

webMethods EntireX

Software AG IDL Extractor for WSDL

Version 10.9

April 2023

This document applies to webMethods EntireX Version 10.9 and all subsequent releases.

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

Copyright © 1997-2023 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: EXX-EEXXWSDLEXTRACTOR-109-20230403

Table of Contents

1 About this Documentation	1
Document Conventions	2
Online Information and Support	2
Data Protection	3
2 Introduction to the IDL Extractor for WSDL	5
3 Using the IDL Extractor for WSDL	7
Step 1: Start the IDL Extractor for WSDL	8
Step 2: Select a Source	9
Step 3a: Specify CentraSite Location	10
Step 3b: Specify UDDI Server	11
Step 3c: Specify WSDL File	11
Step 3d: Specify WSDL File URL	12
Step 4: Specify Output Files	13
Step 5: Specify Broker Settings	13
Step 6: Specify Options for Target Programming Language	14
Extraction Result	18
4 Using the IDL Extractor for WSDL in Command-line Mode	19
5 WSDL to IDL Mapping	21
Extracting IDL from WSDL Files	22
Mapping WSDL XML Schema Data Type to Software AG IDL	22
Extracting the Name for the IDL Library	23
Extracting the Name for the IDL Program	23
6 Writing Web Service Client Applications	25
Web Service Clients	26
Configuring Advanced Web Service Clients	27
Example: Setting up an EntireX Client to Consume a Secured Web Service	28

1 About this Documentation

▪ Document Conventions	2
▪ Online Information and Support	2
▪ Data Protection	3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Monospace font	Identifies service names and locations in the format <code>folder.subfolder.service</code> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the symbol.
[]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at <https://documentation.softwareag.com>.

In addition, you can also access the cloud product documentation via <https://www.software-ag.cloud>. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at <https://knowledge.softwareag.com>.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at <https://tech-community.softwareag.com>. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at <https://github.com/softwareag> and <https://hub.docker.com/publishers/softwareag> and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at <https://empower.softwareag.com>. Many services on this portal require that you have an account. If you do not yet have one, you can request it at <https://empower.softwareag.com/register>. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Introduction to the IDL Extractor for WSDL

The Software AG IDL Extractor for WSDL is a wizard that generates web service client artifacts from a WSDL file. With these artifacts, EntireX RPC client applications can access external web services.

With the IDL Extractor for WSDL you can import from the following sources:

- local file
- URL (http, ftp etc.)
- UDDI registry
- CentraSite registry

The 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 IDL Extractor for WSDL generates for each service definition in the WSDL file (according to the generation rules) IDL/XMM files named *wsdl-service-name.idl* and *wsdl-service-name.xmm*. The XML mapping and IDL parameter directions depend on the WSDL source file; INERR and OUTERR mapping trees are possible.

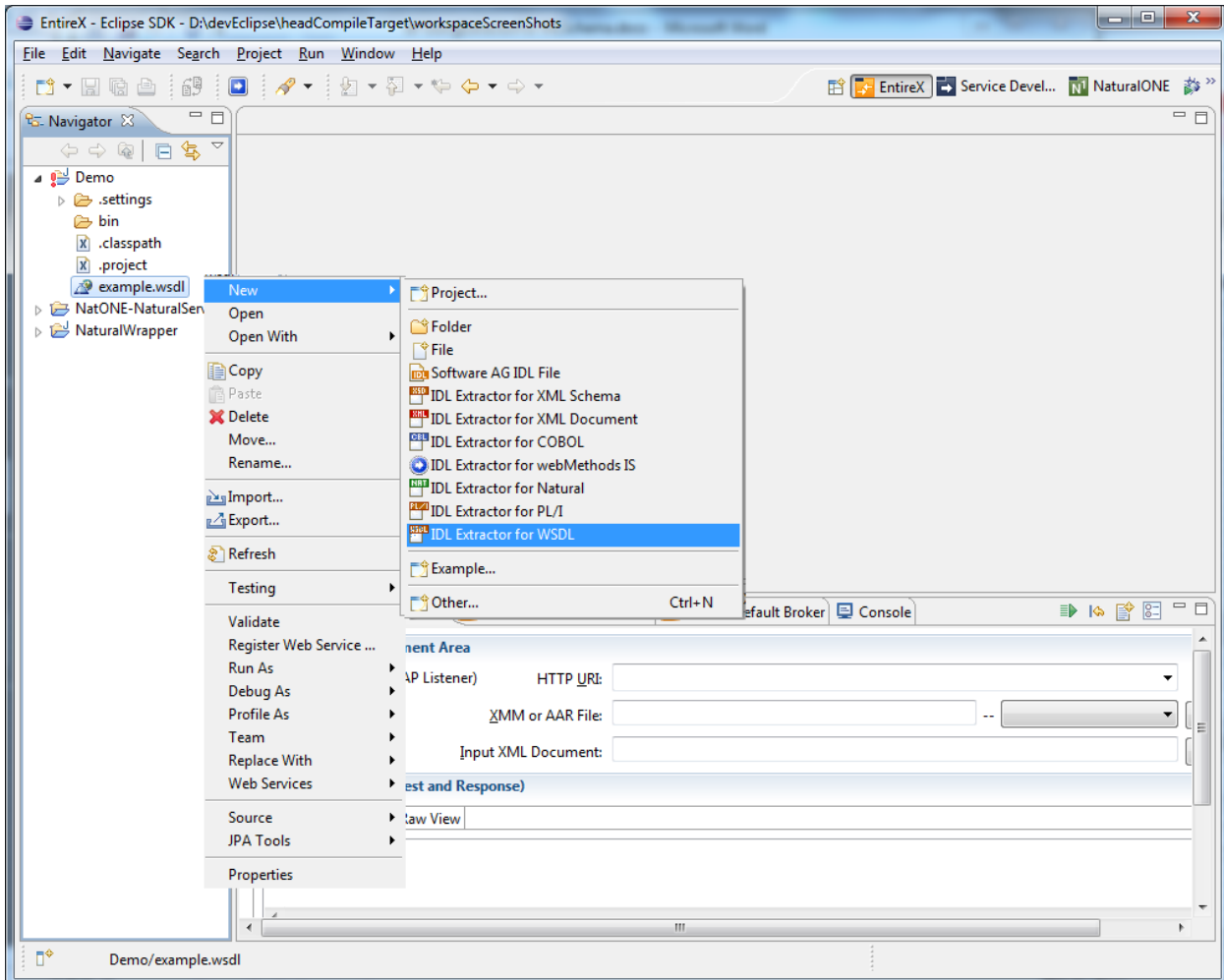
3 Using the IDL Extractor for WSDL

- Step 1: Start the IDL Extractor for WSDL 8
- Step 2: Select a Source 9
- Step 3a: Specify CentraSite Location 10
- Step 3b: Specify UDDI Server 11
- Step 3c: Specify WSDL File 11
- Step 3d: Specify WSDL File URL 12
- Step 4: Specify Output Files 13
- Step 5: Specify Broker Settings 13
- Step 6: Specify Options for Target Programming Language 14
- Extraction Result 18

Caution: If you modify the imported IDL file, do this only in the XML Mapping Editor to ensure the correct dependencies between the IDL and the related XMM file.

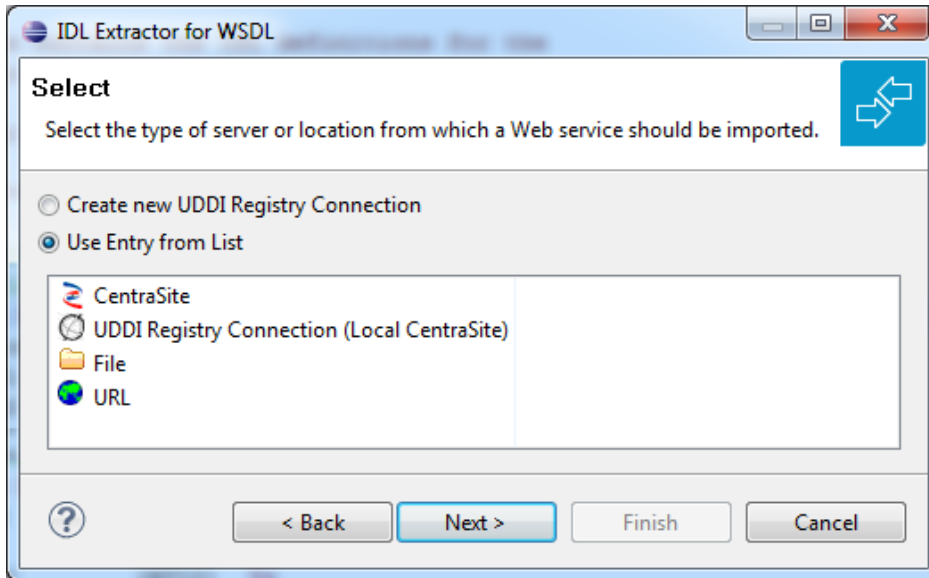
Step 1: Start the IDL Extractor for WSDL

Start the IDL Extractor for WSDL as any other Eclipse New wizard:



Step 2: Select a Source

Depending on the location of the WSDL document to analyze, choose one of the following options:



For **File**, **URL**, **CentraSite** and already defined UDDI registry connections, check the radio button **Use Entry from List**. To define additional connections to UDDI server, check the radio button **Create new UDDI Registry Connection**. UDDI registry connection are defined in the preferences; see also *UDDI Registration*.

Notes:

1. The supported URL protocols are FILE, FTP, HTTP, HTTPS and JAR, for example

```
http://host/myservice?WSDL
```

2. If the connection is over HTTPS, you need to set up HTTPS in Designer:

Define `trustStore` in Designer, for example with the following lines in file *eclipse.ini*

```
-Djavax.net.ssl.trustStore=<path to keystore>
-Djavax.net.ssl.trustStorePassword=<keystore password>
```

If hostname verification for certification is to be disabled, also add the line:

```
-Dcom.softwareag.entirex.ssl.hostnameverify=false
```

■ **CentraSite**

If the WSDL source file to be extracted is available in CentraSite, continue with [Step 3a: Specify CentraSite Location](#). If the connection is over HTTPS, see the [Note](#) below.

■ **UDDI Registry Connection**

If the WSDL source file to be extracted is accessible using a UDDI registry connection (UDDI server), continue with [Step 3b: Specify UDDI Server](#). If the connection is over HTTPS, see the [Note](#) below.

■ **File**

If the WSDL source file to be extracted is available in your workspace and you have selected it, the file location will be entered in the wizard automatically in the next [Step 3c: Specify WSDL File](#).

■ **URL**

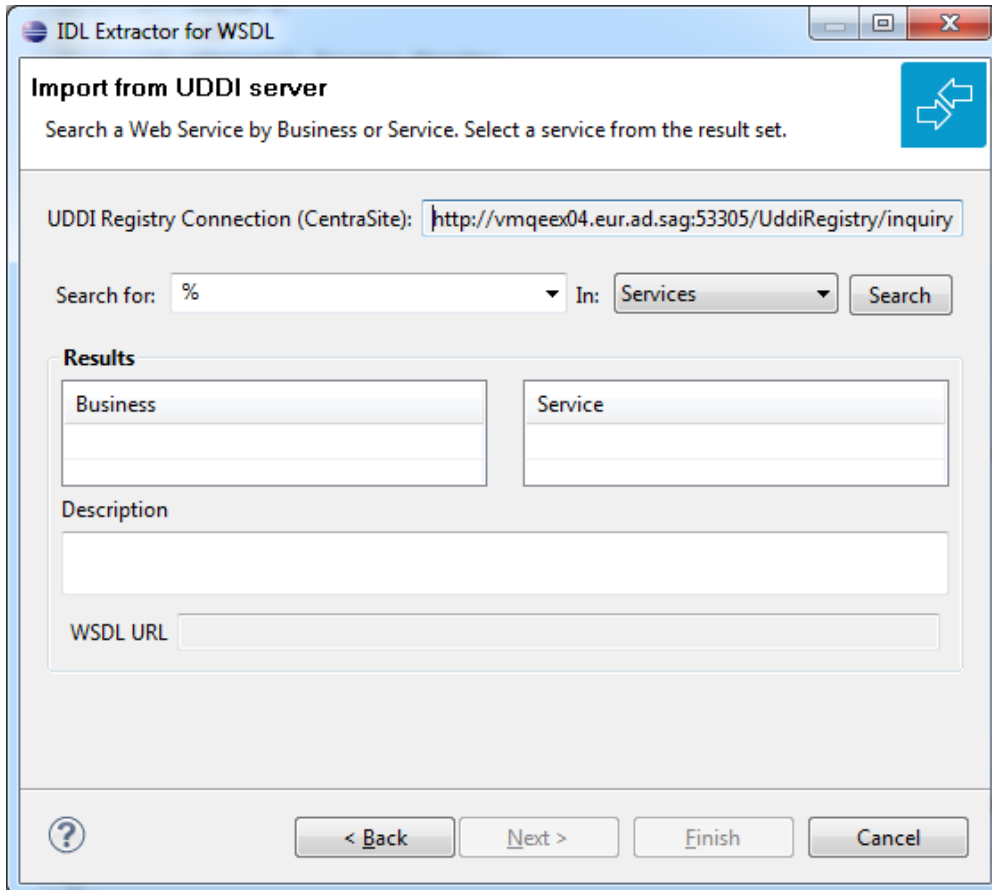
Continue with [Step 3d: Specify WSDL File URL](#). If the connection is over HTTPS, see the [Note](#) below.

Step 3a: Specify CentraSite Location

When importing from CentraSite, the following screen is displayed:

Step 3b: Specify UDDI Server

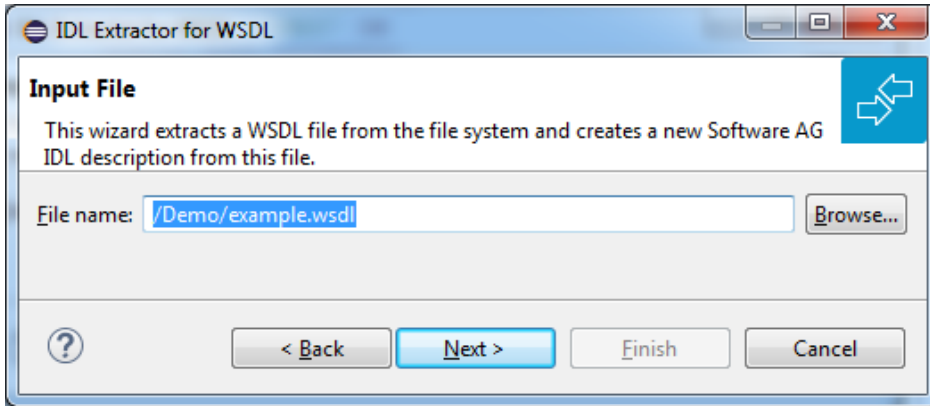
When importing from a UDDI server, the following screen is displayed:



You can search for Businesses or Services. You can restrict your search using % as a wildcard, for example ex%. The search returns a list of service providers and their respective services. Select one service and continue with **Next**.

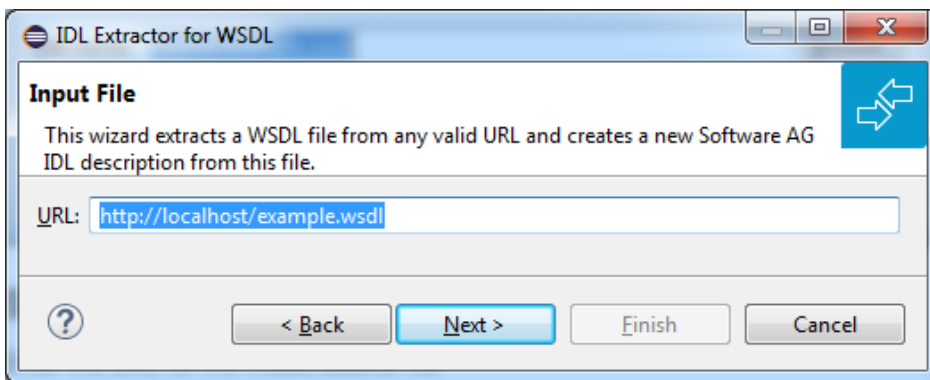
Step 3c: Specify WSDL File

If you selected the WSDL source file before you started the wizard, the file location is already present. Enter or browse for the WSDL source file. Continue with [Step 4: Specify Output Files](#).

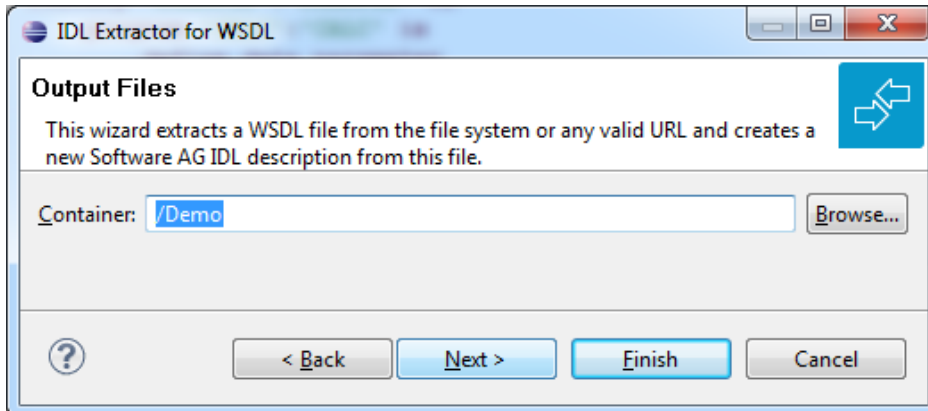


Step 3d: Specify WSDL File URL

Enter the URL for the WSDL source file.



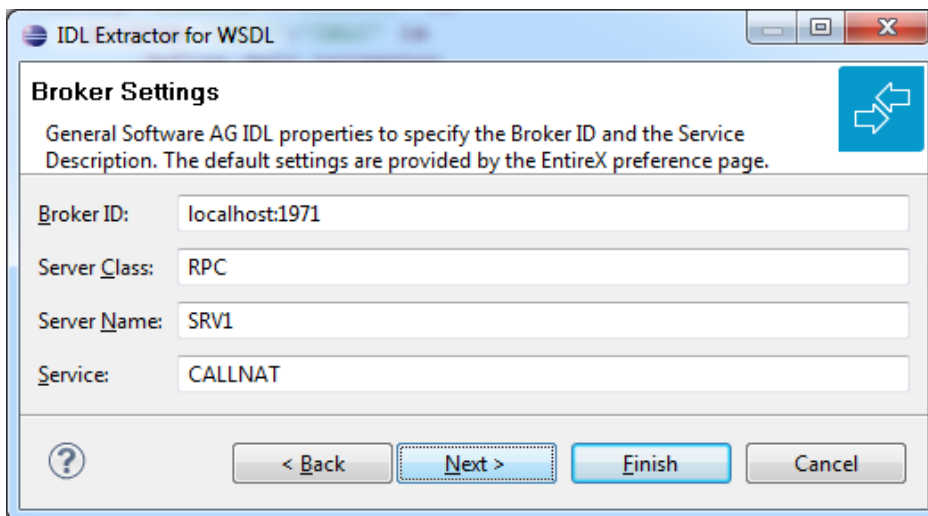
Step 4: Specify Output Files



Select the container where the IDL and XMM files will be stored.

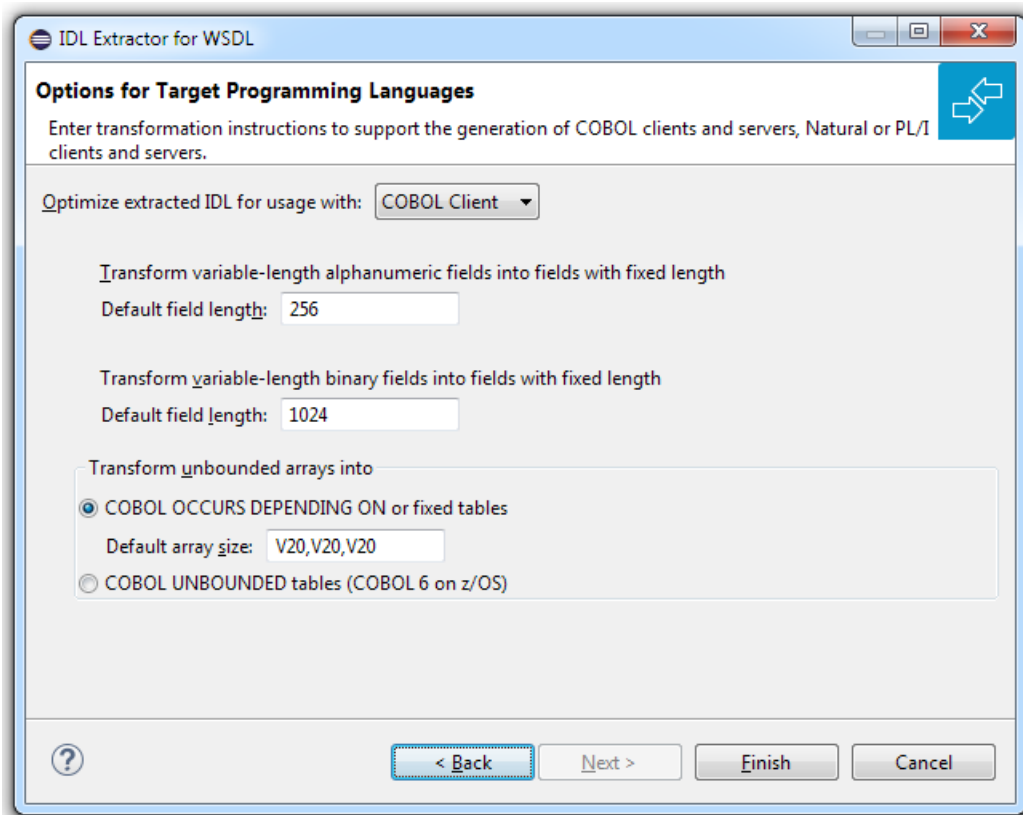
Step 5: Specify Broker Settings

In the following screen you can optionally modify Broker settings.



Step 6: Specify Options for Target Programming Language

The **Options for Target Programming Language** page allows you to specify transformation rules for variable-length fields and unbounded arrays. This is required if you later use the COBOL Wrapper or PL/I Wrapper with the extracted IDL - otherwise COBOL or PL/I wrapping is not possible. If you later use the Natural Wrapper, transformation rules are optional. If they are used, the interface from a Natural point of view is more legacy-like, easier to use but with restrictions.



With the transformation rules, you define default (maximum) lengths and sizes depending on the originating data types on the XML side. If you need different (maximum) lengths and sizes for fields with the same data type, use the XML Mapping Editor. See *Using the XML Mapping Editor*

Caution: If you modify the imported IDL file, do this only in the XML Mapping Editor to ensure the correct dependencies between the IDL and the related XMM file.

Depending on the target programming language of your scenario, the available/possible transformation rules differ. Use the combo-box and choose the target programming language:

- COBOL Client
- COBOL Server

- Natural
- PL/I Client
- PL/I Server
- Other

COBOL Client

For generation of clients with the COBOL Wrapper.

Variable-length fields with unlimited number of elements are not directly supported by COBOL. Unbounded arrays can be transformed into fixed or unbounded tables (use only for COBOL 6 on z/OS). There are two possibilities to specify options:

- **Transform Variable-length Fields into Fixed-length COBOL Fields and Unbounded Arrays into Fixed or Unbounded Tables**

Variable-length fields on the XML side are mapped to fixed-length COBOL data items, that is, they will always be padded (alphanumeric with trailing blanks; binary with x00). Unbounded arrays on the XML side can be mapped to fixed-size COBOL tables, see *COBOL Tables with Fixed Size*. This means they will always be filled up to the maximum number of elements. To use this possibility, enter the length or size to define the restriction, for example 256, 1024 or 20. In case of COBOL 6 on z/OS they can be also transformed into unbounded tables.

- **Limit Variable-length Fields and Unbounded Arrays to a Maximum**

For variable-length fields, EntireX provides a possibility to transform them into variable-length fields with a maximum length. See *IDL Data Types*, AVnumber and BVnumber under column Type and Length. In this case the variable-length fields are also mapped to fixed-length COBOL data items, but they will be trimmed (alphanumeric with blank, binary with x00) on the COBOL side. Unbounded arrays with a maximum are directly supported in COBOL in the form of COBOL tables with the OCCURS DEPENDING ON clause, see *COBOL Tables with Variable Size - DEPENDING ON Clause*. Only filled elements are transferred. In this case the RPC message size is reduced compared with the alternative *Transform to Fixed-length COBOL Fields and Tables* above. To use this possibility, enter a leading V-character before the limited length or limited size of unbounded arrays, such as V256, V1024 or V20.

COBOL Server

For generation of servers with the COBOL Wrapper.

Variable-length fields and unbounded arrays with unlimited number of elements are not directly supported by COBOL. There are two possibilities to specify options:

- **Transform to Fixed-length COBOL Fields and Tables**

Variable-length fields on the XML side are mapped to fixed-length COBOL data items, that is, they will always be padded (alphanumeric with trailing blanks; binary with x00). Unbounded arrays on the XML side are mapped to fixed-size COBOL tables, see *COBOL Tables with Fixed*

Size. This means they will always be filled up to the maximum number of elements. To use this possibility, enter the length or size to define the restriction, for example 256, 1024 or 20.

■ **Limit Variable-length Fields and Unbounded Arrays to a Maximum**

For variable-length fields, EntireX provides a possibility to transform them into variable-length fields with a maximum length. See *IDL Data Types*, AVnumber and BVnumber under column Type and Length. In this case the variable-length fields are also mapped to fixed-length COBOL data items, but they will be trimmed (alphanumeric with blank, binary with x00) on the COBOL side. Unbounded arrays with a maximum are directly supported in COBOL in the form of COBOL tables with the OCCURS DEPENDING ON clause, see *COBOL Tables with Variable Size - DEPENDING ON Clause*. Only filled elements are transferred. In this case the RPC message size is reduced compared with the alternative *Transform to Fixed-length COBOL Fields and Tables* above. To use this possibility, enter a leading V-character before the limited length or limited size of unbounded arrays, such as V256, V1024 or V20.

Natural

For generation of clients and servers with the Natural Wrapper.

Variable-length fields and unbounded arrays with unlimited number of elements are directly supported by Natural. As an alternative, EntireX also provides the possibility to transform to a more legacy-like interface with fixed length.

■ **Transform to Fixed-length Fields and Fixed-size Arrays on the Natural Side**

Variable-length fields on the XML side are mapped to fixed-length Natural data types, that is, they will always be padded (alphanumeric with trailing blanks; binary with x00). Unbounded arrays on the XML side are mapped to fixed-length Natural arrays, that is, they will always be filled up to the maximum number of elements. Using this possibility you benefit from easier and simpler Natural programming. To use this possibility, check the check boxes and enter the restricted length for variable-length alphanumeric fields, such as 253, variable-length binary fields such as 126, and the restricted size, for example 20,20,20 for unbounded arrays.

■ **Transform to Variable-length Fields and Variable-size Arrays on the Natural Side**

Variable-length fields on the XML side are mapped to Natural DYNAMIC data types. No padding occurs on the Natural side. Unbounded arrays on the XML side are mapped to Natural X-Arrays. Only filled elements are transferred. In this case the RPC message size is reduced compared with the alternative *Transform to Fixed-length Fields and Fixed-size Arrays on the Natural Side* above. To use this possibility, clear the check boxes.

PL/I Client

For generation of clients with the PL/I Wrapper. The following possibilities exist in scenarios with PL/I clients:

- **Transform to Fixed-length Fields and Arrays**

Variable-length fields on the XML side are mapped to fixed-length PL/I data items, that is, they will always be padded (alphanumeric with trailing blanks; binary with x00). Unbounded arrays on the XML side are mapped to fixed-size PL/I arrays, see *Arrays* under *PL/I to IDL Mapping*. This means they will always be filled up to the maximum number of elements. To use this possibility, enter the length or size to define the restriction, for example 256, 1024 or 20.

- **Limit Variable-length Fields to a Maximum**

As an alternative, variable-length fields can be mapped to PL/I data type with the attribute `VARYING`. See also *IDL Data Types* `AVnumber` and `BVnumber` under *column Type and Length*. In this case no padding occurs on the PL/I side. To use this possibility, enter a leading V-character before the limited length, such as `V256` or `V1024`.



Note: This alternative does not exist for unbounded arrays.

PL/I Server

For generation of servers with the PL/I Wrapper. The following possibilities exist in scenarios with PL/I servers:

- **Transform to Fixed-length Fields and Arrays**

Variable-length fields on the XML side are mapped to fixed-length PL/I data items, that is, they will always be padded (alphanumeric with trailing blanks; binary with x00). Unbounded arrays on the XML side are mapped to fixed-size PL/I arrays, see *Arrays* under *PL/I to IDL Mapping*. This means they will always be filled up to the maximum number of elements. To use this possibility, enter the length or size to define the restriction, for example 256, 1024 or 20.

- **Limit Variable-length Fields to a Maximum**

As an alternative, variable-length fields can be mapped to PL/I data type with the attribute `VARYING`. See also *IDL Data Types*, `AVnumber` and `BVnumber` under *column Type and Length*. In this case no padding occurs on the PL/I side. To use this possibility, enter a leading V-character before the limited length, such as `V256` or `V1024`.



Note: This alternative does not exist for unbounded arrays.

- **Transform to Variable-size Arrays on the PL/I Side**

As an alternative for unbounded arrays on the XML side, they can be mapped to PL/I arrays using `(*,*,*)` notation. Only filled elements are transferred. Note that PL/I does not allow resizing of these data types and arrays. In this case the RPC message size is reduced compared

with the first alternative *Transform to Fixed-length PL/I Fields and Arrays* above. To use this possibility, uncheck the check box.



Note: This alternative does not exist for variable-length fields.

Other

If you later use wrappers other than the COBOL Wrapper, Natural Wrapper or PL/I Wrapper, no transformation rules are required. Variable-length fields and unbounded arrays are extracted as is; there are no restrictions regarding data length that can be transferred in variable-length fields and the number of elements that can be transferred in unbounded arrays.

Press **Finish** to start extraction.

Extraction Result

When the operation is completed, the IDL file is opened with the *Software AG IDL Editor*.

If the WSDL source files to extract from contain parameters that cannot be mapped to IDL parameters, an IDL file with incorrect IDL syntax is created. The unsupported parameters lead to IDL parameters of data type Error, which is not supported. In the **Problems View** you get a marker for the first error in the IDL file.

4 Using the IDL Extractor for WSDL in Command-line Mode

See *Using EntireX in the Designer Command-line Mode* for the general command-line syntax. The table below shows the command-line option for the IDL Extractor for WSDL.

Task	Command	Option	Description
Extract an IDL file and an XMM file from a web service.	-extract:wsl	-help	Display this usage message.
		-project	Name of the project or subfolder where the IDL and XMM files are stored.

Example

```
<workbench> -extract:wsl /Demo/example.wsl
```

where *<workbench>* is a placeholder for the actual EntireX design-time starter as described under *Using EntireX in the Designer Command-line Mode*.

Status and processing messages are written to standard output (stdout), which is normally set to the executing shell window.

5 WSDL to IDL Mapping

- Extracting IDL from WSDL Files 22
- Mapping WSDL XML Schema Data Type to Software AG IDL 22
- Extracting the Name for the IDL Library 23
- Extracting the Name for the IDL Program 23

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 <i>n</i> or B <i>n</i>) ⁽³⁾
hexBinary ⁽¹⁾	binary	BV (or BV <i>n</i> or B <i>n</i>) ⁽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 <i>n</i> or A <i>n</i>) ⁽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](#).

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>
```


6 Writing Web Service Client Applications

- Web Service Clients 26
- Configuring Advanced Web Service Clients 27
- Example: Setting up an EntireX Client to Consume a Secured Web Service 28

Web Service Clients

EntireX, in conjunction with the Software AG Web Services Stack (WSS), provides development and runtime functionality to support EntireX RPC clients consuming (or calling) web services. The relevant products parts are:

- IDL Extractor for WSDL to generate an XML/SOAP mapping from the service's WSDL.
- XML Mapping Editor to adapt the mapping file if necessary. See *XML Mapping Editor*.
- EntireX RPC Server for XML/SOAP, acting as the EntireX web service runtime, to deploy the XML mapping file into and perform the web service call. See *RPC Server for XML/SOAP*.
- Web Services Stack client runtime, which handles the underlying SOAP, WS-Policy and message transport. See the separate Software AG Web Services Stack documentation.

For each web service client that is deployed in RPC Server for XML/SOAP, a special configuration is required. The default name of the configuration file is *entirex.xmlrpcserver.configuration.xml*. Example configuration:

```
...
<TargetServer name="http://localhost:10010/wsstack/services/example">
  <xmms>
    <exx-xmm
      name="C:\MyWorkspace\Example\example.xmm"
      wsdl="http://localhost:10010/wsstack/services/example?wsdl"
      service="example"
      port="EXAMPLESOAP11Port"
      soapVersion="1.1"
      repository="C:\SoftwareAG\WS-Stack\repository" />
    </xmms>
  </TargetServer>
...
```

where code	is the XMM mapping file for the service
wsdl	is the reachable URL for the WSDL file of the service. This WSDL file can contain additional WS-Policy information for the service that is supported by the Web Services Stack
service	is the service name inside the WSDL file of the service
port	is the port inside the WSDL file. This information is needed when the WSDL file can contain more than one port. The value in this example is the default number; this number can be changed during installation
soapVersion	can be "1.1" or "1.2"
repository	is the client "repository" of the Web Services Stack (containing the <i>conf</i> and <i>modules</i> subfolders)

Configuring Advanced Web Service Clients

A Web Services Stack client using advanced web services functionality (WS-Security, WS-ReliableMessaging, etc.) requires the following configuration data:

- A "repository" containing configuration files and extension modules (.mar files). The "repository" is a folder containing subfolders *conf* and *modules*.
- The *conf* folder contains the client's global configuration file *axis2.xml* for the Web Services Stack engine.
- The modules folder contains modules for WS-* extensions, for example
 - *addressing-1.4.mar* if WS-Addressing is used
 - *rampart-1.4.mar* if WS-Security is used
 - *rahas-1.4.mar* if WS-Trust is used
- If WS-Security is used, an additional security configuration file *wsclientsec.properties* is required. The name and location of this file can be configured in the client's global configuration file *axis2.xml*, using the `securityConfigFile` property.

Here is an example of a client security configuration file *wsclientsec.properties*:

```

USERNAME=client
ENCRYPTION_USER=service
PASSWORD_CALLBACK_HANDLER_CLASS=com.softwareag.wsstack.test.PasswordCallbackHandler
KEYSTORE_FILE_ENCRYPT=client.p12
KEYSTORE_TYPE_ENCRYPT=pkcs12
KEYSTORE_PASSWORD_ENCRYPT=apache
KEYSTORE_FILE_SIGN=client.p12
KEYSTORE_TYPE_SIGN=pkcs12
KEYSTORE_PASSWORD_SIGN=apache
KEYSTORE_SSL_LOCATION=clientKS.p12
SSL_KEYSTORE_TYPE=pkcs12
SSL_KEYSTORE_PASSWORD=apache
TRUSTSTORE_SSL_LOCATION=clientKS.p12
TRUSTSTORE_SSL_TYPE=pkcs12
TRUSTSTORE_SSL_PASSWORD=apache

```

The `USERNAME` specifies the user name that the web service client uses to authenticate itself with the web service. It corresponds to `Alias` as described for the service configuration. `ENCRYPTION_USER` and `PASSWORD_CALLBACK_HANDLER` correspond accordingly. The example Java password callback handler from above can also be used for the client. The Web Services Stack provides some default password callback handlers that can be instantly used without needing to write a custom one. For example `com.softwareag.wsstack.pwcb.ConfigFilePasswordCallbackHandler`, which uses a simple user configuration file `users.xml`. See the separate WS-stack documentation for more details.

A Web Services Stack client that connects to a Web Service that requires advanced Web Services policies (which are attached to the service's WSDL as policy attachment) automatically sets up and processes the necessary SOAP headers in the SOAP message exchange with the service and fills the required parameters according to the configuration information described above.

Example: Setting up an EntireX Client to Consume a Secured Web Service

This section describes how to set up EntireX RPC clients calling a remote web service that has a WS-Security UsernameToken policy in effect. Two scenarios are described: one where the security policy is defined in the WSDL, and one where the policy is *not* defined.

- [Setting up an EntireX RPC Server to Configure WS-Security](#)
- [Scenario 1: Service requires UsernameToken and has a Security Policy in the WSDL](#)
- [Scenario 2: Service requires UsernameToken but does not declare this in the WSDL](#)

Setting up an EntireX RPC Server to Configure WS-Security

To set up a dedicated RPC Server for XML/SOAP instance to connect EntireX RPC clients to a secured web service, the following prerequisites apply, for example in a folder of their own in the file system:

- A startup script, `jxmlserver.bat`. You can copy this from `<Install-Dir>\EntireX\bin` and adapt it.
- Property file and config file, `entirex.xmlrpcserver.properties` and `entirex.xmlrpcserver.configuration.xml`. You can copy these from `<Install-Dir>\EntireX\conf` and adapt them.
- A WSS client repository containing the subfolders `conf`, `modules` and `services`. You can copy this from `<Install-Dir>\WS-Stack\repository` and adapt it



Note: For this example only the `addressing` and `rampart` modules are required; delete the others.

- A WSS client security configuration properties file, `wsclientsec.properties`, containing at least values for `USERNAME` and `PASSWORD_CALLBACK_HANDLER_CLASS`. You can copy this from `<Install-Dir>\EntireX\bin` and adapt it.

Scenario 1: Service requires UsernameToken and has a Security Policy in the WSDL

In this scenario, the Web Services Stack runtime can use the WS-Security policy from the WSDL to determine which security headers need to be attached to the SOAP message. Follow the steps below:

➤ To set up an RPC Server for XML/SOAP with defined security policy

- 1 Store a copy of the service's WSDL (which also includes the policy attachment) in the test folder.
- 2 Generate an XML/SOAP mapping file (.xmm) with the IDL Extractor for WSDL. Enter the name of the RPC Server for XML/SOAP ("XMLSERVER" in this example) under **Broker Settings** on the wizard page.
- 3 Start the RPC Server for XML/SOAP in a command window, using the `start` script.
- 4 Deploy the mapping file to the RPC Server for XML/SOAP. Provide the location of the WSDL and specify the desired service endpoint, name and port.
- 5 Configure the Web Services Stack repository for the service, using the following steps:
 1. Stop the RPC Server for XML/SOAP.
 2. Edit the file `entirex.xmlrpcserver.configuration.xml` and add the repository definition to the `TargetServer` section. For example:

```
<TargetServer
  name="http://localhost:10010/wsstack/services/example.EXAMPLESOAP11Port/">
  <xmms>
    <exx-xmm
      name="D:\TestWS\example.xmm"
      port="EXAMPLESOAP11Port"
      wsd1="D:\TestWS1\example.wsdl"
      service="example"
      soapVersion="1.1"
      repository="repository"/>
    </xmms>
  </TargetServer>
```

3. Restart the RPC Server for XML/SOAP.
- 6 Configure security for the WSS client runtime by modifying files `wsclientsec.properties` and `users.xml`:
 - File `wsclientsec.properties`, containing the lines

```

USERNAME=user
PASSWORD_CALLBACK_HANDLER_CLASS=
  com.softwareag.wsstack.pwcb.ConfigFilePasswordCallbackHandler
    
```

Specify the desired username, which should go into the `UsernameToken`. The password callback handler class is used by the WSS client runtime to inquire a password for this user. The `ConfigFilePasswordCallbackHandler` is a simple default handler delivered with Web Services Stack that reads the password of a given user from a flat file `users.xml`. You can write a custom password callback handler for other methods to acquire passwords.

- File `users.xml`. Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<users>
  <user username="user" password="pass" />
  <user username="client" password="apache" />
  <user username="service" password="apache" />
  <user username="bob" password="bobPW" />
</users>
    
```

To test access to the remote web service, use the *EntireX XML Tester* on the IDL file.

Scenario 2: Service requires `UsernameToken` but does not declare this in the WSDL

For this scenario, perform the steps as described above. Because the WSDL does not contain a security policy stating that `UsernameToken` is required, perform this additional step:

- Explicitly tell the Web Services Stack client runtime about the `UsernameToken` required by the service. Edit `<SuiteInstallDir>/profiles/CTP/workspace/wsstack/repository/conf/axis2.xml`, uncomment the `rampart` module and add the `OutflowSecurity` parameters:

```

<module ref="rampart"/>
<parameter name="OutflowSecurity">
  <action>
    <items>UsernameToken</items>
    <user>user</user>
    <passwordType>PasswordText</passwordType>
    <passwordCallbackClass>
      com.softwareag.wsstack.pwcb.ConfigFilePasswordCallbackHandler
    </passwordCallbackClass>
  </action>
</parameter>
    
```

where `user` is a valid user name for authentication.