
Subject: Extensible RailML

Posted by [Heidrun Jost](#) on Tue, 06 Mar 2007 08:00:17 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hello,

for usage of RailML in the RBC project there are a lot of extensions necessary. I think, this is a general problem for each usage, to place the customer specific extensions in RailML.

The approach of RailML is not an extensible XML, but a rigid amount of defined elements and attributes.

Customer specific extensions are only possible by a request at the forum (for common interesting items) or by usage of "GeneralElements".

The usage of "GeneralElements" has important disadvantages:

Independent of the usage of "GeneralElements" in one file together with the other RailML elements or in an separate file, the XML file containing the "GeneralElements" is not RailML compliant.

Another disadvantage is the definition of additional identifiers to establish a relationship between an element and additional attributes.

For check tools a lot of consistency checks are required.

It is difficult to realize the relationship between the RailML file and the additional information.

I think a better solution is the definition of an extensible RailML using wildcards.

The W3C XML schema provides place holders `xs:any` and `xs:anyAttributes`. Elements and attributes out of a given namespace can be used by usage of these place holders

The owner of the schema can control the occurrences of the place holders.

By usage of processContents attribute it is possible to define the kind of check of the XML content concerning to the place holders (skip - no check, lax - lax check, strict - strictly validation check).

The following example shows the usage of the any and anyAttribute.

At first a RailML schema with the definition of signalGroup and signal with the attributes "name" and "stationID". In addition an anyAttribute definition for customer attributes and any definition for customer elements.

File railML.xsd:

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```

elementFormDefault="qualified">
<xs:element name="signalGroup">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="signal" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="stationID" type="xs:string"/>
            <xs:any minOccurs="0" processContents="strict"/>
          </xs:sequence>
          <xs:anyAttribute namespace="##any" processContents="strict"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

The second schema file defines the customer attribute "custAttr" and customer element "custElement".

File cust.xsd:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.thalesgroup.com"
elementFormDefault="qualified">
<xs:element name="custElement">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="custElementName" type="xs:string"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:attribute name="custAttr">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="si1|si2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:schema>

```

The following xml file contains a signal with a combination of the defined elements and attributes of the two schema files above. The file will be checked against the two schema

files. The additional customer attributes and elements are marked by the defined namespace "cust".
railML.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<signalGroup
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cust="http://www.thalesgroup.com"
xsi:noNamespaceSchemaLocation="railML.xsd"
xsi:schemaLocation="http://www.thalesgroup.com cust.xsd">
<signal cust:custAttr="si1">
<name>signal1</name>
<stationID>stationID1</stationID>
<cust:custElement>
  <cust:custElementName>ExitInfo</cust:custElementName>
</cust:custElement>
</signal>
<signal>
<name>signal2</name>
<stationID>stationID2</stationID>
</signal>
</signalGroup>
```

This is a way to check the RailML definitions only
and to ignore the unknown customer definitions.
The customer definitions will be checked by another schema.
All files with customer extensions are RailML compliant.

This is also a way to "clean up" the current schema
definition from special customer items (if existent)
and to restrict the definitions to topology aspects only.

Best regards,
Heidrun Jost
