webMethods EntireX

EntireX Web Services Wrapper for Natural

Version 9.7

October 2014

WEBMETHODS

This document applies to webMethods EntireX Version 9.7.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-EEXXWEBSERVICESWRAPPER-97-20160805NAT

Table of Contents

I

1 Using Web Services Wrapper for Natural

The Web Services Wrapper for Natural allows you allows you to generate Web services from Natural subprograms in a NaturalONE project in Eclipse. The generated Web service objects, which are WS-Stack Web service archives (file extension .aar) can be deployed in a Web Services Stack runtime, registered in CentraSite and tested with the XML Tester. Web services client applications can then access these Web services that expose some business logic implemented by Natural server components.

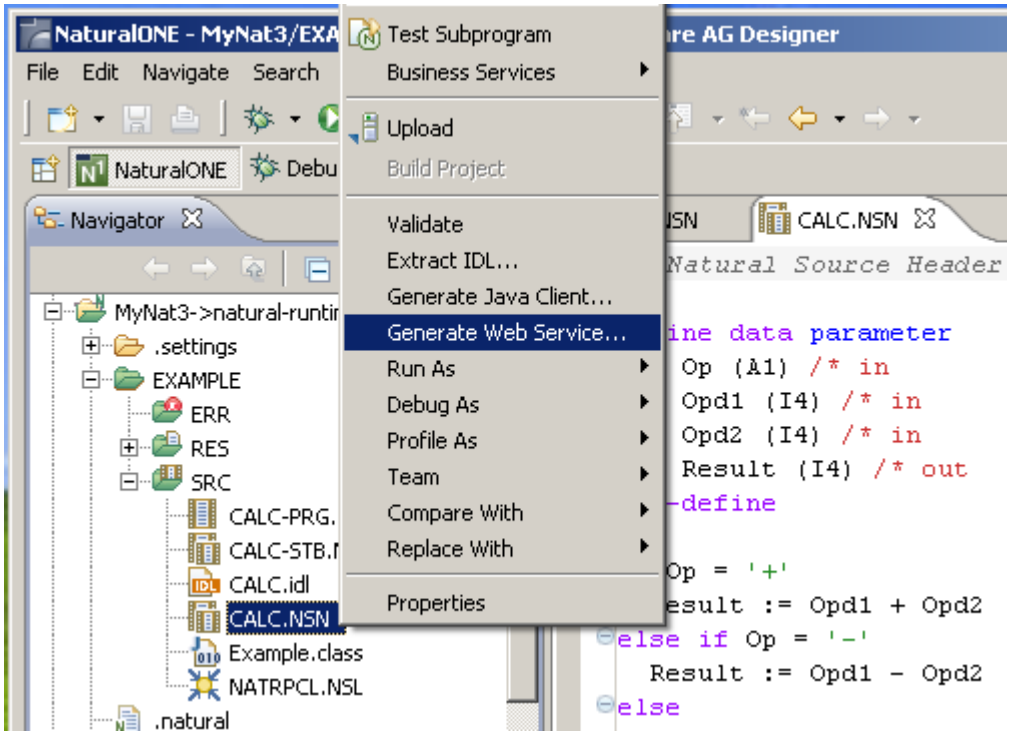
This chapter describes the steps for generating a Web service using Web Services Wrapper for Natural.

Prerequisites

To use the Web Services Wrapper for Natural you need Software AG Designer with the NaturalONE and EntireX plug-ins installed.

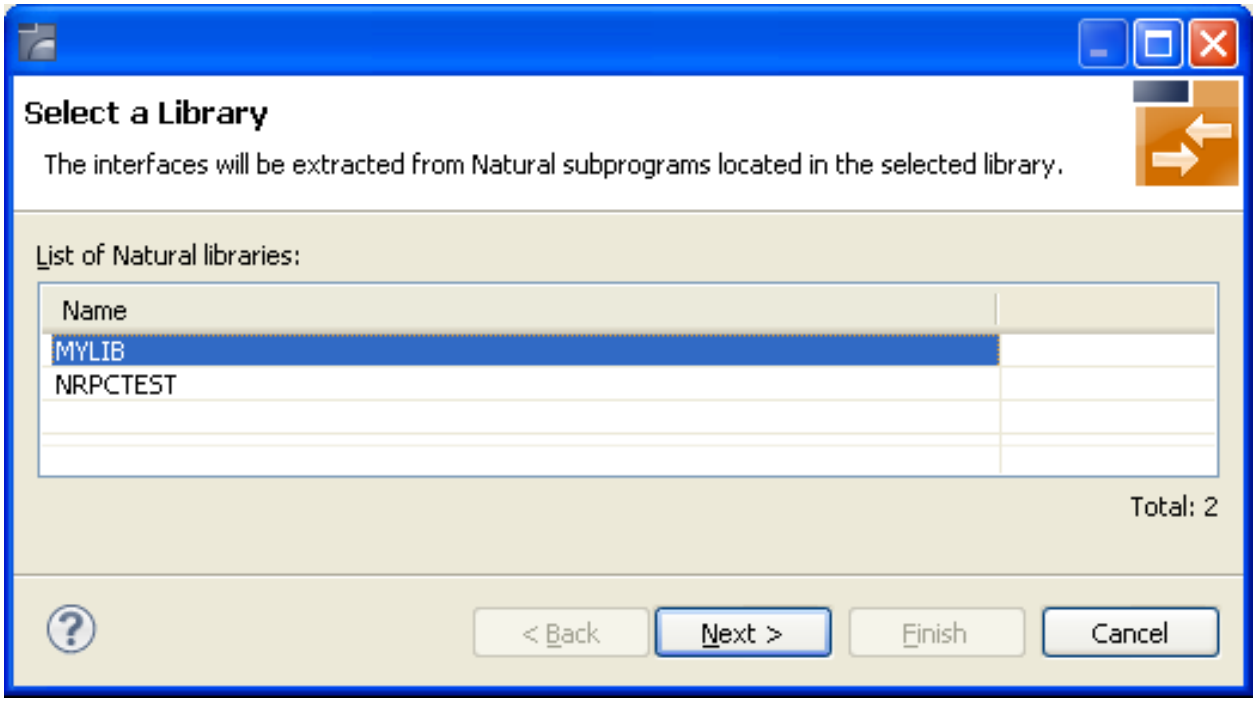
Step 1: Start the Web Services Wrapper for Natural

To start the Web Services Wrapper for Natural, select a Natural subprogram (file extension .NSN) located in a library of a NaturalONE project, and from the context menu choose **Generate Web Service...**. Alternatively you can start the Web Services Wrapper for Natural from the context menu of the Natural source folder or any parent folder in the project, including the Natural library and the Project folder.



Step 2: Select the Natural Library (Optional)

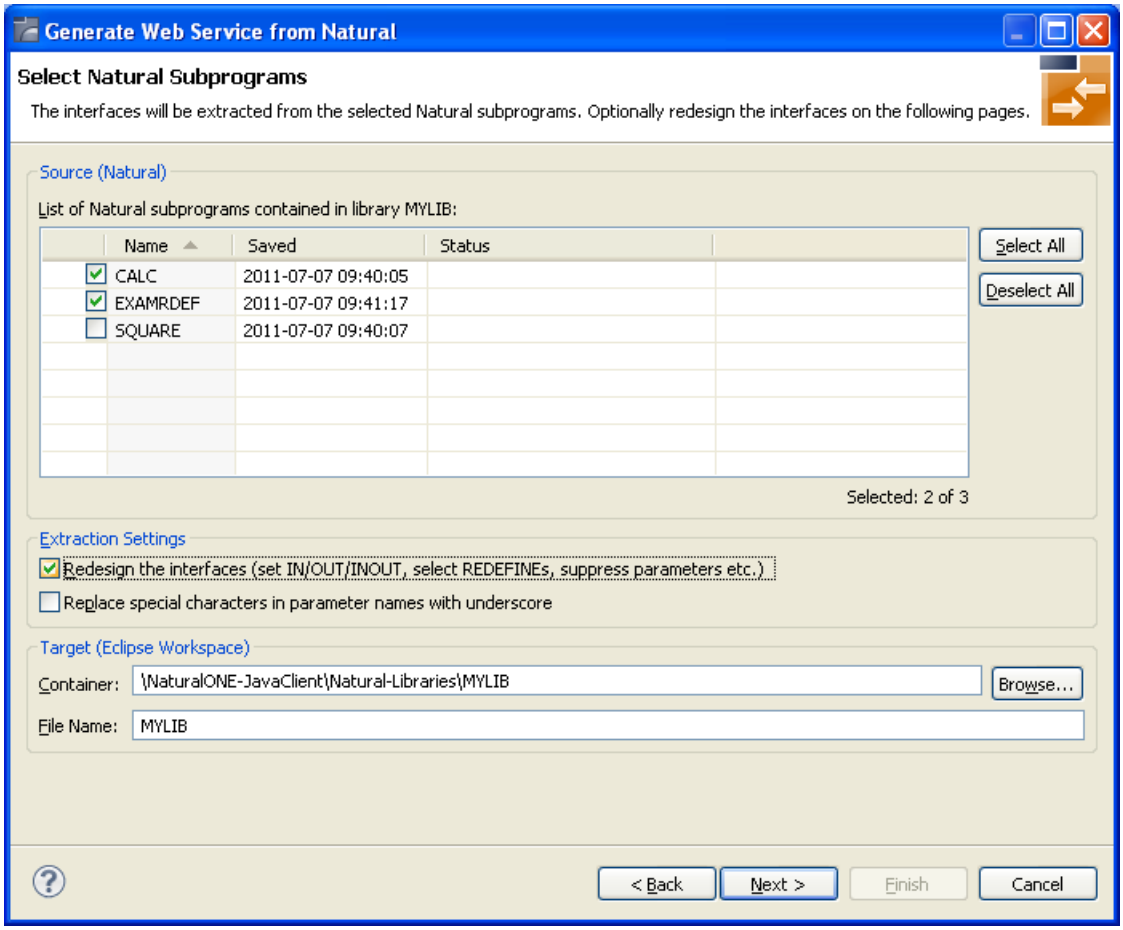
If you have started the wizard from a folder containing multiple Natural libraries, the wizard displays a page showing all available libraries from which you can select one.



Select the Natural library from the list and continue with *Step 3: Select the Natural Subprograms*.

Step 3: Select the Natural Subprograms

The following wizard page provides a list of available Natural subprograms.



In the **Source** pane, select at least one program from the list of Natural subprograms (CALLNATs). You can also choose **Select All** or **Deselect All**.

In the **Extraction Settings** pane, check **Redesign the interfaces** if you want to design the extracted interfaces to the Natural subprograms. The **Next** button will be enabled. See *Step 4: Redesign the Interface for Natural Subprograms (Optional)*. If you do not check **Redesign the interfaces**, see *Natural to IDL Mapping* in the IDL Extractor for Natural documentation for default mappings.

Check **Replace special characters in parameter names by underscore** to substitute the special characters '$', '#', '&', '@', '/' by underscores. See also *Extracting IDL Parameter Names*.

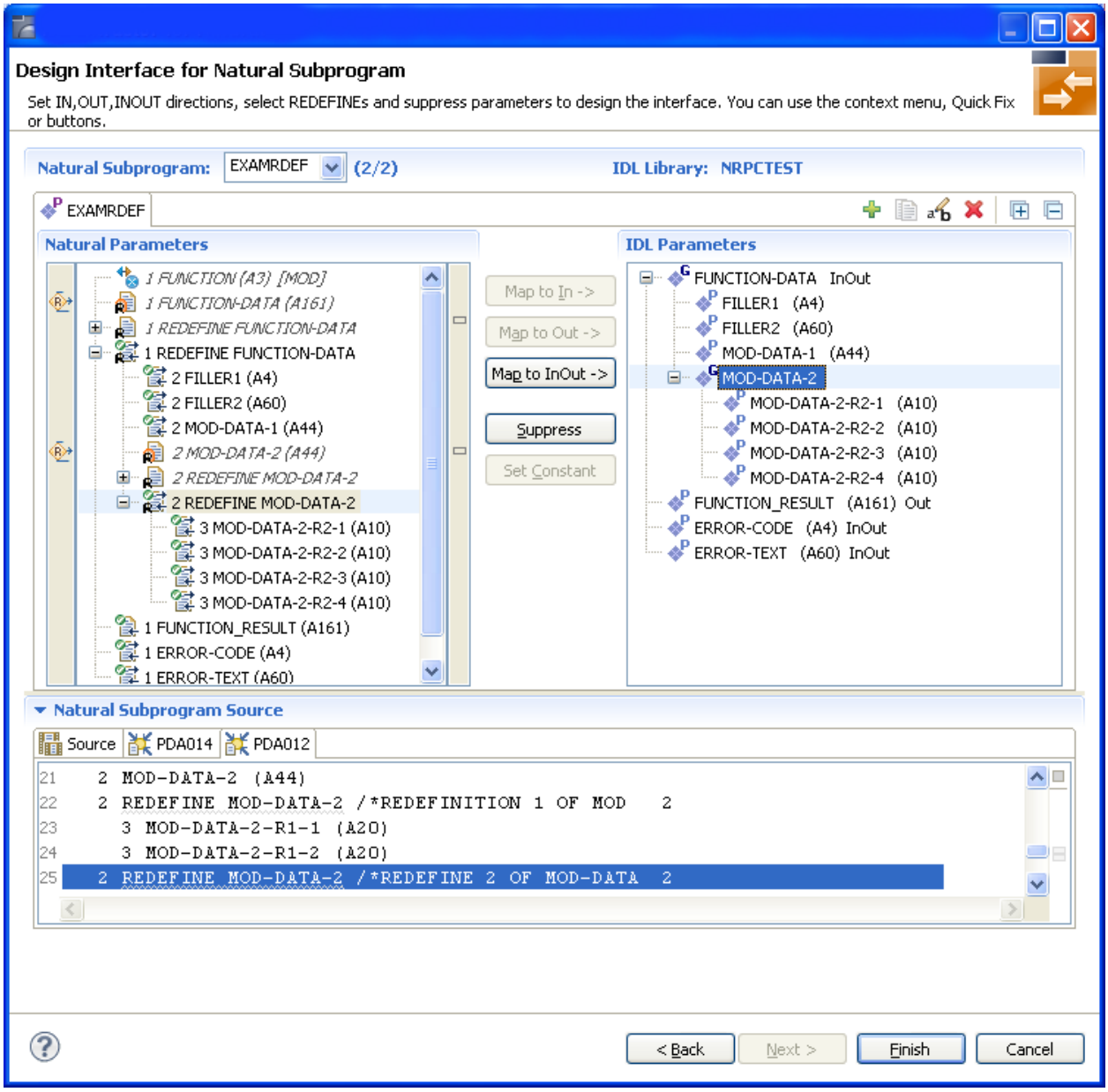
Press **Next** to continue.

- If **Redesign the interfaces** is checked, continue with *Step 4: Redesign the Interface for Natural Subprograms (Optional)*.
- Otherwise continue with *Step 5: Configure the Web Service Client*.

Step 4: Redesign the Interface for Natural Subprograms (Optional)

In this step, you can redesign the interface. This includes:

- *Extracting Multiple Interfaces*
- *Extracting Natural* `REDEFINES`
- *Extracting IDL Directions (*`IN`,`OUT`,`INOUT`*)*
- *Setting Natural Parameters to Constants*
- *Suppressing Natural Parameters*
- *Renaming a Program*



Use this page for the following tasks:

- Define the direction of parameters in the extracted interface. Choose **Map to In**, **Map to Out** or **Map to InOut** for each parameter on level 1.
- Define which parameters redefined in the Natural PDA are part of the extracted interface. Choose **Map to In**, **Map to Out** or **Map to InOut** for the `REDEFINE` base parameter or any `REDEFINE` path.
- Hide or suppress unneeded parameters in the extracted interface. Choose **Suppress**.
- Set parameters to constants and hide or suppress them in the extracted interface. Choose **Set Constant**.

This page consists of the following main parts:

- **Top line**
  The top line contains the current Natural subprogram and the IDL library name. The combo box can be used as quick navigation if more than one Natural subprogram is selected.
- **Middle**
  The middle part contains a tab item for each interface (IDL program) extracted from the Natural subprogram.

  Note: It is possible to extract more than one interface (IDL program) from a Natural subprogram. To create, rename and remove interfaces, use the toolbar on the right side of tab folder.

Icon	Function	Description
	Create	Creates a new interface (IDL program) based on the original parameters of the Natural subprogram.
	Duplicate	Creates a new interface (IDL program) based on the current interface (active tab). All modifications of the current interface are copied.
	Rename	Change the name of the current interface (active tab). The name must be unique.
	Remove	Removes the current interface (active tab). At least one interface must exist.
	Expand All	Expands the Natural and IDL tree.
	Collapse All	Collapse the Natural and IDL tree.

- **Middle left**
  Input pane. The parameters of the Natural subprogram to extract from. For each Natural subprogram parameter you can choose one of the operations **Map to In**, **Map to Out**, **Map to InOut**, **Suppress** and **Set Constant**. Additionally for `REDEFINE`s, a quick fix is available (icons on the left side of the pane) to choose which parameters redefined in the Natural PDA are part of the extracted interface.

  Notes:

  1. The mapping operations **Map to In**, **Map to Out**, **Map to InOut**, **Suppress** and **Set Constant** are also available in the context menu of the Natural parameter tree.
  2. Natural parameters that are suppressed or set to constant in the interface are rendered in italic type. For example, in the screen above, `FUNCTION (A3)` is set to constant; `FILLER1(A4)` and `FILLER2(A60)` are suppressed; `FUNCTION-DATA(A161)` and its first `REDEFINE` path are implicitly suppressed because the second `REDEFINE` path with prefix `MOD-DATA-2-R2` is selected.
  3. The value for Natural parameters set to constant are displayed behind the parameter in the Natural parameter tree (e.g. in the screen above, `FUNCTION (A3) [MOD]`).
  4. Natural parameters mapped in the interface are displayed with a green tick ( ).

- **Middle right**
  Output pane. The extracted interface (IDL).
- **Bottom**
  Reference. The Natural subprogram source and its PDA sources, each displayed in a separate tab.

Tips:

- The panes can be resized.
- To enlarge parameter lists, use the vertical bars on the side.
- You can close the bottom pane if it is not needed by clicking on the triangle next to **Natural Subprogram Source**. In this way, you have more space for viewing the upper panes.

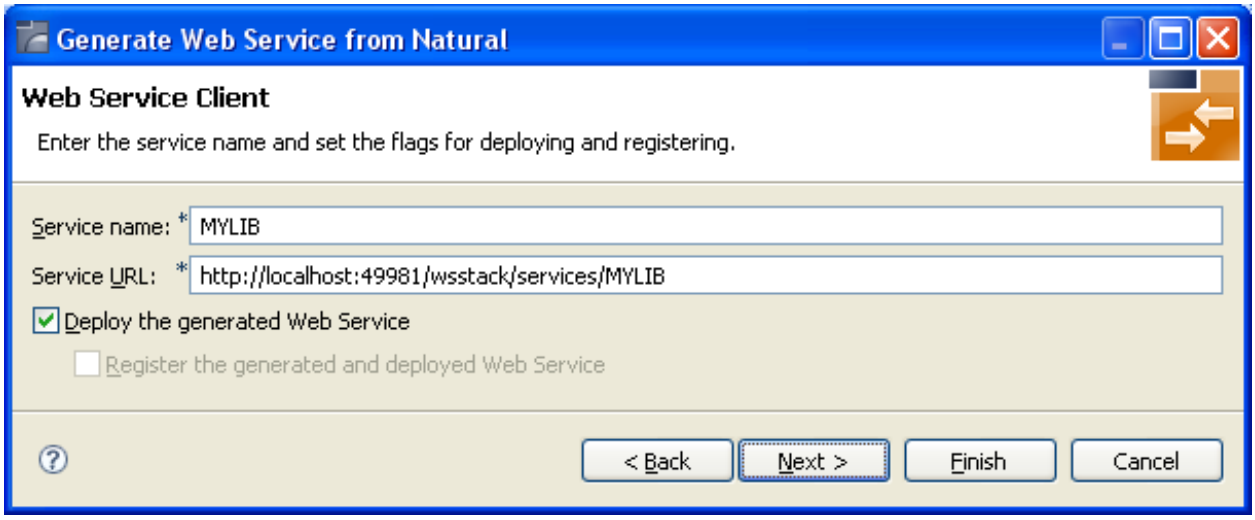
Use the quick navigation or choose **Next** to continue. If multiple Natural subprograms have been selected in the Natural subprogram selection step, redesign the next interface. The amount of subprograms extracted so far is indicated by the fraction next to the title (current/total).

If multiple Natural subprograms have been selected in the Natural subprogram selection step, redesign the next interface. The number of subprograms extracted so far is indicated by the fraction next to the title (current/total).

If only one Natural subprogram has been selected or no further one has to be redesigned, continue with *Step 5: Configure the Web Service Client*.

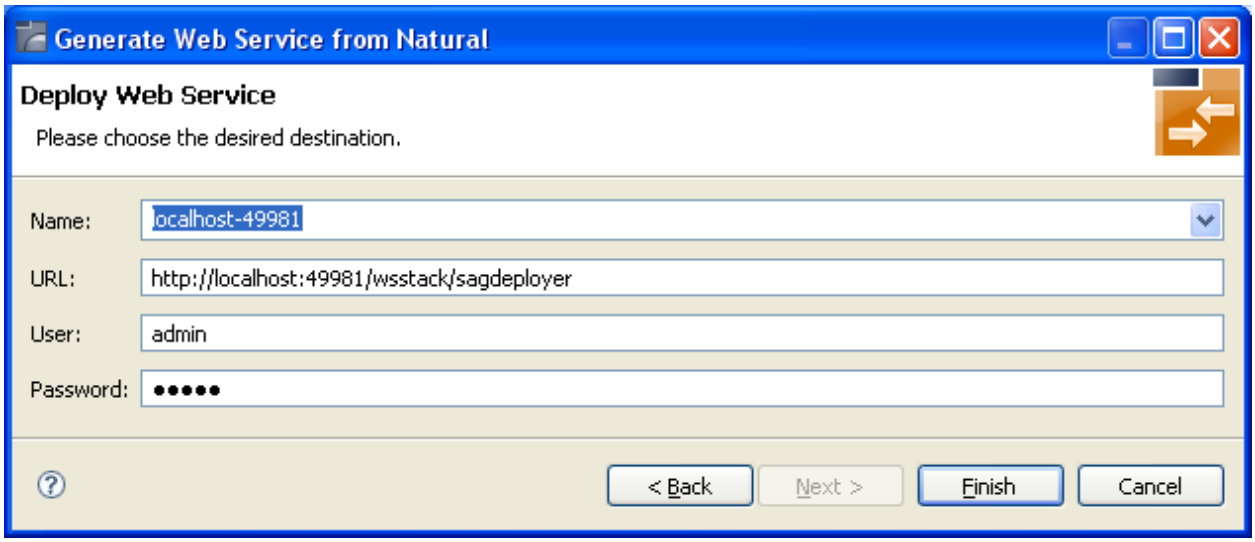
Step 5: Configure the Web Service Client

On the next wizard page you can specify the name of the Web service and service URL (which depends of course on where it is deployed) and whether to deploy the Web service in a WS-stack runtime and register it in a CentraSite instance. You can change the proposed default values according to your needs.



Step 6: Deploy the Web Service

If you chose to deploy the Web service, a wizard page is displayed where you can select the target WS-stack instance for the deployment. In the workspace preferences (**Window > Preferences > Software AG > WS-Stack > Deployment**) you can define a list of additional WS-stack deployment targets. Select one of these targets.



Note: Deployment in WS-stack requires authentication. The default authentication credentials are preconfigured (user: admin, password: axis2). If the WS-stack installation was secured differently, contact you administrator for the current credentials. See also *Deploying and Undeploying Web Service Archives* in section *Software AG Designer Plug-in* of the *Web Services Stack* documentation.

If you chose to register the Web service in a CentraSite, additional Wizard pages will be displayed that guide you through the Web service registration process. See the WS-stack documentation, section *Configuration > Eclipse Plug-in > Registering a Web Service Package in CentraSite* for more information.

Generation Result

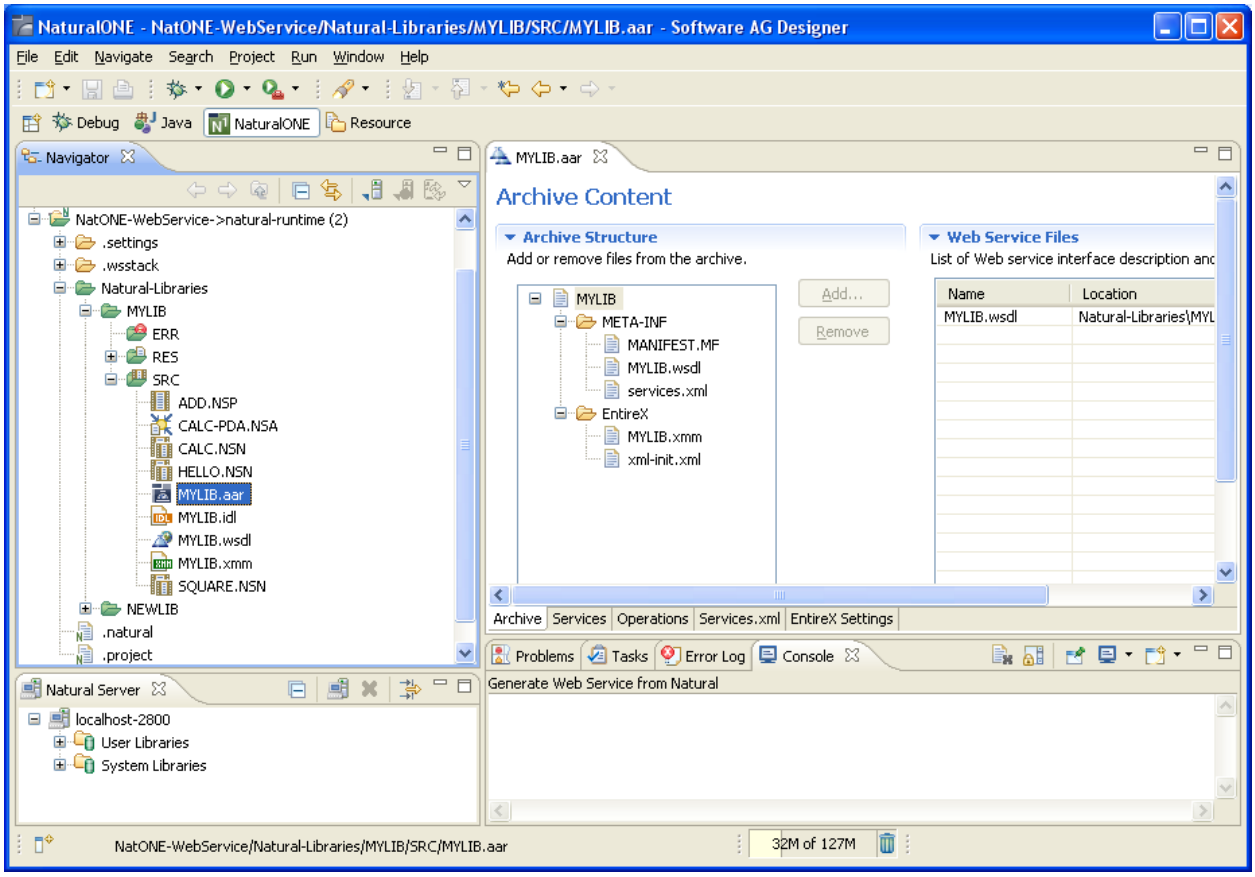
When the wizard has finished successfully, you will see additional artifacts in your Eclipse project:

- **.aar file**
  This is the generated Web service package (containing the metadata files that define the Web service).
- **.wsdl file**
  This is the Web service description language file that describes the basic interface of the service to be used by Web service client applications.

  Note: The actual WSDL of the deployed Web service might differ from this file (containing the actual service URL, transport bindings and policies that are in effect according to the service configuration in the target WS-stack). You can query the effective WSDL of the deployed service using the service's URL, appended with "?wsdl" (without quotes).

- **.idl file**
  The Software AG IDL file that describes the RPC interface of the Natural server component that implements the service's business logic. The IDL file is opened with the IDL Editor. See *Software AG IDL File* in the IDL Editor documentation.
- **.cvm file (optional)**
  This server mapping file completes the IDL file with a mapping from the programming-language-neutral parameter definition in the IDL file to the parameters and data types expected by the Natural subprograms (CALLNATs). Always keep the server mapping file in the same folder as its related IDL file. See *Server Mapping Files for Natural*.
- **.xmm file**
  This is a mapping file that specifies the mapping of XML/SOAP to EntireX RPC and back. You can edit this file with the EntireX XML Mapping Editor and customize it for subsequent Web service re-generation.



For more information see the NaturalONE documentation.

Preferences

Use the preference page for the IDL Extractor for Natural to manage the default values relevant for step *Step 3: Select the Natural Subprograms*. See Preferences.

2 Using the Web Services Wrapper for Natural in Command-line Mode

Command-line mode is currently not supported.

II Natural to XML Schema Type Mapping

Natural to IDL Mapping
XML Structures and IDL-XML Mapping
WSDL to IDL Mapping

3 Natural to IDL Mapping

This chapter describes how Natural data types are mapped to Software AG IDL files by the Software AG IDL Extractor for Natural and covers the following topics:

For more information on Natural syntax, refer to the Natural documentation.

Mapping Natural Data Types to Software AG IDL

The IDL Extractor for Natural maps the following subset of Natural data types to Software AG IDL data types.

The following metasymbols and informal terms are used for the IDL in the table below.

- The metasymbols "[" and "]" surround optional lexical entities
- The informal terms *n* and *m* are sequences of numeric characters, for example 123.

Natural Data Type	Software AG IDL Data Type	Description
A*number*	A*n*	Alphanumeric
A DYNAMIC	AV*n*	Alphanumeric variable length
B*number*	B*number*	Binary
B DYNAMIC	BV	Binary variable length
C	not supported	
D	D	Date
F4	F4	Floating point (small)
F8	F8	Floating point (large)
I1	I1	Integer (small)
I2	I2	Integer (medium)
I4	I4	Integer (large)
L	L	Logical
N*number*[.*number*]	N*number*[.*number*]	Unpacked decimal
P*number*[.*number*]	P*number*[.*number*]	Packed decimal
T	T	Time
Unumber	Unumber	Unicode
U DYNAMIC	UV	Unicode variable length

Redesigning the Extracted Interface

The IDL Extractor for Natural allows you to design the interface to your Natural subprogram (`CALLNAT`). This includes

- *Extracting Multiple Interfaces*
- *Extracting Natural* `REDEFINES`
- *Extracting IDL Directions (*`IN`,`OUT`,`INOUT`*)*
- *Setting Natural Parameters to Constants*
- *Suppressing Natural Parameters*

See *Step 6: Redesign the Interface for Natural Subprograms (Optional)* for more information.

Extracting the IDL Library Name

The Natural library from where Natural programs are extracted is used as the IDL library name. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Extracting the IDL Program Name

The Natural program name is used as the IDL program name, see `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Extracting IDL Parameter Names

For **source extractions**, Natural parameter names are kept and used as IDL parameters, see `simple-parameter-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation and `group-parameter-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

For **object extractions**, Natural programs must be compiled (cataloged) with the compiler option `SYMGEN=ON` to keep original Natural parameter names. Otherwise, generic parameter names are generated (`PARAMETER-1`, `PARAMETER-2`, etc.).

In **Select Natural Sources** (see *Step 3: Select the Natural Subprograms from NaturalONE Project* if you are extracting from NaturalONE projects or ***Step 3: Select the Natural Subprograms*** if you are extracting from a Natural RPC environment), you can choose special characters (`$`, `#`, `&`, `@`, `/`) in

Natural parameter names to be replaced by underscores. See *Rules for Coding Group and Parameter Names* under *Software AG IDL File* in the IDL Editor documentation.

Extracting IDL Directions (IN,OUT,INOUT)

In most Natural subprograms, parameters have no specification for a direction. Missing a direction is unproblematic for local calls. For remote RPC calls, however, specifying the direction helps to reduce data sizes.

If you redesign the interface, you can define IDL directions in *Step 6: Redesign the Interface for Natural Subprograms (Optional)* using the mapping operations **Map to In**, **Map to Out**, **Map to InOut**.

Otherwise, IDL directions can be inserted at top-level parameters (level 1) using a Natural line comment in the Natural subprogram (`CALLNAT`) interface definition (`DEFINE DATA PARAMETER`), example:

```
DEFINE DATA PARAMETER
1 #IN-FIELD-1            (P9)  /* IN
1 #OUT-FIELD-1           (P9)  /* OUT
1 #INOUT-FIELD-1         (P9)  /* INOUT
1 #INOUT-FIELD-2         (P9)
1 #IN-GROUP-1                  /* IN
  2 #IN-GROUP-FIELD-1    (A10)
1 #OUT-GROUP-1                 /* OUT
  2 #OUT-GROUP-FIELD-1   (A10)
1 #INOUT-GROUP-1               /* INOUT
  2 #INOUT-GROUP-FIELD-1 (A10)
1 #INOUT-GROUP-2
  2 #INOUT-GROUP-FIELD-2 (A10)
1 #INOUT-GROUP-3
  2 #INOUT-GROUP-FIELD-3 (A10) /* OUT
END-DEFINE
```

If no direction is specified (such as in `#INOUT-FIELD-2` and `#INOUT-GROUP-2` in the example above), the default direction INOUT applies.

Specifications on a level greater than 1 (such as `#INOUT-GROUP-FIELD-3` in the example above) are ignored. Note that in IDL directions are specified on top-level fields (level 1), see `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Specifications on IDL directions are only considered when extracting from a source. If you are extracting from an object (compiled), as described in *Step 5: Select Natural Subprograms from RPC Environment*, the default direction `INOUT` always applies.

Extracting Natural REDEFINES

A redefinition is a second parameter layout of the same memory portion. The parameter `#BASE-FIELD` is redefined by the fields `FILLER-1` thru `R-P3-01`.

```
DEFINE DATA PARAMETER
1 #BASE-FIELD            (A161)
1 REDEFINE #BASE-FIELD
  2 FILLER-1             (A4)
  2 FILLER-2             (A60)
  2 R-P1-01              (A1)
  2 R-P2-01              (A10)
  2 R-P3-01              (I4)
END-DEFINE
```

With the extractor wizard you can select a single redefine path for IDL usage (here the fields `FILLER-1` thru `R-P3-01`) if you redesign the interface. See *An Example for Extracting Natural `REDEFINES`* and *Step 6: Redesign the Interface for Natural Subprograms (Optional)*.

Extracting Multiple Interfaces

Legacy Natural subprograms often implement multiple functions in a single Natural subprogram. The function executed is often controlled by a so-called function code or operation-code field. See *An Example for Extracting Multiple Interfaces*.

With the extractor wizard you can extract the functions from the server as separate interfaces (IDL programs). In this way, the legacy server with a single physical interface can be

- turned into a web service with operations, where the legacy functions match operations.
- called with an object-oriented wrapper such as the *Java Wrapper*, the *.NET Wrapper* or the *DCOM Wrapper*, where the legacy functions match methods.

Note that every function in the Natural subprogram may have a different interface described with `REDEFINE` syntax. Therefore, multiple interface extraction is often combined with *Extracting Natural `REDEFINES`*.

For more information, see *Step 6: Redesign the Interface for Natural Subprograms (Optional)*.

Extracting Natural Arrays, Groups, X-Arrays and Variable Arrays

This section describes IDL mapping for Natural arrays and groups:

Arrays and Groups with Fixed upper Limits

Ordinary Natural arrays and groups with fixed/bound upper limits are mapped to Software AG IDL fixed-bound-array definitions, see *array-definition* under *Software AG IDL Grammar* in the IDL Editor documentation.

Natural syntax example:

```
DEFINE DATA PARAMETER
1 #ARRAY1   (I4/1:10) /* lower bound is fixed at 1, upper bound is 10
1 #ARRAY2   (I4/10)   /* shortcut for (A5/1:10)
1 #GROUP1   (10)
  2 #FIELD1 (I2)
  2 #FIELD2 (A10)
. . .
END-DEFINE
```

X-Arrays and X-Groups

For X-arrays (eXtensible arrays) the number of occurrences is flexible at runtime. The number of occurences can be resized, i.e. increased or reduced. It is defined by specifying an asterisk (*) for index bounds.

Natural syntax example:

```
DEFINE DATA LOCAL
1 #X-ARRAY1 (A5/1:*) /* lower bound is fixed, upper bound is variable
1 #X-ARRAY2 (A5/*)   /* shortcut for (A5/1:*)
. . .
END-DEFINE
```

Natural X-arrays are mapped to Software AG IDL unbounded-array definitions, see `array-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Natural X-arrays with variable lower bounds are *not* supported by Software AG RPC technology, example:

```
DEFINE DATA PARAMETER
1 #X-ARRAY1 (A5/*:10) /* lower bound is variable, upper bound is fixed
. . .
END-DEFINE
```

Variable Arrays and Variable Groups

In a Natural parameter data area (PDA), you can specify an array or group with a variable number of occurrences. This is done with the index notation `1:V`. The maximum number of occurrences for such an array is either passed to the subprogram using an extra parameter such as `#ARRAY1-LIMIT` (see example below), or it can be accessed using the system variable `*OCCURRENCE`.

Natural syntax example:

```
DEFINE DATA PARAMETER
1 #ARRAY1-LIMIT (I4) /* extra parameter to pass the upper limit
1 #ARRAY1       (I4/1:V)
. . .
END-DEFINE
```

Natural variable arrays are mapped to Software AG IDL unbounded-array definitions, see `array-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation.

If the Natural server program uses a separate parameter such as `#ARRAY1-LIMIT` (see the example above) instead of `*OCCURRENCE` to determine the upper bound limit, it is required to extract this extra parameter, too. During runtime, it is also required to specify the number of occurences in a calling RPC client.

In a Natural server program, Natural variable arrays

- cannot be resized for direction INOUT, which means you can only reply the same number of occurrences to the RPC client.
- cannot be used for direction OUT either, because they cannot be created (instantiated). You may get error 20050031 during extraction.

Arrays and Groups with Mixed Dimensions (X, Variable and Fixed)

Natural arrays and groups with a mixture of fixed variable and eXtensible dimensions are *not* supported by Software AG RPC technolgy, example:

```
DEFINE DATA PARAMETER
1 #ARRAY1   (I4/1:10,1:*) /* first dimension fixed and second eXtensible
1 #ARRAY2   (I4/1:10,1:V) /* first dimension fixed and second variable
1 #ARRAY3   (I4/1:V,1:*)  /* first dimension variable and second eXtensible
. . .
END-DEFINE
```

Extracting Natural Structure Information (IDL Levels)

Source Extractions

Natural levels are always kept. This means that the structure in the extracted IDL is the same as in the original Natural program.

Object Extractions

- **UNIX or Windows**
  In UNIX or Windows RPC environments, Natural levels are *not* kept. The IDL is extracted in a flat way, where
  - all IDL parameters are at level 1;
  - all Natural groups are removed;
  - Natural fields within groups using repetition (`PERIODIC GROUPS`) are mapped to IDL arrays;
  - the dimension of Natural arrays within groups using repetion (`PERIODIC GROUPS`) is increased in the IDL. For example, a one-dimensional array may become a two-dimensional or three-dimensional IDL array depending on the dimension of the group;
- **z/OS**
  In z/OS RPC environments, the Natural programs must be compiled (cataloged) with the compiler option `SYMGEN=ON` to keep Natural levels, otherwise flat extraction is carried out.

Extracting Parameters defined with OPTIONAL

For a parameter defined without `OPTIONAL`, a value must be passed from the invoking Natural object, i.e. the caller.

For a parameter defined with `OPTIONAL`, a value can, but need not be passed from the invoking Natural object to this parameter. With the `SPECIFIED` option, a Natural server can find out at runtime whether an optional parameter has been defined or not.

The IDL Extractor for Natural ignores the `OPTIONAL` specification, i.e. the parameter is extracted as without the `OPTIONAL` specification. See the *Natural Documentation* for more information.

EntireX RPC technology does *not* support optional IDL parameters. Using pure Natural RPC (Natural client to Natural server), Natural optional parameters are supported.

Setting Natural Parameters to Constants

Setting parameters to constant values and suppressing them in the IDL is part of the redesign process of the extracted interface. This keeps the IDL client interface lean. See *An Example for Set Constant*.

EntireX and Natural RPC make sure the constant value is passed to the Natural server during runtime. No data is transferred between the RPC client and the RPC server.

For more information, see *Step 6: Redesign the Interface for Natural Subprograms (Optional)*.

Suppressing Natural Parameters

Hiding or suppressing unneeded parameters in the IDL is part of the redesign process of the extracted interface. This keeps the IDL client interface lean and minimizes the amount of data to be transferred during runtime.

EntireX and Natural RPC make sure to provide low values as input for suppressed parameters to the Natural server called (blank for IDL type `A`, zero for numeric data types such as IDL `I`, `N` and `P`). No data is transferred between an RPC client and the RPC server.

For more information, see *Step 6: Redesign the Interface for Natural Subprograms (Optional)*.

Renaming a Program

Renaming a program to a different name in the IDL is part of the redesign process of the extracted interface. You can adjust the short Natural name to a meaningful longer name for better readability. See *An Example for Extracting Multiple Interfaces* where the original Natural name `CALC` is renamed to IDL names `ADD`, `SUBTRACT`, `MULTIPLY` etc.

EntireX and Natural RPC make sure the original Natural server is called during runtime. For more information, see *Step 6: Redesign the Interface for Natural Subprograms (Optional)*.

4 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.

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

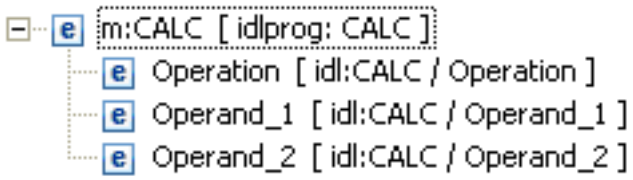
In the element-preferred mode, the IDL is mapped to the incoming XML document:

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

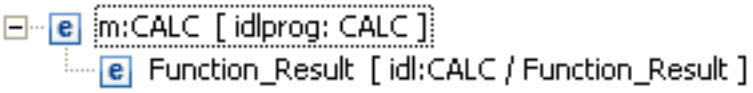
and the outgoing XML document:

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

The corresponding XML structure trees are:



and



Attribute-preferred Mode

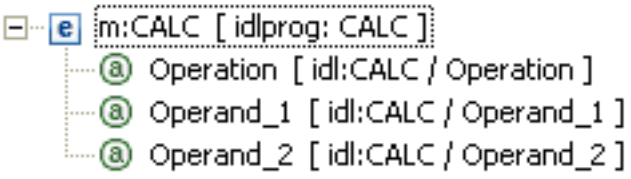
In the attribute-preferred mode, the Software AG IDL above is mapped to the incoming XML document:

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

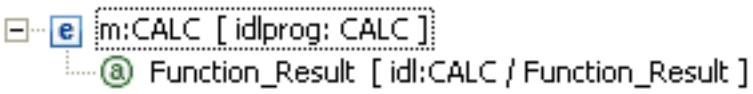
and the outgoing XML document.

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

The corresponding XML structure trees are:



and



SOAP-conformant Mapping

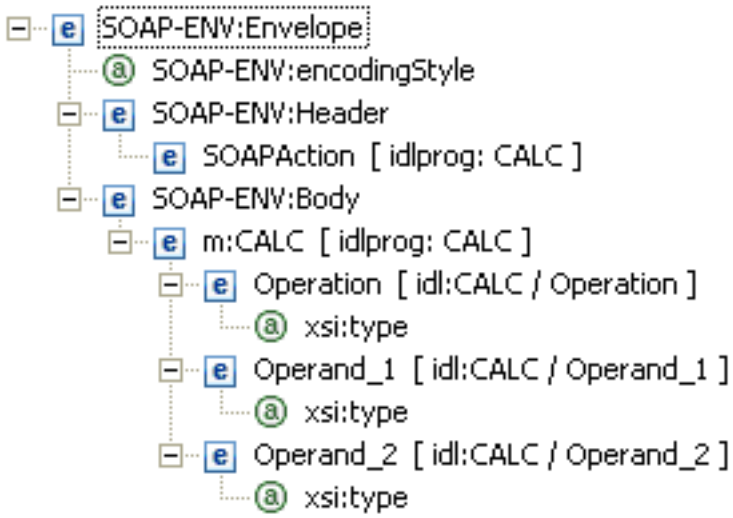
In SOAP-conformant mapping or the SOAP-conformant mode, the above Software AG IDL is mapped to the incoming XML document:

```
<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>
```

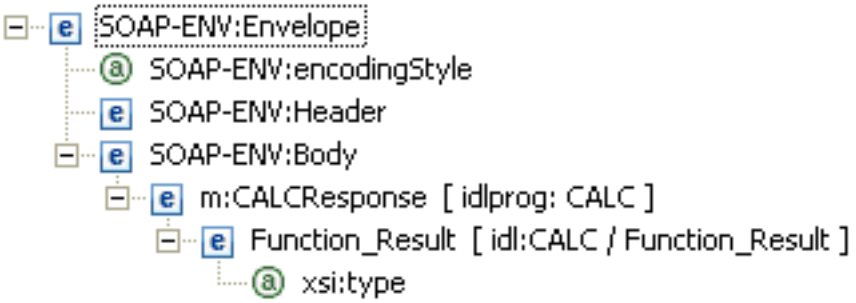
and the outgoing XML document:

```
<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.

5 WSDL to IDL Mapping

Extracting IDL from WSDL Files

The Software AG IDL Extractor for WSDL produces the IDL file and an XML mapping file. The SOAP-binding information is written into the XML mapping file (XMM), for example, the SOAPAction value and the namespace definitions. The two other bindings (HTTP and MIME) only return the IDL file, but no XML mapping file. In this case, a warning dialog is displayed. WSDL files with mixed bindings, including a SOAP binding, also return an XML mapping file, but display the warning message too. The XML mapping and IDL parameter directions depend on the WSDL source file; INERR and OUTERR mapping trees are possible.

Mapping WSDL XML Schema Data Type to Software AG IDL

WSDL / XML Schema	XMM	Software AG IDL
`binary`, `base64Binary`	`binary`	`BV` (or `BV`*n* or `B`*n*) (3)
`hexBinary` (1)	`binary`	`BV` (or `BV`*n* or `B`*n*) (3)
`boolean`	`boolean`	`L`
`date`	`date:yyyy-MM-dd` (2)	`D`
`float`	`float`	`F4`
`double`	`float`	`F8`
`byte`, `unsignedByte`	`integer`	`I1`
`short`, `unsignedShort`	`integer`	`I2`
`int`, `unsignedInt`	`integer`	`I4`
`integer`, `positiveInteger`, `nonPositiveInteger`, `negativeInteger`, `nonNegativeInteger`	`number`	`N29.0`
`decimal`, `number`	`number`	`N22.7`
`long`, `unsignedLong`	`number`	`N19.0`
`time`	`dateTime:HH:mm:ss` (2)	`T`
`dateTime`	`dateTime:yyyy-MM-dd'T'HH:mm:ss` (2)	`T`
`gYearMonth`	`string`	`A8`
`gDay`, `gYear`	`string`	`A11`
`gMonth`	`string`	`A12`
`gMonthDay`	`string`	`A13`
`string` (and all types not listed here)	`string`	`AV` (or `AV`*n* or `A`*n*) (3)

Notes:

1. The `hexBinary` format is not supported by the XML/SOAP Runtime.
2. Edit the `date` and `dateTime` patterns manually to match the formats of the original documents.

   Example: `<myTime xsi:type="xsd:date">11:08:23+01:00</myTime> --> dateTime:HH:mm:ss' +01:00 ' --> T`

   Note: The `+01:00` is not supported by IDL (EntireX RPC protocol).

3. Mapped according to specified transformation rules. See *Step 6: Specify Options for Target Programming Language* in the IDL Extractor for WSDL documentation.

Extracting the Name for the IDL Library

The IDL library name (see `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation) will be used from the value of the name attribute of the tag `<service>`, for example:

```
<definitions ...>
    <service name="LIBRARYNAME">
        <port .../>
    </service>
</definitions>
```

Extracting the Name for the IDL Program

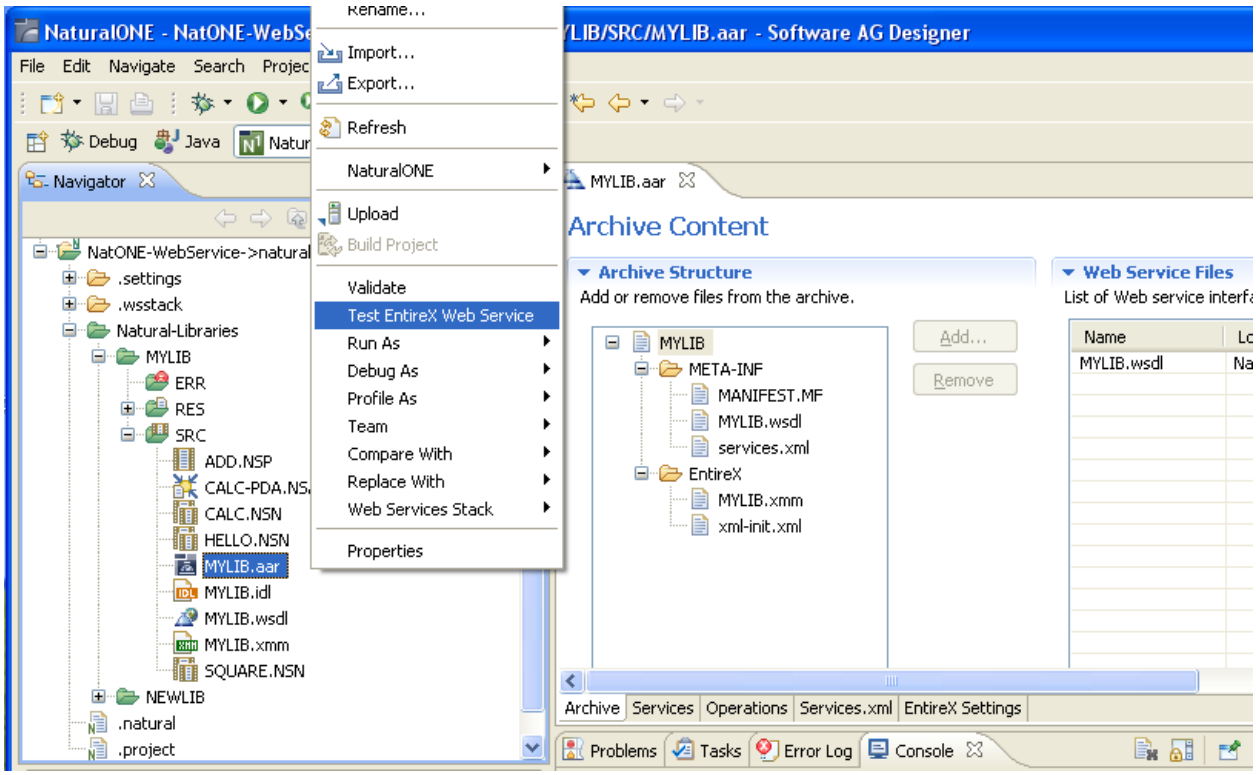
The RPC program name (see `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation) will be used from the value of the name attribute of the tag `<operation>` as child of the tag `<portType>`, for example:

```
<definitions ...>
    <portType name="...">
        <operation name="PROGRAMNAME">
            <input .../>
            <output .../>
        </operation>
    </portType>
</definitions>
```

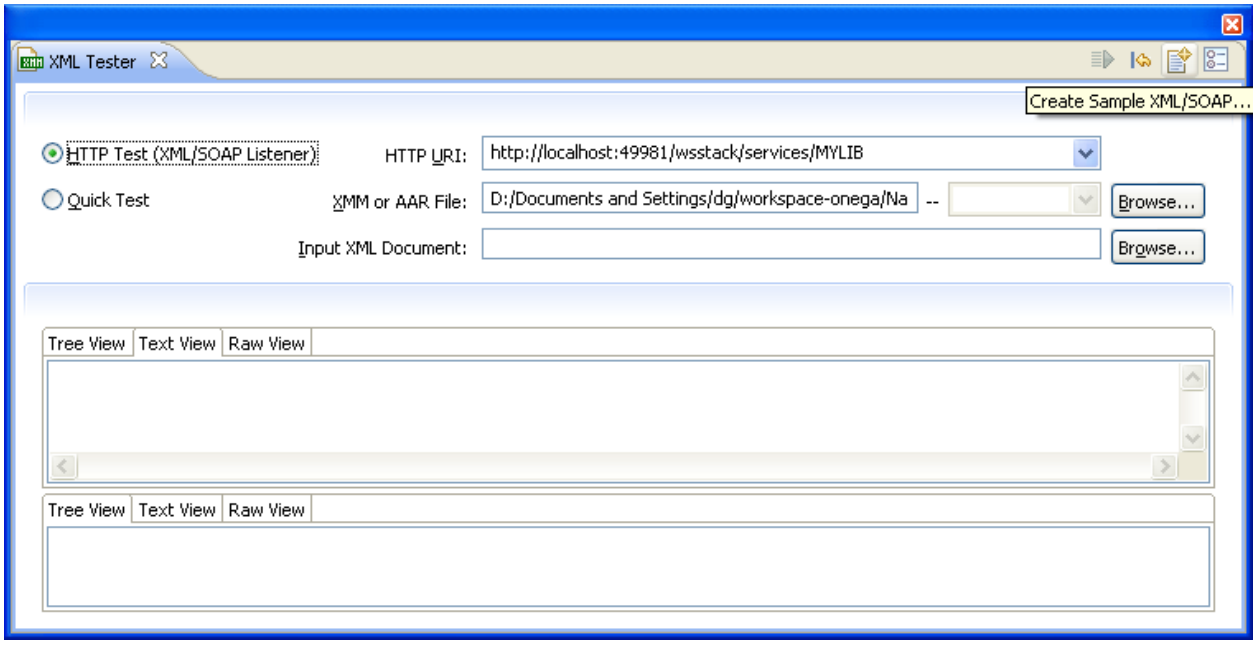
III

6 Writing Applications with the Web Services Wrapper for Natural

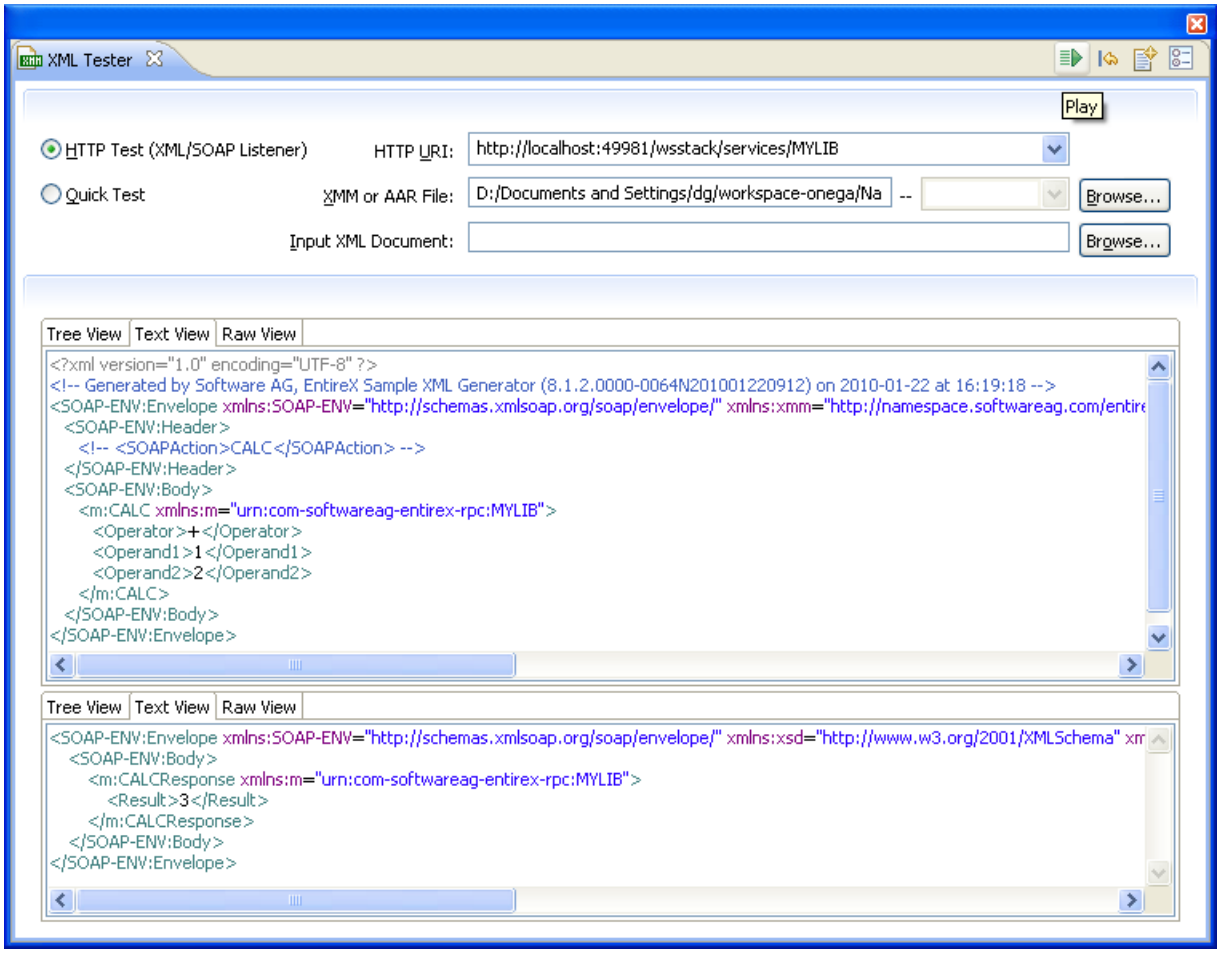
If you have generated and deployed a Web service from one or more Natural subprograms, you can use a variety of Web service application development tools to develop web service client applications, for example Microsoft Visual Studio, Java Frameworks such as Axis2 and others. These are not covered in detail here. However, this chapter describes how to test a generated Web service with the EntireX XML Tester. You can launch the EntireX XML Tester from the context menu of a Web services archive .aar:



On the toolbar of the XML Tester you can initiate the generation of a sample XML/SOAP request message that complies with the interface definition of the service.



Some default values for the request parameters are generated into the sample SOAP request. You can edit them according to your needs and send the message to the Web service, using the **Play** button on the toolbar.



The SOAP response is displayed in the output pane of the XML tester and can be viewed there in different formats.

For more information, see *XML Tester* in the XML/SOAP Wrapper documentation.

7 Delivered Client and Server Examples for Natural

See the readme files of the following examples for more information:

- <Entire>/examples/RPC/basic/example/NaturalServer
- <Entire>/examples/RPC/basic/example/SOAPClient
- <Entire>/examples/RPC/reliable/NaturalServer
- <Entire>/examples/RPC/reliable/SOAPClient