XML Structures and IDL-XML Mapping

To understand the functionality and usage of the XML Mapping Editor, it is necessary to look at the possible XML structures and how they are mapped to Software AG IDL.

This chapter covers the following topics:

- XML Structure Description
- Basic IDL-XML Mapping
- Arrays
- Groups
- IN / OUT / IN OUT Parameters

XML Structure Description

An XML structure is the type of an XML document, that is, the blueprint to build or parse the XML document. Every program of every library within a Software AG IDL file corresponds to at least two XML structures: one for the incoming and one for the outgoing XML document. The Error or Fault directions are also described as XML structures. There are InErr and OutErr XML structures that are returned by the broker or server in case of broker or data errors or servicing problems. The XML structures all start with one root node (corresponding to the library/program combination of the Software AG IDL file), and all XML elements and attributes are linked under that root node and may be further cascaded.

The XML structure may be represented as a tree of XML structure nodes (so-called XML parts). For the EntireX XML/SOAP Wrapper, no cyclic XML structures (that is, non-tree XML structures) are allowed because mapping to Software AG IDL parts would be illegal. However, in general this is not a severe restriction because Software AG IDL parts may not be cyclic either.

The XML parts have various properties. Important properties are the node name (tag name) and the node type (element or attribute). Other properties are the data type and length, minimum and maximum occurrence, the default values, the encoding, or the used or defined namespace. XML parts may have links to IDL nodes (their IDL mapping links, see below).

In the XML Mapping Editor, the XML structures are displayed as node trees. For XML structures, sample XML documents can be generated to visualize the expected or generated XML documents.

Basic IDL-XML Mapping

Mapping between IDL and XML describes the location of the data in the XML documents that correspond to the RPC parameters. Getting data for the RPC request is called incoming direction, putting RPC response data into an XML document is called outgoing direction. RPC parameters can correspond to elements or attributes.

There are various basic strategies for generating elements or attributes from a given Software AG IDL. The most important strategies are:

- Element-preferred mapping: Model the library/program and every IDL parameter as an element. The elements are cascaded in the same way as their IDL counterparts. The tree of IDL parts and the tree of XML elements are very similar. Arrays or repeated groups may get envelope elements.
- Attribute-preferred mapping: Model the library/program as root element. Model Arrays and repeated groups as elements, possibly with envelope elements around them. Model all other IDL parameters as attributes.
- SOAP: Construct a SOAP message format according to the SOAP specification. Every Software AG IDL part relates to an element in the SOAP:Body portion. Further fine-tuning according to the specification may be done.

Some users prefer element modelling, others like to add attributes to existing elements wherever possible. To minimize the work of building XML structures in the XML Mapping Editor, default mappings are available. The XML Mapping Editor allows you to choose an element-preferred, an attribute-preferred or a SOAP-conformant strategy (plus various detail level switches). The element-preferred mode generates only element nodes, whereas the attribute-preferred mode generates attributes wherever possible.

Example

IDL File

```
Library 'EXAMPLE' Is
 Program 'CALC' Is
  Define Data Parameter
   1 Operation (A1) In
   1 Operand_1 (I4) In
   1 Operand_2 (I4) In
   1 Function_result (I4) Out
  End-Define
```

Element-preferred Mode

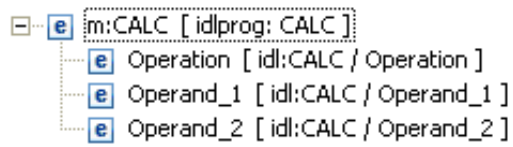
In the element-preferred mode, the IDL is mapped to the incoming XML document:

```
<CALC>
    <Operation> ... </Operation>
    <Operand_1> ... </Operand_1>
    <Operand_2> ... </Operand_2>
</CALC>
```

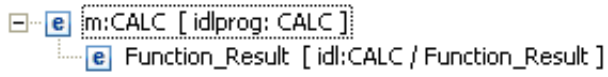
and the outgoing XML document:

```
<CALC>
    <Function_result> ... </Function_result>
</CALC>
```

The corresponding XML structure trees are:



and



Attribute-preferred Mode

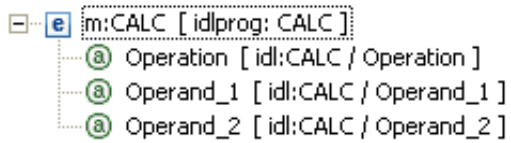
In the attribute-preferred mode, the Software AG IDL above is mapped to the incoming XML document:

```
<CALC Operation="..." Operand_1="..." Operand_2="..."/>
```

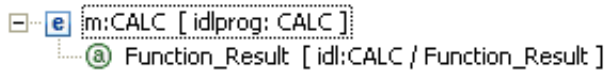
and the outgoing XML document.

```
<CALC Function_result="..."/>
```

The corresponding XML structure trees are:



and



SOAP-conformant Mapping

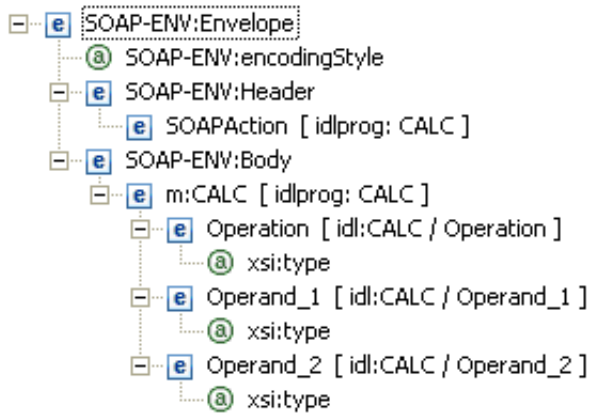
In SOAP-conformant mapping or the SOAP-conformant mode, the above Software AG IDL is mapped to the incoming XML document:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <CALC>
         <Operation xsi:type="SOAP-ENC:string">...</Operation>
         <Operand_1 xsi:type="SOAP-ENC:int">...</Operand_1>
         <Operand_2 xsi:type="SOAP-ENC:int">...</Operand_2>
      </CALC>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

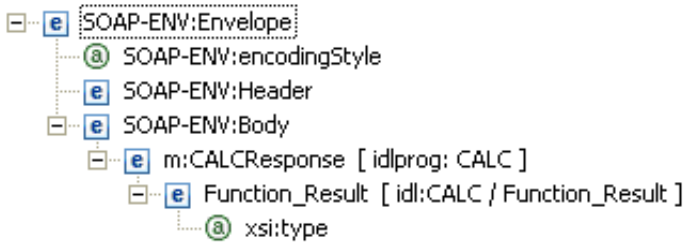
and the outgoing XML document:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
   <SOAP-ENV:Header></SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <CALCResponse>
         <Function_result xsi:type="SOAP-ENC:int">...</Function_result>
      </CALCResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The corresponding XML structure trees:



and



Arrays

The IDL may contain array parameters, i.e. basic data types that are to be repeated for a specified number of times. See following example:

```
...
1 Operation (A1/5) In
...
```

The `Operation` may be repeated 5 times. In this case `Operation` must correspond to an XML element, because with attributes the repetition cannot be modelled. The corresponding XML document may then contain up to 5 `Operation` elements:

```
...
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
<Operation> ... </Operation>
...
```

or it may contain a surrounding parent element (envelope) `Operation`:

```
...
<Operations>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
   <Operation> ... </Operation>
</Operations>
...
```

A switch in the XML Mapping Editor determines whether arrays get a surrounding parent element or not.

There is a maximum of three dimensions per IDL array. Each of the dimensions can have upper and lower bounds defined. The format is as follows:

```
1 Operation (A1/3,6,8) In
```

which corresponds to a possible XML document:

```
<Operations1>
   <Operations2>
      <Operations3>
         <Operation> ... </Operation>
         ...
      </Operations3>
      ...
   </Operations2>
   ...
</Operations1>
```

and XML structure:

```
...
Operations1 (element)
   Operations2 (element, minocc=1, maxocc=3)
      Operations3 (element, minocc=1, maxocc=6)
         Operation (element, minocc=1, maxocc=8)
```

Groups

The IDL may contain entries that represent groups of parameters.

Example

```
Library 'EXAMPLE' Is
 Program 'ADD' Is
  Define Data Parameter
   1 Parameters In
```

```
    2 Operand_1 (I4)
    2 Operand_2 (I4)
   1 Function_result (I4) Out
  End-Define
```

The entry parameters are a group of two Operands. Groups will always be converted to XML elements; the group elements will be mapped either to elements or to attributes.

Element-preferred mode

```
<ADD>
   <Parameters>
      <Operand_1> "..." </Operand_1>
      <Operand_2> "..." </Operand_2>
   </Parameters>
</ADD>
```

Attribute-preferred mode

```
<ADD>
   <Parameters Operand_1="..." Operand_2="..."/>
</ADD>
```

Array of Groups

Groups can be used in arrays, for example:

```
1 myparams (/5,4) In
  2 Operation (I4)
  2 Description (A80)
  2 Mygroup
    3 Operand_1 (I2)
    3 Operand_2 (I2)
  2 Options (A1/4)
```

Using the element-preferred mode, this IDL structure may be mapped to:

```
<Myparms1>
   <Myparms2>
     <Myparms>
       <Operation> "..." </Operation>
       <Description> "..." </Description>
       <Mygroup>
         <Operand_1> "..." </Operand_1>
         <Operand_2> "..." </Operand_2>
       </Mygroup>
       <Options1>
         <Options>"..."</Options>
         ...
       </Options1>
     </Myparms>
     ...
   </Myparms2>
   ...
</Myparms1>
```

The corresponding XML structure for the element-preferred strategy is then:

```
...
Myparms1 (element)
  Myparms2 (element, minocc=1, maxocc=5)
    Myparms (element, minocc=1, maxocc=4)
    Operation (element, I4)
    Description (element, A80)
    Mygroup (element, group, minocc=1, maxocc=1)
      Operand_1 (element, I2)
      Operand_2 (element, I2)
   Options1 (element, minocc=1, maxocc=4)
      Options (element, A1)
```

Grouping XML Elements or Attributes

You can introduce new elements by grouping one or more existing elements or attributes.

This is especially useful when the IDL contains many simple data types that could be semantically grouped. This will increase the level of hierarchies in the XML document without affecting the IDL.

Example

```
Library ’EXAMPLE’ Is
  Program ’SIMPLE’ Is
    Define Data Parameter
      1 par1 (A1) In
      1 par2 (A1) In
      1 par21 (A1) In
      1 par22 (I1) In
      1 par23 (I4) In
      1 par3 (A1) In
      1 par31 (I2) In
      1 par32 (I4) In
      1 par4 (I1) In
      1 par5 (A1) In
    End-Define
```

will be transformed by the attribute-preferred strategy to:

```
<SIMPLE par1="..." par2="..." par21="..."
par22="..."

par3="..." par31="..." par32="..."
par4="..." par5="..."/>
```

or with the element-preferred strategy to:

```
<SIMPLE>
   <par1> ... </par1>
   <par2> ... </par2>
   <par21> ... </par21>
   <par22> ... </par22>
   <par23> ... </par23>
   <par3> ... </par3>
   <par31> ... </par31>
   <par32> ... </par32>
   <par4> ... </par4>
   <par5> ... </par5>
<SIMPLE>
```

You can now reorganize the XML structure, for example to:

```
<SIMPLE par1="..." par4="..." par5="...">
     <par2 par21="..." par22="..." par23="...">... </par2>
     <par3 par31="..." par32="..."> ... </par3>
</SIMPLE>
```

IN / OUT / IN OUT Parameters

The incoming XML request must correspond to the incoming `IN` and `IN OUT` IDL parameters, and the (created) outgoing XML response must contain the `IN OUT` and `OUT` IDL parameters. In the incoming XML structure, there is no difference between `IN` and `IN OUT` parameters; the same applies to `IN OUT` and `OUT` parameters for the outgoing XML document.

Make sure that all IDL parameters marked as `IN` are properly mapped to (incoming) XML parts. Otherwise, the XML/SOAP Runtime can only assign a null representation (value "0", empty string, Boolean "false" etc.) to the respective unmapped IDL parameters. This is probably not the desired value to be sent to the server. The XML Mapping Editor issues a warning when unmapped IN parameters are found.