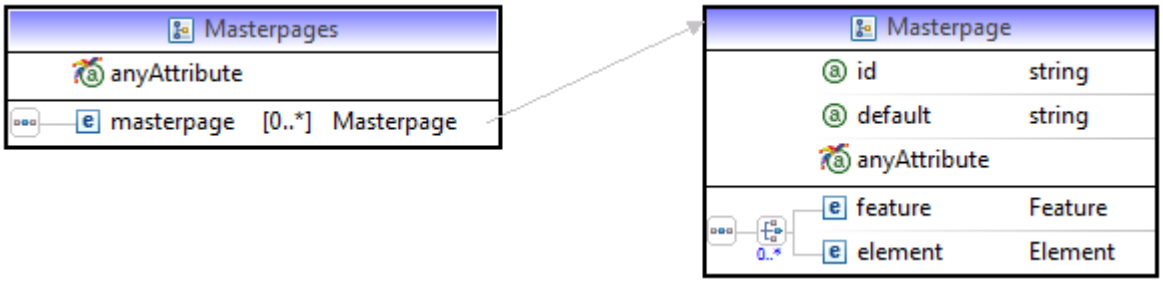
3.3.6 Variable

Each variable element has exactly one “definition” feature that contains the following properties:

Property	Description	Mandatory
type	The type of the variable. The only accepted variable at the time of writing this document is *user*.	yes
name	The name of the variable. Constraint: The *name must be unique in the set of variables names.*	yes
default	A default value that will be used if the user doesn’t provide on in the Document Specification file.	no
type	Reserved. Must be user	no
access	Defines if the variable is visible to the end user or not. Values: internal, external	no
description	An optional text provided by the template designer for documenting the intended usage of this variable	no
central variable	The URL of the central variable to which this template variable is connected	no

3.4 Master pages

Rational Publishing Engine uses the same concept of master page as the one found in Adobe Frame maker or Eclipse BIRT. A *master page* defines the layout of the output page (margins, borders, orientation) and its header and footer.



Example:

```
- <masterpages>
  - <masterpage>
    - <feature tag="definition">
        <property name="name" value="MP_Landscape"/>
        <property name="default" value="false"/>
      - <property name="description">
          <expression type="constant"/>
        </property>
      </feature>
    - <element id="6" tag="content">
      + <element id="6" tag="header" recursive="0" recursive_segments="1"></element>
      + <element id="7" tag="footer" recursive="0" recursive_segments="1"></element>
      </element>
    + <feature tag="format"></feature>
    </masterpage>
  </masterpages>
```

3.4.1 Definition

A master page is defined by a single *feature* element with the tag “definition” and a single content element which in turn contains exactly 1 header and 1 footer element.

Property	Description	Mandatory
name	The unique name of the masterpage. This identifier is used everywhere the masterpage needs to be referred. *Value must be unique in the set masterpages.*	yes
default	If set to true this masterpage will be used everywhere a masterpage is not explicitly specified. *Only one masterpage can be set default*	no
description	An optional text provided by the template designer for documenting the intended usage of this master page	

3.4.2 Content

One *header* element and one *footer* element.

3.4.3 Header

Same structure as the template's content element. The header cannot contain data source elements. See RPEElementsRules.xml for details.

3.4.4 Footer

Same structure as the template's content element. The footer cannot contain data source elements. See RPEElementsRules.xml for details.

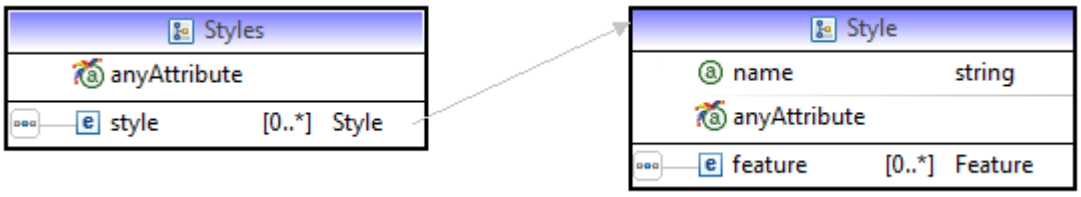
3.4.5 Format

The page's formatting properties. Only the specific section is described here, for the other properties that apply to master pages see RPEElementsDefinitions.xml.

Property	Description	Mandatory
page orientation	The page orientation. Possible values: *landscape*, *portrait*	no
page vertical alignment	The alignment of content in the master page. Top, center, bottom	no
same as previous	If set to true this master page copies the formatting and content of the previously used master page, if any. <empty>, True, False	no

3.5 Styles

A style defines how a given element in the template will be rendered in the output document. Each style is defined in a **Style** element and the Style elements are grouped in a **Styles** element.



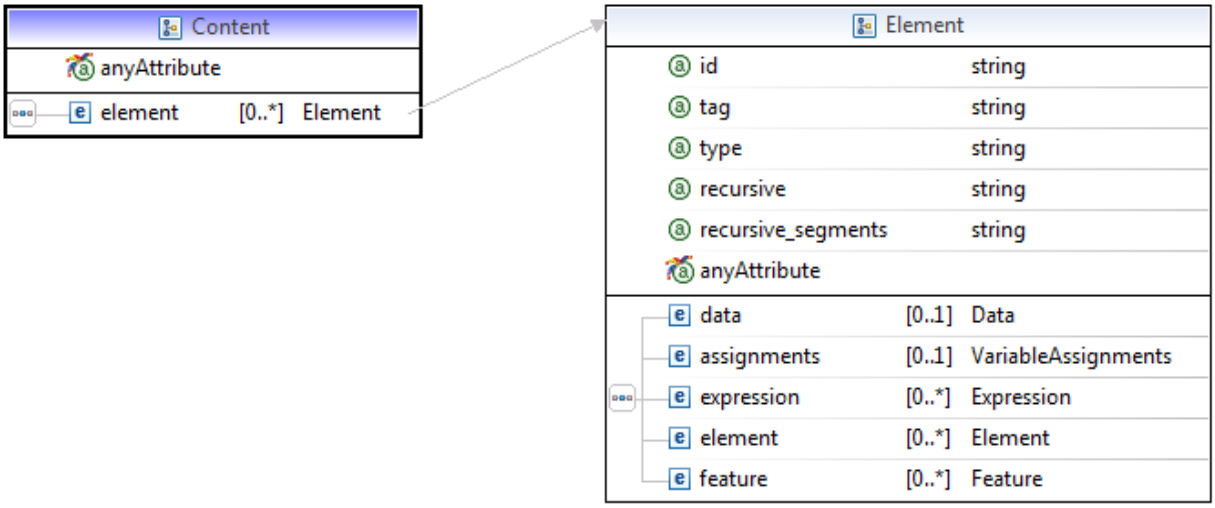
The name of the style is a property of the "definition" feature while the actual formatting details are stored in the "format" feature.

```
- <styles>
  - <style>
    - <feature tag="definition">
        <property name="name" value="Title"/>
      </feature>
    - <feature tag="format">
      - <feature tag="text">
        - <feature tag="font">
            <property name="bold" value="true"/>
            <property name="font size" value="18"/>
          </feature>
        </feature>
      </feature>
    </style>
  + <style></style>
  + <style></style>
  + <style></style>
  + <style></style>
  </styles>
```

Any formatting property of the Rational Publishing Engine Template Elements can be used in styles. For a complete list please consult RPEElementsDefinitions.xml

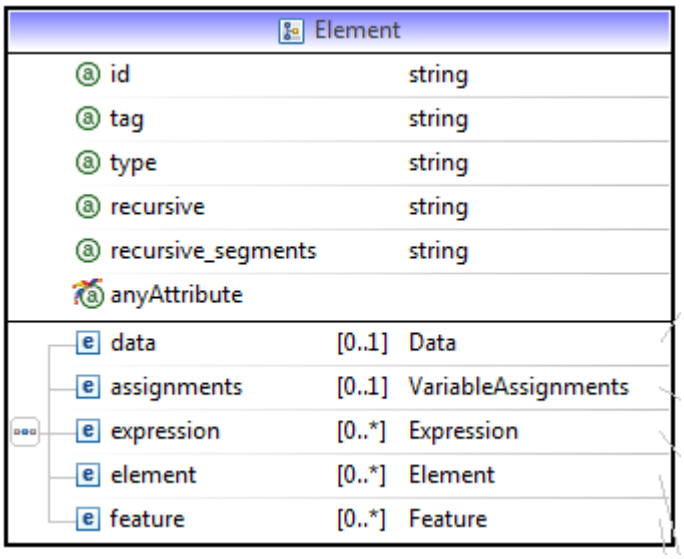
3.6 Content

This section drives the document generation process by defining the content of the document. The content is a set of zero or more *element*s.



3.6.1 Element

This is the building block of a template. An element has only one property, the "*tag*", which defines the element's name.



An element has:

- [0..1] data elements
- [0..1] assignments element
- [0..N] expressions used to define its content and more
- [0..N] child elements
- [0..1] features defining the elements formatting

The list of template elements is well defined. At the time of the writing the following elements are supported by Rational Publishing Engine.

Element	**Version Added**	**Container**	**Accepts queries**	**Description**
table	1	Yes (rows)	Yes	
list	1	Yes (rows)	Yes	
List-detail	1	Yes	Yes	
paragraph	1	Yes (any)	Yes	
container	1	Yes (any)	Yes	Container element with no formatting of its own. A container is used to group other elements.

Element	Version Added	Container	Accepts queries	Description
image	1	No	Yes	
text	1	No	Yes	
Styled text	1	No	Yes	A text element which supports a very narrow set of formatting options: bold, italic, underline, colors.
hyperlink	1	No	Yes	
row	1	Yes (cell)	Yes	
cell	1	Yes (any)	Yes	
region	1	No	no	An element used to gather data.
bookmark	1	No	No	
table of contents	1	No	No	
comment	1	No	No	
footnote	1	No	No	
Data Source Configuration	1	No	No	
Page Number	1	No	No	
Total Pages Number	1	No	No	
Page Break	1	No	No	
Section Break	1	No	No	
Include File	1	No	No	
Table Caption	1	No	No	
Figure Caption	1	No	No	

Element	Version Added	Container	Accepts queries	Description
Java Script	2	No	No	
Document Property	2	No	No	
Template Comment	2	No	No	A design time only artifact which is not rendered in the output.
Iteration	2	Yes	No	
Included Template	2			

Each element has associated **data** and **formatting** which are described in the next sections.

3.7 Element Formatting

All the formatting properties are defined as property elements grouped in features. The simplest way to use properties is to provide static text in the "value" attribute of the property element

Example 2: formatting property with constant expression

```
– <feature tag="text">
  – <feature tag="font">
      <property name="bold" value="true"/>
      <property name="font size" value="18"/>
    </feature>
  </feature>
```

3.7.1 Conditional formatting

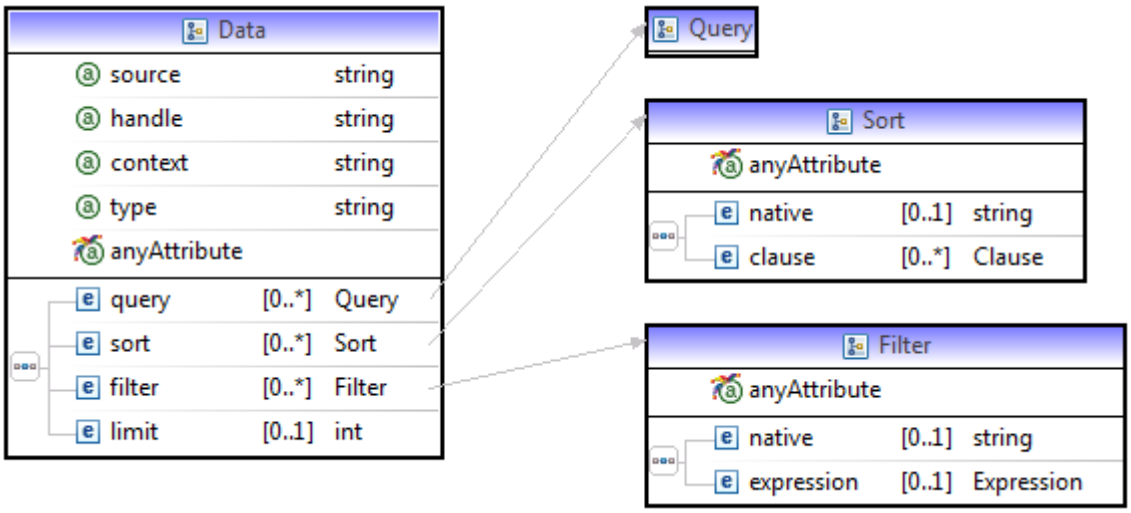
Formatting properties accept dynamic evaluation as well if the value is provided as a child expression of the property element. In this case the "value" attribute must not be present in the "property" element.

Example 1: formatting property with script expression

```
- <feature tag="font">
  - <property name="font color">
    - <expression type="script">
      - <code>
          if ( Priority == "high") { "FF0000"; } else { ""; }
        </code>
        <inXHTML>false</inXHTML>
        <outXHTML>false</outXHTML>
      - <expression_variables>
          <expression_variable type="data" context="$1">Priority</expression_variable>
        </expression_variables>
      </expression>
    </property>
    <property name="font size" value=""/>
    <property name="font family" value=""/>
    <property name="bold" value=""/>
    <property name="italic" value=""/>
    <property name="direction" value=""/>
  </feature>
```

3.8 Element Data

This **Data** element defines all the data related aspects for its parent template element.



The **Data** element contains:

- Zero or 1 query
- Zero or 1 sort
- Zero or 1 filters
- Zero or 1 limit

3.8.1 Queries

A *query* defines a set of *homogenous* elements from a *single* data source. A query is defined using a data element inside a valid template element. See the element table for the list of template elements that can host queries.

```
− <data source="XML" handle="$1">
    <query>Project/Requirements/PRRequirement</query>
    <sort/>
    <filter/>
    <limit>0</limit>
  </data>
  <assignments/>
```

The query has the following properties that define it:

Property	Description	Mandatory
Source	The ID of the data source the data will be pulled from. Must be one of the IDs from the data source list	yes
handle	The ID of this query. *Must be unique*	yes
context	The ID of the parent query if any. The two queries must have the same data source.	no

A *query* is further defined by:

- the query path
- the sort expression (optional)
- the filter expression (optional)
- a limit expression (optional)
- recursive level
- recursive segments

3.8.1.1 Query Context

Contexts are the result of nesting elements defining data blocks. An element in a template can refer any data defined in any direct ancestor of the current element.

It is important to note that Rational Publishing Engine **does not** allow working with the data as a whole. The RPE engine, through the input loaders, performs the query and iterates the result set. At any time only one element in the result set is available. This element and all its direct ancestors form the context.

```
− <data source="XML" handle="$2" context="$1">
    <query>TracesFrom/Relationship/RelatedReq</query>
    <sort/>
    <filter/>
    <limit>0</limit>
  </data>
```

3.8.1.2 Query Path

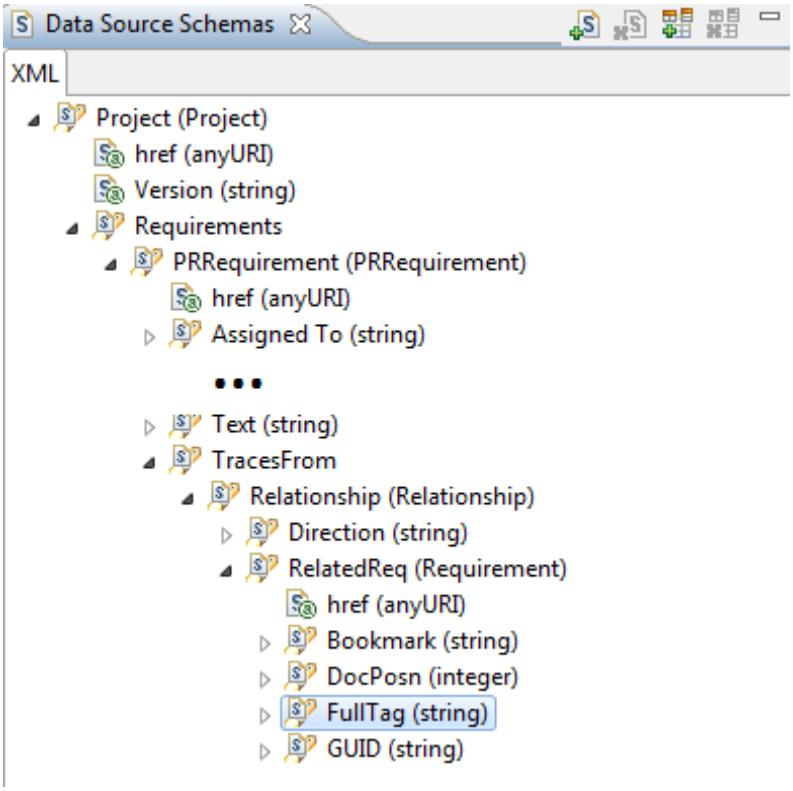
An expression that defines the location of the data in the XML. The language used by Rational Publishing Engine is a very simplified version of XPath which only supports selection with no filtering or functions being allowed.

If a query is defined in the context of another query it is a child query and its path is relative to the parent query. A query that has no parent context is named “top level query”.

When resolving the data Rational Publishing Engine uses the **absolute path of a query** which is defined recursively:

- If the query is a top level the absolute path is the query path
- If the query is not top level the absolute path is the parent query absolute path + the query path

Example:

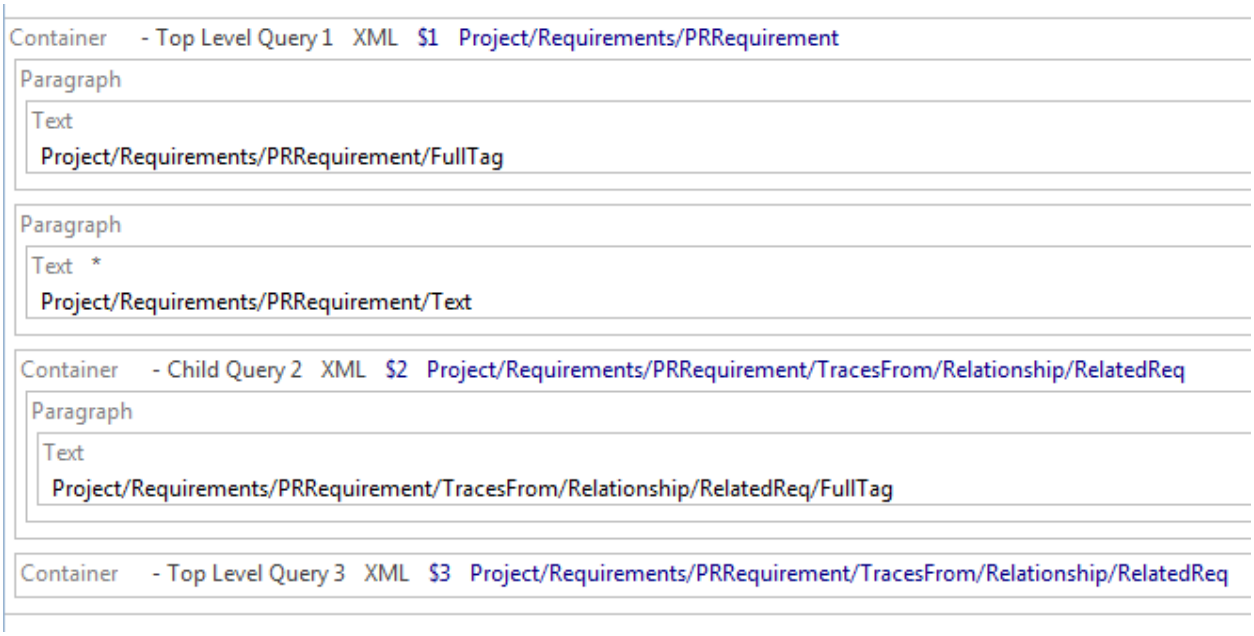


Some examples of queries in the context of the above schema:

	Query Paqth	**Query Absolute Patj**
Top level query 1	Project/Requirements/PRRequirement	Project/Requirements/PRRequirement
Query 2, child of query 1	TracesFrom/Relationship/RelatedReq	Project/Requirements/PRRequirement/TracesFrom/Relationship/RelatedReq
Top level query 3	Project/Requirements/PRRequirement/TracesFrom/Relationship/Re	Project/Requirements/PRRequirement/TracesFrom/Relationship/RelatedReq

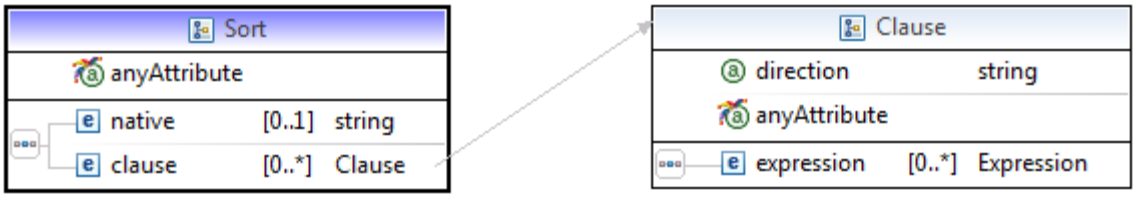
	Query Paqth	Query Absolute Patj
	latedReq	

Rational Publishing Engine Document Studio will display query 2 and 3 identically as the absolute path of all queries that are rendered in the UI.



3.8.2 Sort

Defines the sorting to be applied to the query results.



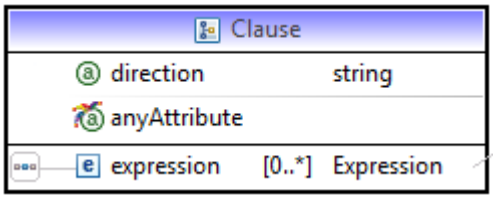
Rational Publishing Engine supports two ways of specifying the sort:

- *RPE Sort Expression*
- *Native Sort Expression*

Both type of sorts are defined as children of the <sort> element of the <data> element of the query. The sort element exists always and if it is empty it means that the query has no sort defined.

3.8.3 RPE Sort

An RPE sort is defined by list of the clause elements.



Each clause defines the data property to use and the direction of the sort. When Rational Publishing Engine compares 2 input elements to determine their relative order it will take each clause in order and compare the values of the clause property for each object. If the 2 objects have equal values for the current clause, the next clause is used otherwise the comparison ends there.

```
- <element id="1" tag="container" recursive="0" recursive_segments="1">
  - <data source="XML" handle="$1">
      <query>Project/Requirements/PRRequirement</query>
    - <sort>
      - <clause direction="asc">
          <expression result="number" type="data" context="$1">Priority</expression>
        </clause>
      - <clause direction="desc">
          <expression type="data" context="$1">Difficulty</expression>
        </clause>
      </sort>
```

Property	Description	Mandatory
direction	Must be asc or desc	yes

The data property to be used is defined by a child expression element of the clause. The expression element has 2 properties listed in the table below and the value of the expression is the name of the data property to use.

```
<expression type="data" context="$1">Difficulty</expression>
```

Property	Description	Mandatory
type	Must be **data**	yes
context	The ID of the query. Must be the a query visible in the current context	yes
result	Optional attribute that tells Rational Publishing Engine how to interpret the data property. Accepted values: numeric, text	no

3.8.3.1 Native sort

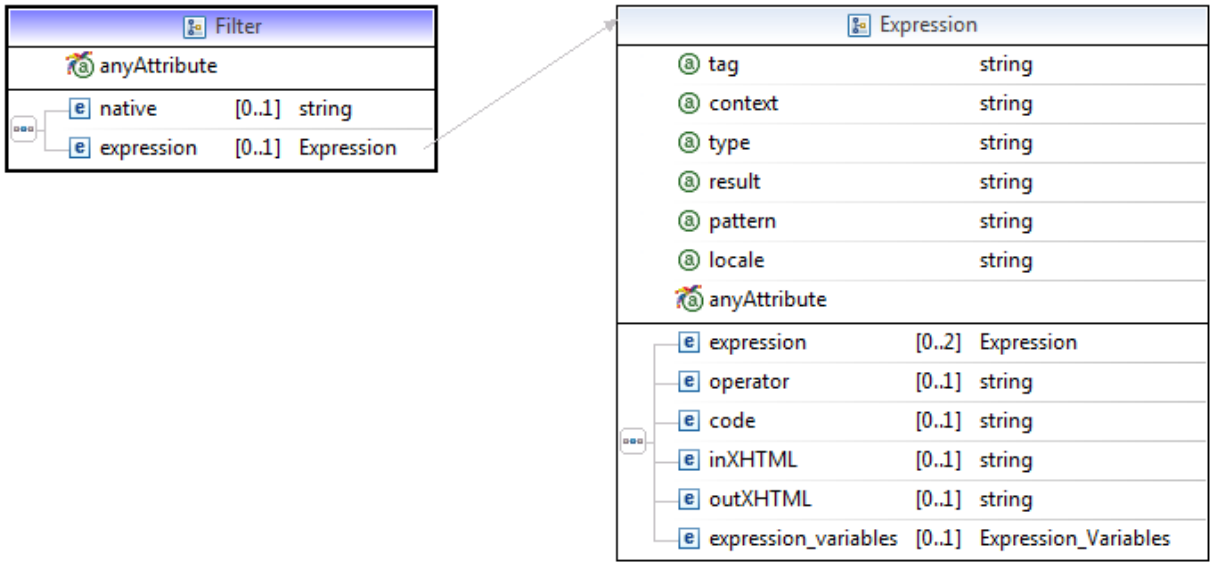
The native source is expressed using the data source query language. In the template this is represented as plain text contained in a <native> element inside the <sort> element. Rational Publishing Engine does not attempt to interpret the sort in any way, but simply dispatches it to the data source to handle it.

```
– <sort>
    <native>native sort value, specific to each data source</native>
  </sort>
```

NOTE: To date only the DOORS data source supports native sorts.

3.8.4 Filter

Defines the filter rules to use for selecting a subset of the data.



Rational Publishing Engine supports two ways of specifying the sort:

- *Scripted Filter*
- *Native Filter*

Both types of filters are defined as children of the <filter> element of the <data> element of the query. The filter element exists always and if it is empty it means that the query has no filter defined.

3.8.4.1 Scripted Filter

A scripted filter is a normal script expression defined as a child of the filter element.

```
- <element id="1" tag="container" recursive="0" recursive_segments="1">
  - <data source="XML" handle="$1">
      <query>Project/Requirements/PRRequirement</query>
    + <sort></sort>
    - <filter>
      - <expression type="script">
          <code>FullTag != "PR1"</code>
          <inXHTML>false</inXHTML>
          <outXHTML>false</outXHTML>
        - <expression_variables>
            <expression_variable type="data" context="$1">FullTag</expression_variable>
          </expression_variables>
        </expression>
      </filter>
      <limit>0</limit>
    </data>
```

3.8.4.2 Native filter

The native source is expressed using the data source's query language if such language exists. In the template this is represented as plain text contained in a <native> element inside the <filter> element. Rational Publishing Engine does not attempt to interpret the filter in any way, but simply dispatches it to the data source to handle it.

```
- <data source="XML" handle="$2" context="$1">
    <query>TracesFrom/Relationship/RelatedReq</query>
    <sort/>
  - <filter>
      <native>${title} == "Requirements"</native>
    </filter>
    <limit>0</limit>
  </data>
```

Template variables can be used in the native filter and for Rational Publishing Engine to recognize them they must be enclosed in ${}.

3.8.4.3 Limit

The limit element defines the maximum number of elements Rational Publishing Engine will use from each query. The value must be a positive number. If 0 or missing this value is ignored by Rational Publishing Engine and all query entries are used.

```
− <data source="XML" handle="$1">
    <query>Project/Requirements/PRRequirement</query>
  + <sort></sort>
  + <filter></filter>
    <limit>5</limit>
  </data>
```

3.8.5 Recursive level and recursive segments

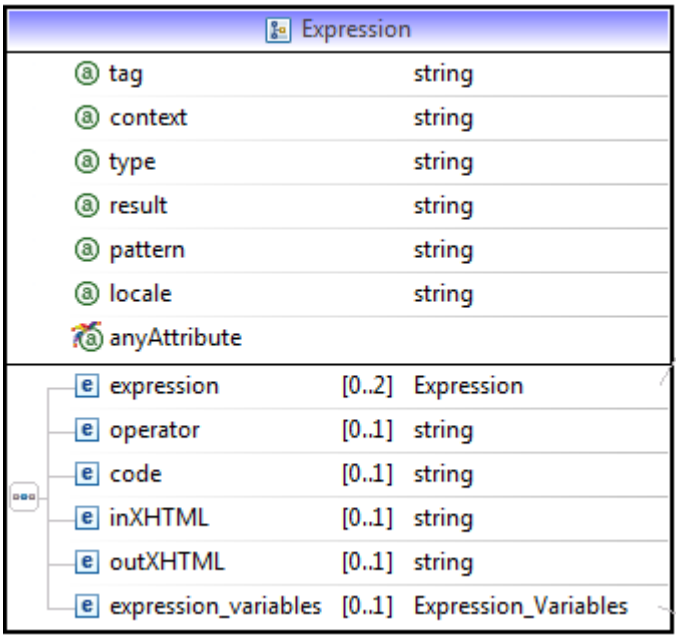
These values define the recursive properties of the query. Even though these properties define aspects of the query they are defined on the element hosting the query. This will be corrected in the next iteration of the template language.

```
− <element id="1" tag="container" recursive="0" recursive_segments="1">
  − <data source="XML" handle="$1">
```

NOTE: setting these values on elements that do not have an associated query has no effect

3.9 Expressions

This element is the vehicle for using data in the template.



Any data present in the template is introduced by an expression element. There following expression types are supported in Rational Publishing Engine.

Type	Description
constant	Plain text
variable	A user defined variable in the template
data	A property available in the current data context
script	A Java Script expression

3.9.1 Constant Expression

```
<expression tag="content" type="constant">Requirement ID: </expression>
```

A constant expression is plain text.

Property	Description	Mandatory
tag	Depends on what the expression is used for. Usually is **content**	yes
type	Must be **constant**	yes
<value>	The plain text which is the value of the expression	no

3.9.2 Data expression

```
<expression tag="content" type="data" context="$2">FullTag</expression>
```

A data expression references one property for a query in the current context path.

Property	Description	Mandatory
tag	Depends on what the expression is used for. Usually is **content**	yes
type	Must be **data**	yes
context	The ID of the query from which the property will be read.	
<value>	The name of the **data property** to use. The value of the property for the current element becomes the expression's value	no

3.9.3 Variable Expression

```
<expression tag="content" type="variable">title</expression>
```

A variable expression references one template variable.

Property	Description	Mandatory
tag	Depends on what the expression is used for. Usually is **content**	yes
type	Must be **variable**	yes
<value>	The name of the **variable** to use. The value of the variable at the time of the evaluation becomes the expression's value	no

3.9.4 Script expression

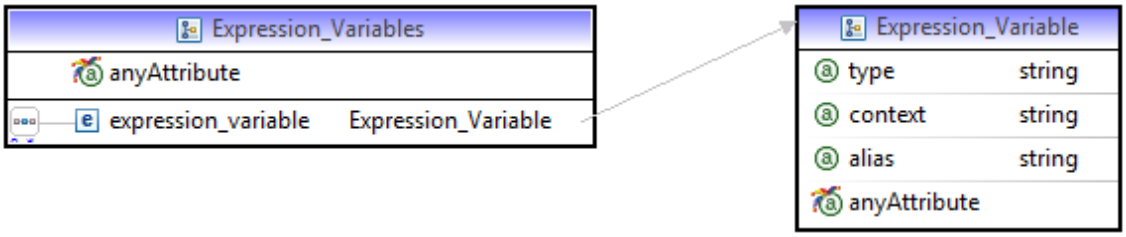
A script expression is defined by a Java Script code snippet along with all the variables and data properties it references.

```
- <expression tag="content" type="script">
    <code>FullTag + "_" + pos</code>
    <inXHTML>false</inXHTML>
    <outXHTML>false</outXHTML>
  - <expression_variables>
      <expression_variable type="data" context="$1">FullTag</expression_variable>
      <expression_variable type="variable">pos</expression_variable>
    </expression_variables>
  </expression>
```

The expression element for a script expression has the following attributes:

Attribute	Description	Mandatory
tag	Depends on what the expression is used for. Usually is **content**	yes
type	Must be **script**	yes

Each data source property or variable used in the script expression must be listed using an **expression_variable** element inside the **expression_variables** element of the expression.



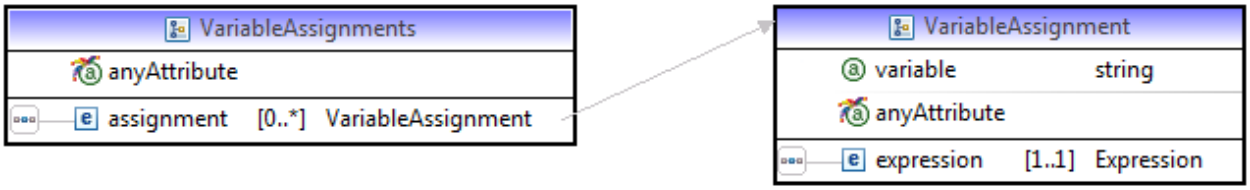
Every variable of a data property defined in this section of the expression can be used in the script code as a variable.

Attribute	Description	Mandatory
type	Must be **variable** for variables and **data** for data properties	yes
context	For data properties defines the ID of the query to which the property belongs. Not used for variables	no
<value>	The name of the variable or of the data propery	

The **code** element is plain text and contains the JavaScript code. The script must be valid JavaScript with no syntax errors.

3.9.5 Assignments

Assignments are template constructs that allow you to give new values to variables.



A single element can have 0 or more assignments. Each assignment is defined by an <assignment> element and all the <assignment> elements being grouped under an <assignments> element. If no assignment exists for the element the <assignments> element is omitted.

Attribute	Description	Mandatory
Variable	The name of the variable that receives the value	yes
<value>	An expression that gives the value.	yes

```
– <element id="24" tag="container" recursive="0" recursive_segments="1">
  – <assignments>
    – <assignment variable="pos">
      – <expression type="script">
          <code>DocPosn+1</code>
          <inXHTML>false</inXHTML>
          <outXHTML>false</outXHTML>
        – <expression_variables>
            <expression_variable type="data" context="$1">DocPosn</expression_variable>
          </expression_variables>
        </expression>
      </assignment>
    – <assignment variable="title">
        <expression type="constant">Sample Requirements Document</expression>
      </assignment>
    </assignments>
```

3.9.6 Conditions

The conditions are defined as expression elements directly under the <element> they apply to. They are regular script expressions for which the “tag” attribute is set to be “**condition**”. There can be a maximum of 1 condition per element.

```
- <element id="19" tag="container" recursive="0" recursive_segments="1">
    <assignments/>
  - <expression tag="condition" type="script">
      <code>Stability != "Low"</code>
      <inXHTML>false</inXHTML>
      <outXHTML>false</outXHTML>
    - <expression_variables>
        <expression_variable type="data" context="$1">Stability</expression_variable>
      </expression_variables>
    </expression>
```

4 RPETemplateElements.xsd

This file describes the formatting properties available for each element.

```
- <elements>
    <element tag="template comment"> </element>
    <element tag="javascript code"> </element>
  + <element tag="data source configuration"></element>
  + <element tag="styled text"></element>
  - <element tag="text">
    - <feature tag="common">
        <!-- inherited style name -->
      + <property name="style name"></property>
        <!-- parent masterpage -->
      + <property name="masterpage"></property>
        <!-- page change behavior-->
      + <property name="force page change"></property>
      + <property name="heading level offset"></property>
      + <property name="target region"></property>
      </feature>
    + <feature tag="date"></feature>
    + <feature tag="data"></feature>
      <!-- List properties -->
```

Rules

1. The grouping in features is irrelevant and used for display purposes mainly
2. The properties must have unique names (the name attribute).
3. The same property can be used in more than 1 element but all its occurrences must be identical.

4.1 Property Domain

Each formatting property has a number of properties that define how Rational Publishing Engine displays, edits and interprets the property. These properties are stored as child elements of the property.

```
- <property name="force page change">
    <any value="false"/>
  - <domain>
      <!-- true/false -->
      <type value="boolean"/>
      <item value="true"/>
      <item value="false"/>
      <item value="" default="true"/>
    </domain>
  - <description>
    - <text>
        When true, a page break is inserted if current masterpage is similar with the previous one
      </text>
      <output type="Html">not supported</output>
      <output type="Word">supported</output>
      <output type="Pdf">supported</output>
      <output type="XslFo">supported</output>
      <output type="Legacy Pdf">supported</output>
    </description>
</property>
```

Propery	Description
any	Through its value attribute defines if this property accepts any text as value. Used by Rational Publishing Engine to restrict the UI editor but otherwise has no impact at runtime. If an invalid property value is specified the result is undefined behavior
description/text	Used to document the intended usage for this property. Used only for display purposes, no impact at runtime
Description/output	One per output type supported by Rational Publishing Engine. Specifies through its value if the property is supported by the specified output. Used only for display purposes, no impact at runtime

The domain element allows you to define more formally the values accepted by the value. The types of domains supported by Rational Publishing Engine are listed below.

4.1.1 Enumerated Domain

Each accepted entry is introduced through an *item* element. The *value* is stored in the value attribute of the *item* element. If a default value exists this is marked by the *default* attribute set on the element with the value true.

```
- <property name="underline">
    <any value="false"/>
  - <domain>
      <item value="" default="true"/>
      <item value="true"/>
      <item value="false"/>
      <item value="single"/>
      <item value="words"/>
      <item value="double"/>
      <item value="dotted"/>
      <item value="thick"/>
      <item value="dash"/>
      <item value="dash long"/>
      <item value="dot dash"/>
      <item value="dot dot dash"/>
      <item value="wavy"/>
      <item value="dotted heavy"/>
      <item value="dash heavy"/>
      <item value="dash long heavy"/>
      <item value="dot dash heavy"/>
      <item value="dot dot dash heavy"/>
      <item value="wavy heavy"/>
      <item value="wavy double"/>
    </domain>
  - <description>
      <text>Draws a line under the text</text>
      <output type="Html">supported : true, single</output>
      <output type="Word">supported</output>
      <output type="Pdf">supported</output>
      <output type="XslFo">supported : true, false</output>
      <output type="Legacy Pdf">supported : true, false, single</output>
    </description>
  </property>
```

4.1.2 Boolean Domain

An instance of enumerated domain with the type defined to boolean.

```
- <domain>
    <!-- true/false -->
    <type value="boolean"/>
    <item value="true"/>
    <item value="false"/>
    <item value="" default="true"/>
  </domain>
```

4.1.3 Integer Domain

The domain defines the range of the numeric values accepted by the property through the min and max elements. A default value can be specified through an *item* element with a *default* attribute. The type is integer.

```
- <property name="all borders margin">
    <any value="true"/>
  - <domain>
      <type value="integer"/>
      <min value="0"/>
      <max value="1000"/>
      <item value="" default="true"/>
    </domain>
```

4.1.4 Double domain

The domain defines the range of the numeric values accepted by the property through the min and max elements. A default value can be specified through an *item* element with a *default* attribute. The type is double.

```
- <property name="multiple line spacing">
    <any value="true"/>
  - <domain>
      <type value="double"/>
      <min value="0.50"/>
      <max value="130.00"/>
      <item value="" default="true"/>
    </domain>
```

5 RPEElementsRules.xml

The file defines the nesting rules supported by Rational Publishing Engine and enforced by Rational Publishing Engine Document Studio.

```
- <element tag="table" alias="" display="Table">
  - <disallowed>
      <element_ref tag="header"/>
      <element_ref tag="footer"/>
      <element_ref tag="region"/>
      <element_ref tag="page break"/>
      <element_ref tag="section break"/>
      <element_ref tag="document break"/>
      <element_ref tag="content"/>
      <element_ref tag="masterpage"/>
      <element_ref tag="table of contents"/>
      <element_ref tag="table of tables"/>
      <element_ref tag="table of figures"/>
      <element_ref tag="snippet"/>
    </disallowed>
  - <children>
      <element_ref tag="row"/>
      <element_ref tag="table container"/>
      <element_ref tag="template comment"/>
      <element_ref tag="javascript code"/>
    </children>
  </element>
```

For each element in Rational Publishing Engine an “element” entry exists in this file.

Propery	Description
tag	The tag of the element described by the current entry
alias	(optional) for elements that are identical to Rational Publishing Engine but have to have different nesting rules in the context this is the tag of the “base” element. Example: containers in tables or rows versus top level container elements. `- <element tag="row container" alias="container" display="Row Container">`
display	The name used for display purposes

Each element entry defines 2 lists:

- **Disallowed** – the list of elements that cannot be nested in this element
- **Children** – the list of elements accepted as children by this element

5.1 Nesting rule composition

When interpreting the rules Rational Publishing Engine Document Studio uses the following rule to determine if an element can be added as a children of another element:

1. If the element is not listed in the “children” list, return false.
2. If the element is listed in the disallow list, return false.
3. Recursively check if any of the parents of the current element disallow using that element.