Provided by **EPO Consulting**
E-Mail info@epoconsulting.com
Internet www.epoconsulting.com

DEPLOYMENT, CONFIGURATION & DEVELOPMENT DOCUMENTATION

SAP® Certified
Powered by SAP NetWeaver®

Status:	Release
Version:	30A SP3
Last updated:	October 2009

EPO XML CONNECTOR FOR SAP

Short table of contents – Configuration & Development documentation

Full table of contents – Configuration & Development documentation

1 GENERAL, FUNCTIONS, AREA MENU

1.1 INTRODUCTION

This documentation is intended to provide:

- ✓ instructions on using the EPO XML Connector to exchange information between SAP and any other third-party system
- ✓ describing the scope of the integrations, which can be done with the EPO XML Connector
- ✓ explaining how existing and new integrations have to be set up

For installation of the EPO XML Connector please refer to the "Installation Documentation".

1.1.1 WHAT IS THE EPO XML CONNECTOR? – SOA LAYER

The EPO XML Connector is an SAP ABAP add-on software product, which enables easy and standardised integration from SAP R/3 to any third-party system and vice versa.
The communication between these two systems is based on sending and receiving XML messages (Web Services) or alternatively you can use file protocol functionality for XML or other file format communication.
It creates a service layer (SOA), where communication messages can be viewed and edited. The communication layer is separated from the functionality of the SAP system and the integrated third-party systems. This is a very important concept of this connector, because it brings great advantages during development and productive usage. Especially changes of one system can be handled much easier.

It creates direct integration between systems with a communication layer.

Technically the EPO XML Connector is an ABAP add-on product. There are no modifications of the SAP system. It is developed in namespace /EPO1/.

1.1.2 IMPORTANT NAMING TERMINOLOGY

For the EPO XML Connector we always take the look of SAP, when describing (web) services.

1.1.2.1 INBOUND (CALLING A SAP SERVICE)

- ✓ *SAP provides the service:* This can be a web service of a function module (published) or any other service from SAP, which can be accessed from outside using the HTTP(S) protocol.

- ✓ *An external application consumes the SAP service*
- ✓ *The request XML comes from outside, the response XML is from SAP*
- ✓ *Request XML has direction IN*
- ✓ *Response XML has direction OUT*

1.1.2.2 OUTBOUND (CALLING AN EXTERNAL SERVICE)

- ✓ *Service is provided external:* This can be a web service or any other service outside SAP using the HTTP(S) protocol.
- ✓ *SAP consumes the external (web) service*
- ✓ *The request XML comes from SAP, the response XML is sent to SAP*
- ✓ *Request XML has direction OUT*
- ✓ Response XML has direction IN

1.1.2.3 OVERVIEW OF FUNCTIONALITY (SAP INTEGRATION)

Technical view

Web Services
- ✓ Integrate external Web Service
- ✓ *Provide Web Services*

File Interfaces/FTP/…
- ✓ *Create Files (XML and others) and store it or send it out*
- ✓ *Receive (Upload) files into SAP*

IDOC interfaces
- ✓ *Currently only supported with API for monitoring. Full IDOC support only with additional programming. Please contact EPO Consulting, if you want to use EPO XML Connector for IDOC interfaces.*
  Will be fully, integrated supported with next major release of the EPO XML Connector

Business view

Third-party software integration
- ✓ Tested and SAP certified with Metastorm BPM (using Microsoft Web Service technology = .NET Web Service technology)
- ✓ *Integrates any web service enabled software*

.NET developments
- ✓ *Enables direct web service integration using SOAP and WSDL. Significant advantage to .NET Connector from SAP (using proprietary RFC)*

.Java developments
- ✓ *Enables direct web service integration using SOAP and WSDL. Significant advantage to .Java Connector (JCO) from SAP (using proprietary RFC)*

Application integration
- ✓ *Web service integration*
- ✓ *File integration*

Document integration
- ✓ *XML document integration (Adobe Interactive Forms, MS Excel XML documents etc.)*
- ✓ *Microsoft Excel integration*

- ✓ *Other structured document integration (CSV etc.)*

SAP Portal integration

- ✓ *Web service integration instead of JCO integration (using proprietary RFC)*

Portal and Website integration

- ✓ *Web service integration*
- ✓ *xHTML interactive forms with basic workflow functionality*

Adobe products integration

- ✓ *Adobe forms integration*

There will be new and existing applications, which will have SAP integration provided by the EPO XML Connector.

1.2 SYSTEM REQUIREMENTS

- ✓ Any SAP system with SAP NetWeaver Application Server (ABAP) 6.20, 6.40, 7.00 or newer. See SAP menu: System – Status – Component version. See component SAP_ABA.

  (WAS 6.20 needs patch level 29 for EPO XML Connector. It uses cl_abap_gzip class to compress XML messages)

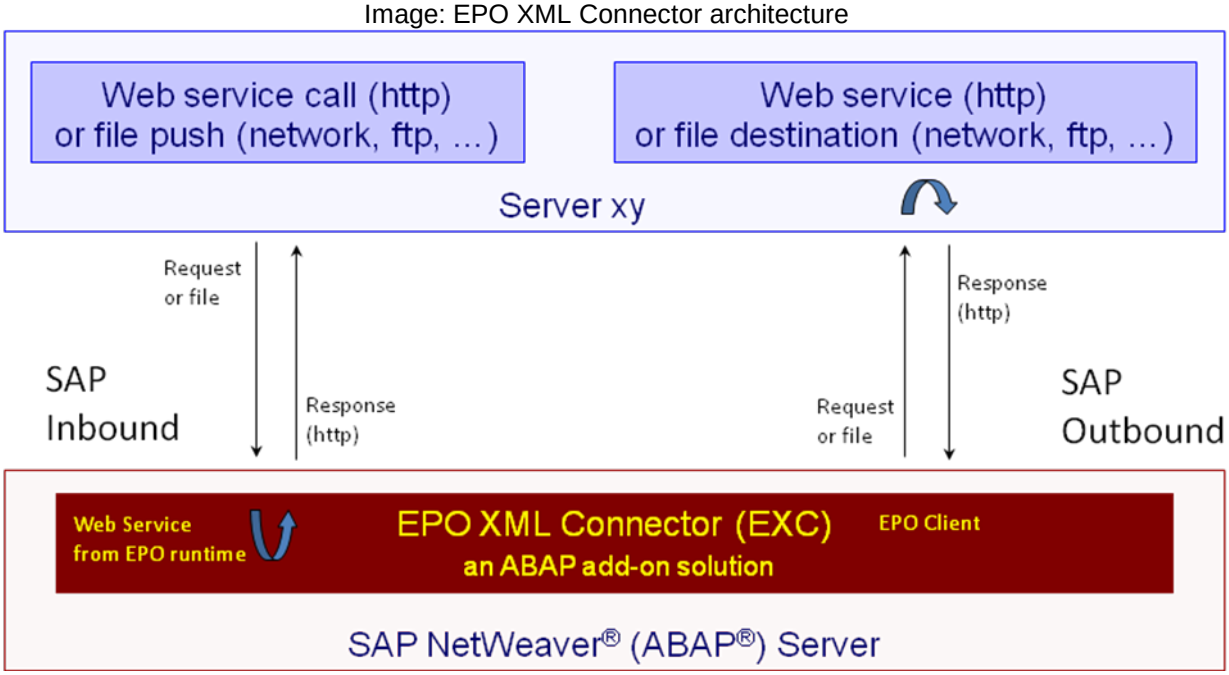
Note: The EPO XML Connector is an official SAP component. So you can see it after installation in the SAP menu: System – Status – Component version. See component EPO1.

1.3 ARCHITECTURE, SCALABILITY, PERFORMANCE AND STABILITY

1.3.1 ARCHITECTURE

The EPO XML Connector is written 100% within the ABAP workbench. It can be installed on any SAP NetWeaver ABAP server.
It can handle any incoming message and forward it to the SAP function (SAP Inbound). It can also take a message for sending out and send it (SAP Outbound). The protocol used (http, file, ftp,...) does not influence the interface within the EPO XML Connector. Therefore protocols for interfaces can be changed anytime with little effort.

Image: EPO XML Connector architecture



1.3.2 Scalability

The architecture allows endless scalability. Scaling up the solution at any time is easily possible.

Setting up http listener processes can be done with setting a profile parameter (number of parallel threads per application server). All other settings influencing the performance of the EPO XML Connector are SAP NetWeaver Web Application Server settings (including logon procedures), too.

Central SAP system installation
The EPO XML Connector can be installed on the central SAP system. This is often SAP ERP (ECC 6.0). Load balancing can be achieved with standard load balancing (1 message server, several application servers – load balancing also works with http, same like rfc). The SAP Web Dispatcher is another option for load balancing and security settings (can be setup in DMZ).

Central SAP system installation on dedicated server
The EPO XML Connector can also be installed on a separated, dedicated SAP NetWeaver application server. The communication from this “middleware / EAI” server (which is a SAP NetWeaver runtime) to the back-end SAP systems is easily achieved with RFC calls.
Another option would be using SAP Solution Manager or another central SAP system.

Decentralised SAP system installation
The EPO XML Connector can be installed on any SAP solution running on SAP NetWeaver ABAP. It can be used to allow direct connections to each of those SAP systems.

1.3.3 Performance

The EPO XML Connector was designed and implemented for highest performance. Because of its architecture allowing direct integrations it outpaces all other solution. There is virtually no overhead, which would slow down integrations. The actual function is either a standard SAP function module or a custom program, which must be called anyway.
In fact, because of using 100% ABAP it reaches almost same performance values as users experience within the SAP GUI.

1.3.4 Stability

The EPO XML Connector uses SAP NetWeaver application server without modifying anything. It just uses standard functionality (which is often completely unknown) of SAP NetWeaver. This is the SAP server used for all SAP solution based on ABAP technology (like SAP ERP, ECC 6.0, SCM...). It is almost needless to say, that this is one of the most stable servers.

The EPO XML Connector is used on SAP production systems with millions of transactions per year.

1.4 EPO XML Connector Area Menu /EPO1/SOA7

Access area menu with transaction code: **/n/EPO1/SOA7**

Note: For WAS 6.20 version the area menu is **/n/epo1/soa** and contains only functionality available on WAS 6.20.

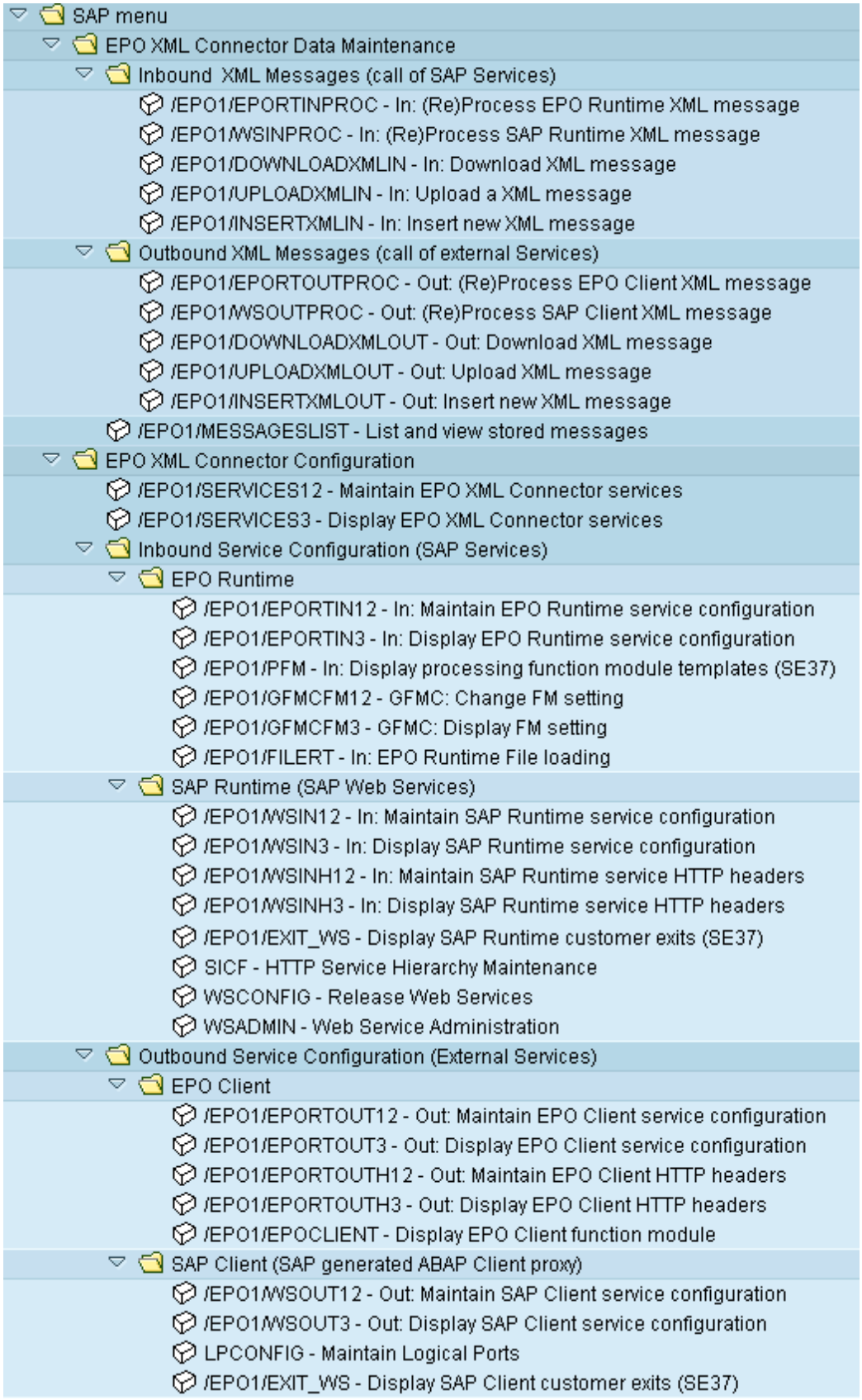
Note: For some reason it is necessary to enter the transaction code for the area menu twice, because it gives an error message the first time (error message: transaction code does not exist).

The area menu is designed to help you navigate through the functionality of EPO XML Connector. It is divided into three main sections. Each main section will be used by different groups of users:

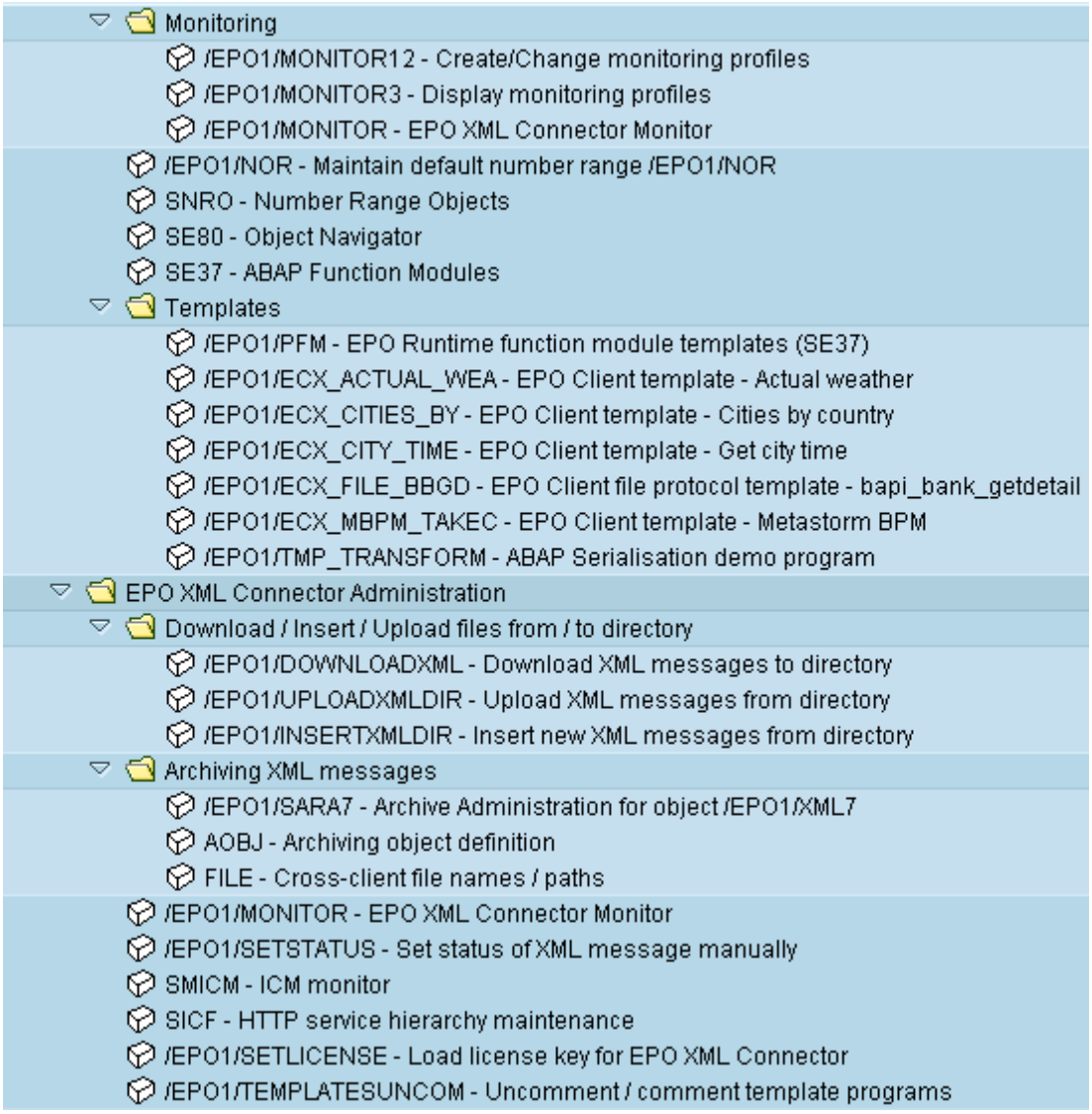
- “Data maintenance”,
- “Configuration” and
- “Administration”.

“Data maintenance” and “Configuration” section then divide into “Inbound” and “Outbound” subsections hierarchically to help you choose the right option. Finally the “Administration” section comprises universal file handling functions, archiving object manipulation and set of transaction for the Connector maintenance.

Tip: Authorisation roles can be directly created from sections or subsections of area menus.



Area menu continues on the next page.



1.5 FUNCTIONS AND FEATURES

Common functionality

- ✓ Synchronous and asynchronous processing and reprocessing
- ✓ Logging transactions, storing XML request and response messages
- ✓ XSLT transformations of request and response messages
- ✓ Customer exits where it is not possible to enter the process directly
- ✓ Using file(s) (FILE protocol) as request(s).
- ✓ Downloading and uploading XML messages to/from files
- ✓ Synchronous and asynchronous monitoring of transactions

1.5.1 INBOUND

“Inbound” in EPO XML Connector stands for integration where your SAP system provides the service, therefore primary connection is made from outside by “incoming” request. There are two options to implement such scenario in the connector: EPO runtime and SAP runtime.

1.5.2 OUTBOUND

The outbound part of the EPO XML Connector represents scenarios where there is an existing web service “outside” your SAP system, which you want to use in SAP. Thus your system needs to send out the request in order to receive the response. Like in the inbound part there are two options to implement this scenario in the connector: The unique EPO Client and the generated ABAP client proxy (enhanced SAP standard). The EPO Client also allows writing request messages down to a file system or to send it out using ftp (ftp target folder must be mapped like a network drive).

1.5.3 MAPPING & INTERFACE STRUCTURE DEFINITION

Starting from EXC 17E (07E on WAS 6.20) support package one we have enabled you to create mapping and structure function modules for your services. Although you do not need to use this functionality, it is recommended to do so. The configuration input parameters are only informative, no functions are called automatically, and you need to call them yourself in your service code. The main reason for doing this is having central storage for your XML interfaces, both inbound and outbound, and ability to choose from existing interfaces rather then writing a new one for each service. There are two types of function modules available – structure and mapping. Structure FM we use for converting XML into ABAP variables, structures, or tables and vice versa. In mapping FM we change output or prepare input variables for structure FM. Of course this is only how we used it in our examples, as I said before; the notations in configurations do not really do anything, so you can use these functions for other purposes.

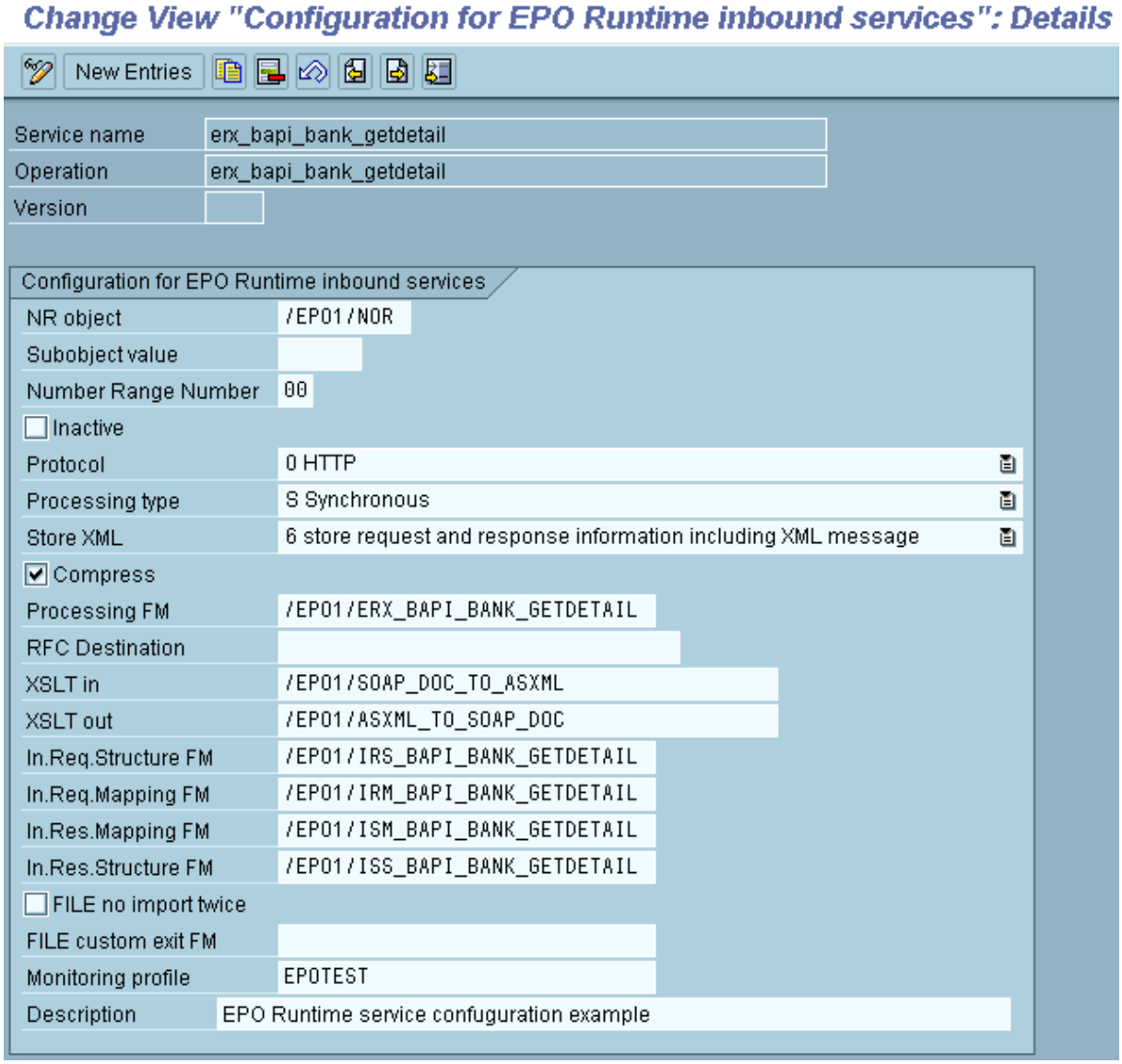
Example: You use the same XML request message for several services, each with different functionality. So, you wrote CALL TRANSACTION ID SOURCE XML... for getting the ABAP variables out of the XML. You may or not put in into function module to be able to use it again, but now you can make a note into your service configuration, that it uses this particular FM. To be able to do this you only need to put the FM into package /EPO1/EXC_REPOSITORY and corresponding Function Group.

Available function groups are: (in package /EPO1/EXC_REPOSITORY)

- EPO Runtime (inbound)
  - /EPO1/IN_REQUEST_STRUC (XML to ABAP)
  - /EPO1/IN_REQUEST_MAP (ABAP to ABAP)
  - /EPO1/IN_RESPONSE_MAP (ABAP to ABAP)
  - /EPO1/IN_RESPONSE_STRUC (ABAP to XML)

- EPO Client (outbound)
  - /EPO1/OUT_REQUEST_MAP (ABAP to ABAP)
  - /EPO1/OUT_REQUEST_STRUC (ABAP to XML)
  - /EPO1/OUT_RESPONSE_STRUC (XML to ABAP)
  - /EPO1/OUT_RESPONSE_MAP (ABAP to ABAP)

Image: Structure and mapping FMs in EPO Runtime service configuration



1.5.4 Feature: Central data storage

There is an option for each scenario to store

- header data and
- message data

For synchronous services (web services) this storage can be done for request and response.

Header data (logging)
All header data is stored in table /EPO1/XMLHEAD

Message data
All message data is stored in table /EPO1/XMLDATA. The data is stored in binary format.

This feature creates a central place for monitoring of the integration. Independently from the type of the integration (file upload/download up to web services) all data is stored in 1 central place.

It enables
- logging (who, what, when,)
- reprocessing
- display of messages
- editing of message

1.5.5 Feature: Reprocessing of services

When a service is set up to store message data, it can be reprocessed. Reprocessing is only possible, if the status of the message to be reprocessed is less than 53.

Asynchronous services
Processing must be done to fulfil the purpose of the service. Jobs can be scheduled to automate this. Reprocessing can be done in error cases.

Synchronous services
Reprocessing can be done in error cases.

1.5.6 Feature: XSL Transformations (XSLT)

Since Release 6.10 of the SAP Web Application Server (SAP Web AS), XSL Transformations (XSLT) have been integrated in ABAP via the CALL TRANSFORMATION command. XSLT is the most powerful and advanced technology available for the transformation of XML documents. XML data can be transformed into ABAP data structures and vice versa; however, XSLT is not limited to those types of output. You can also generate HTML documents or plain text files that are made available as loadable assets to other applications. XSLT is widely used and well documented, we don't intend to give all the information in this document – just a brief start point - you can find useful links in the end of this appendix. In EXC we use these transformations for formatting input and output XML messages and for XML to asXML conversion.
We recommend using your own transformation software to develop your XSLT transformations. You can also get professional service from EPO Consulting for this task. Or you can use SAP transaction XSLT and XSLT_TOOL to develop XSLT transformations.

XSLT can be applied to services in the EPO XML Connector for inbound and outbound messages in the configuration.

You must store all XSLT as "Transformations" in data dictionary using the object browser (SE80). See "Others – Transformations".

1.5.7 Feature: SAP authorisation by service

With EXC release 30A SP3 you can switch on authorisation check for services. This allows you to control the authorisation for any integration with standard SAP security. Technical details are described in chapter 3.2 Authorisation object.

1.5.8 Concept of XML to ABAP and ABAP to XML transformation (Serialisation)

XML messages must be transformed to ABAP variables or ABAP variables must be transformed into XML messages. In the EPO XML Connector we are using the standard ABAP command

CALL TRANSFORMATION

This is an implementation of the standard XSLT transformations as defined from W3C (www.w3.org).

You can find a demo program in the EPO XML Connector area menu (EPO XML Connector Configuration - Templates). The documentation including the source code of the demo program can be found in Appendix 2: Demonstration of ABAP Serialisation.

asXML format
SAP has defined asXML as an internal XML format for ABAP transformations. asXML can be transferred with the internal "ID" transformation to ABAP variables. Also when transferring ABAP variables into an XML string using the ID transformation, asXML will be the result.
Important: asXML can only contain XML elements in capital letters. For example <MESSAGE> is allowed, but not <Message>.

XML transformation into ABAP
When transforming a XML document into ABAP it can be done in 2 steps:
First transform the XML document into asXML and then transform it into ABAP using the ID transformation. Obviously these 2 steps can be combined into 1. But for understanding the concept the first transformation is decisive.

Step 1: XML to asXML

```
CALL TRANSFORMATION yourxslt
        SOURCE    XML                          sourcexml
        RESULT    XML                          asXML
```

Step 2: asXML to ABAP variables

```
CALL TRANSFORMATION ID
        SOURCE    XML                          asXML
        RESULT    XMLElement1            =     ABAPVariable1
                  ComplexXMLElement1     =     ABAPDeepStructureVariable1.
```

Or in 1 step: XML to ABAP variables

```
CALL TRANSFORMATION yourxslt
        SOURCE    XML                          sourcexml
        RESULT    XMLElement1            =     ABAPVariable1
                  ComplexXMLElement1     =     ABAPDeepStructureVariable1.
```

The EPO XML Connector uses the 2 step technique for EPO runtime and EPO Client. This enables you to assign the XSLT in the configuration and do only the ID transformation in your custom ABAP programming.

For SOAP XML to asXML transformation a generic XSLT is provided with the EPO XML Connector:

/EPO1/IN_SOAP_TO_ASXML	SOAP XML to asXML transformation translating all XML elements into uppercase.

ABAP into XML transformation

This is achieved again with the ABAP command CALL TRANSFORMATION and can also be done in 1 or 2 steps.

Step 1: ABAP variables to asXML

```
CALL TRANSFORMATION ID
     SOURCE   XMLElement1          =  ABAPVariable1
              ComplexXMLElement1   =  ABAPDeepStructureVariable1
     RESULT   XML                     asXML.
```

Step 2: asXML to XML

```
CALL TRANSFORMATION yourxslt
     SOURCE   XML                     asXML
     RESULT   XML                     targetxml.
```

Or in 1 step: ABAP variables to XML

```
CALL TRANSFORMATION yourxslt
     SOURCE   XMLElement1          =  ABAPVariable1
              ComplexXMLElement1   =  ABAPDeepStructureVariable1
     RESULT   XML                     yourxml.
```

The EPO XML Connector uses the 2 step technique for EPO runtime and EPO Client. This enables you to assign the XSLT in the configuration and do only the ID transformation in your custom ABAP programming.

For asXML to SOAP XML transformation a generic XSLT is provided with the EPO XML Connector:

/EPO1/ASXML_TO_SOAP	asXML to SOAP XML transformation. It uses a XML element <RESPONSEOPERATION> to determine the name of the operation (the child of <body>). This allows the XML Transmitter to create a valid WSDL file. Example: <RESPONSEOPERATION>BAPIUserGetDetail</RESPONSEOPERATION> will lead to <body><BAPIUserGetDetailResponse>… If <RESPONSEOPERATION> is not created with CALL TRANSFORMATION, the xml element will be named just <Response>.

Creating a web service for SOAP XML for SAP Inbound (SAP provides the web service)
This means, that the request XML will be a SOAP message posted to SAP. So you will apply XSLT /EPO1/IN_SOAP_TO_ASXML in the configuration for direction I (In). For the response XML you will apply XSLT /EPO1/ASXML_TO_SOAP for direction O (Out).

1.5.9 Concept of other data format to ABAP transformation and vice versa

All messages are kept in binary format in the EPO XML Connector. Therefore any message format can be handled.

1.5.9.1 SAP Inbound data conversion using EPO runtime

EPO runtime requires using a processing function module for processing the data in SAP. The message data is provided with IMPORTING parameter
I_REQUESTXML TYPE XSTRING

Now you can use any technique available in ABAP to use this binary data. Beside XML data handling with ABAP command CALL TRANSFORMATION there are various function modules etc. available.

Examples:
“Upload” or http(s) receipt of Microsoft Excel files (.XLS)
“Upload” or http(s) receipt of text files, CSV files or other structured files
“Upload” or http(s) receipt of IDOC files

1.5.9.2 SAP Outbound data conversion using EPO Client

EPO Client needs the request data in binary format. It must be provided to IMPORTING parameter
I_REQUESTXML TYPE XSTRING

Now you can use any technique available in ABAP to produce this binary data. Beside XML data handling with ABAP command CALL TRANSFORMATION there are various function modules etc. available.

Examples:
“Send” or download of Microsoft Excel files (.XLS)
“Send” or download of text files, CSV files or other structured files
“Send” or download of IDOC files

2 INITIAL CONFIGURATION

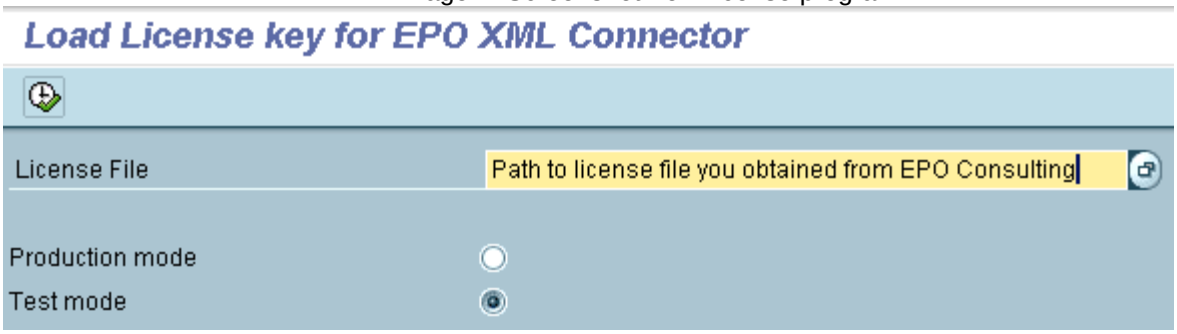
There are few things you have to do before you can start working with EPO XML Connector. First you need the software license key for the connector to operate, next you create number ranges for your services and finally you must activate connector services for inbound calls.

2.1 UPLOAD LICENSE KEY

In order to use EPO XML Connector on SAP production systems you need to upload the license key file into your system, which you can obtain from EPO Consulting. To try out our product first, there are evaluation licenses available as well.

> **Area menu:** *EPO XML Connector Administration → Load license key for EPO XML Connector*
> **Transaction:** */EPO1/SETLICENSE*

Image 1: Screenshot from license program



You must upload the license key in “Production mode”!

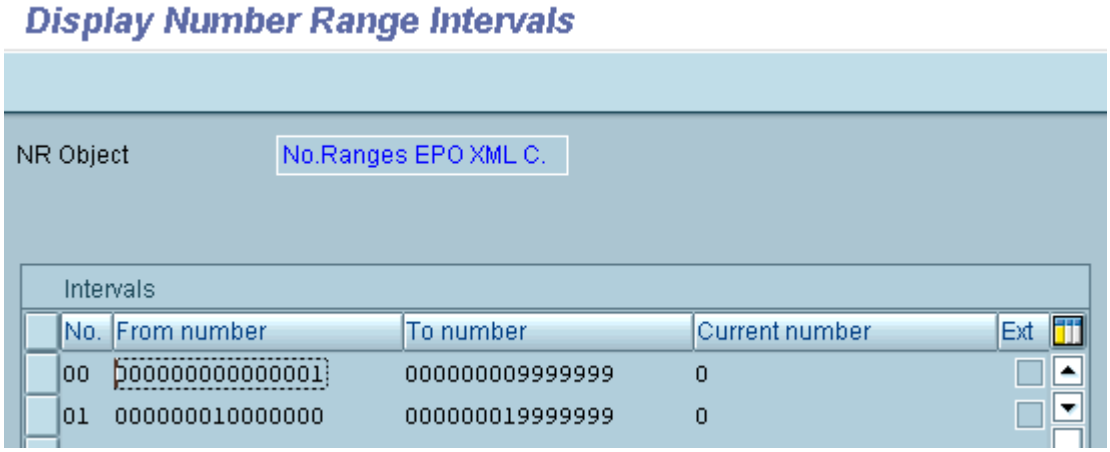
2.2 CREATE NUMBER RANGE(S)

The numbers generated by number range(s) are used to identify each transaction=message (TransactionID) for services used with the EPO XML Connector. You can create your own number range object using transaction SNRO (also in EPO area menu - *EPO XML Connector Configuration→ Number Range Objects*) or can use predefined “/EPO1/NOR” number range object, but in that case you need to set up at least one number range – default “00” range, although you can have different number range for each service eventually (see example on image below). We recommend using at least 2 different number ranges for inbound and outbound services.

Note: Default number range number ‘00’ (identifier of number range) is used when nothing is set in the service configuration.

> **Area menu:** *EPO XML Connector Configuration → Maintain number range for EPO XML Connector messages*
> **Transaction:** */EPO1/NOR*

Image: Number ranges example



2.3 ACTIVATE SERVICES

Since it is not possible to deliver active services, you are required to activate **epo1soa** service and all subsequent children of it (images below). The inbound part of the EPO XML Connector depends on these services.

Area menu: *EPO XML Connector Administration → HTTP Service Hierarchy Maintenance (ICF)*
Transaction: *SICF*

Image: Service activation in SICF

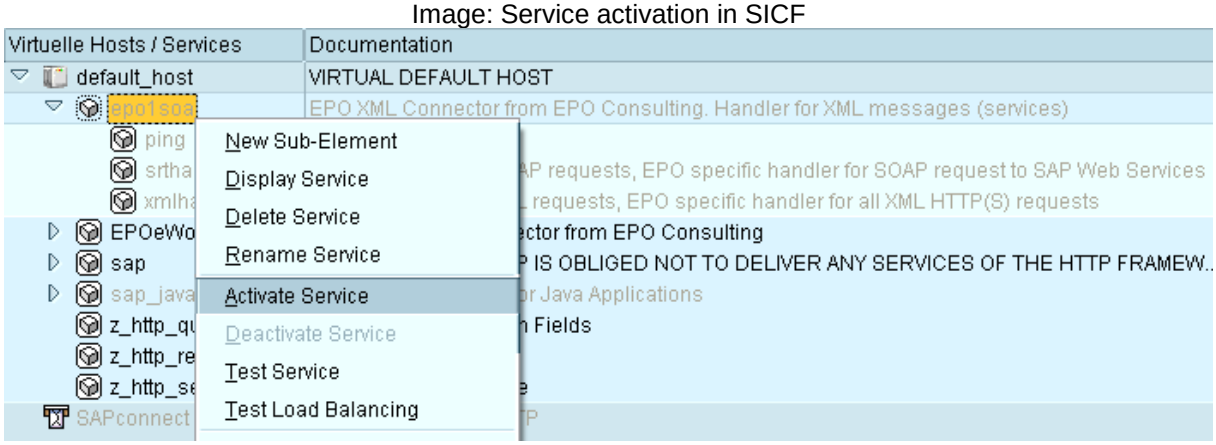
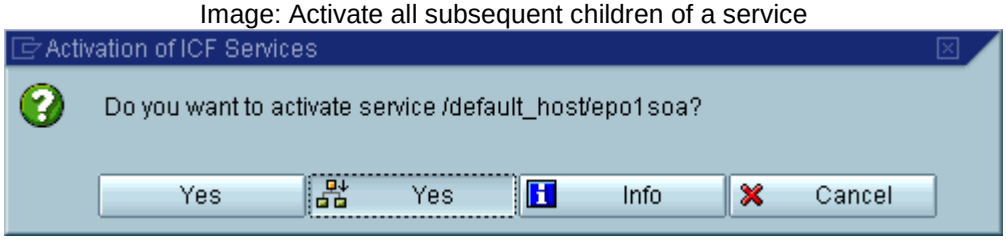


Image: Activate all subsequent children of a service



3 EPO XML CONNECTOR SERVICES CONFIGURATION & DEVELOPMENT

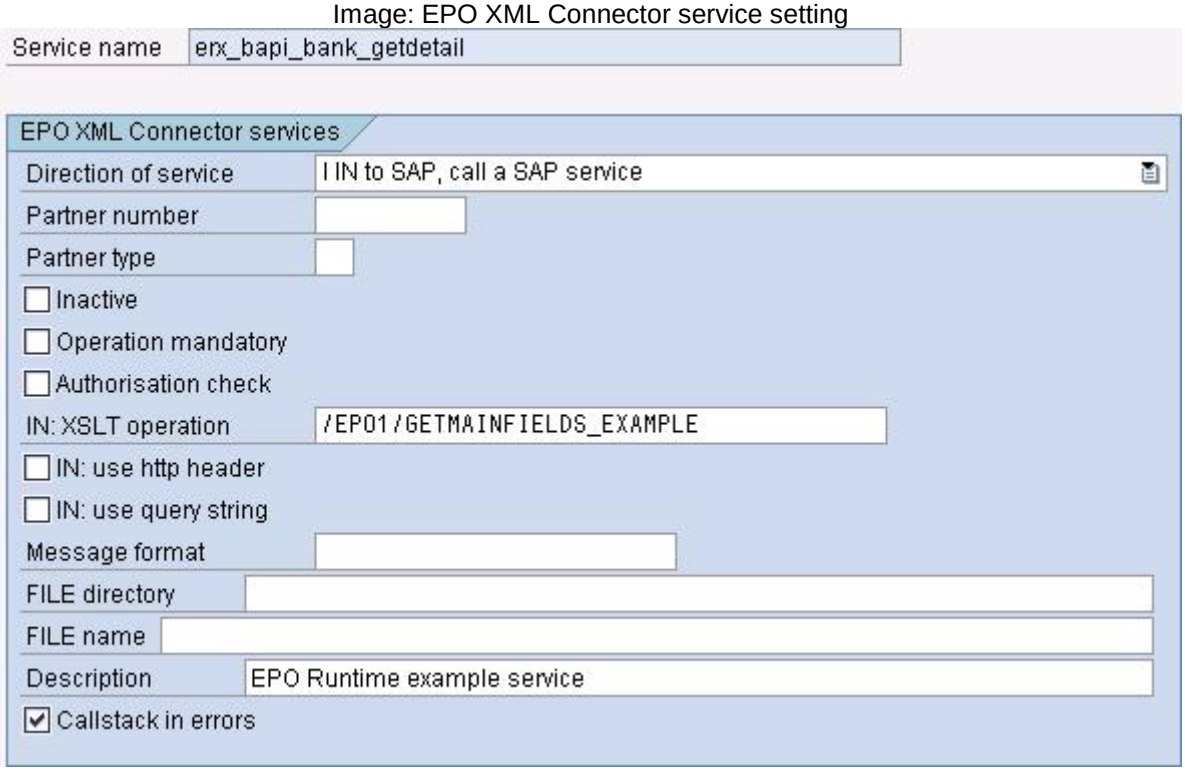
3.1 DEFINITION OF SERVICES (INBOUND & OUTBOUND)

Every operation (= integration scenario) of EPO XML Connector we call a service and every one of these services must be defined in the “services” table. Although the service record contains more information, basically it is just name and direction, which you use to create “service configuration”. This enables you to use different operations and versions of the same service.

Area menu: *EPO XML Connector Configuration → Maintain Services EPO XML Connector*
Transaction: */EPO1/SERVICES12*

Area menu: *EPO XML Connector Configuration → Display Services EPO XML Connector*
Transaction: */EPO1/SERVICES3*

Image: EPO XML Connector service setting



Service name: For inbound services the service name is determined by the name of web service you are using.
For outbound services you can freely define a unique service name.

Direction of service: Inbound (IN to SAP), outbound (OUT of SAP)

Partner number and type: Partner definition for the service. The fields will also be used for IDOC monitoring and / or integration.

Inactive: You can turn off the service by setting it to inactive.

Operation mandatory: This setting makes sure that no processing takes place if passed operation does not exist in service's configuration table. In opposite situation (operation mandatory not set) the configuration with empty operation, if exists, will be used for any non-existing operations.

Authorisation check: When this is enabled, EXC always checks user authorisation for object /EPO1/ECS for service being processed.

IN: XSLT operation: Used only for inbound.
This field is optional. XML transformation which extracts
* operation,
* version and
* foreign keys (FKEY1-FKEY4) out of the request XML message. Foreign keys (if used) are stored in the message header table (if configured to store). So you don't need to store the whole xml messages, if you want to log just up to 4 parameters from it.
Extract the "operation" of a web service here, if the web service has got more then 1 operation and http header "SOAPaction" is not used for the call.

Note: This applies to FILE protocol as well (Inbound only)

In: Use http header: Used only for inbound.
This field is optional and will only be used, if "XSLT operation" (see above) does not provide the "operation".
The handler will get
*operation – http header: "SOAPaction",
* version – http header: "version" and
* foreign keys - http header: "fkey1" to "fkey4"
from HTTP header if you set this checkbox.

Note: If you both use "XSLT operation" and set "Use http header" then using HTTP header has lower priority and will only be processed, if "operation" was not retrieved with "XSLT operation".

In: Use query string: Used only for inbound.
This field is optional and will only be used if "XSLT operation" and "HTTP Header" (see above) functions (if set) does not provide the "operation".
The handler will get
- operation variable "operation"
- version variable "version" and
- foreign keys variable "fkey1" to "fkey4"

from URL (http://server/service?operation=something).

Note: If you use all “XSLT operation”, “Use http header” and “Use query string” then using Query string has the lowest and will only be processed, if “operation” was not retrieved using the other two methods.

Message format: This field is optional
Format of a service message. By default the value is XML even when this entry is left empty. This format is used as filename extension in file handling programs and UM protocols of EPO XML Connector.

FILE directory: Directory for file reading (EPO Runtime - inbound) or storing (EPO Client - outbound) used in FILE protocol.

FILE name: File name for FILE and UM protocols. For EPO Runtime (inbound) files this field can contain search pattern. The search is using ABAP logical expression “CP”. Masking signs are ‘*’ and ‘+’ and search is not case-sensitive.
Hint: It will read first always all files in the specified directory and then reduce it to the number of matching files. So keep you directories as clean as possible.

Description: Your description of the service.

Callstack in errors: Set this flag to include ABAP call stack in /epo1/callstatus error messages for the service.

3.2 AUTHORISATION OBJECT

Authorisation check can be enabled in service configuration. It allows you to create and assign authorisations for each service.

The authorisation object is /EPO1/ECS (class EPO1), authorisation /EPO1/EXSA00, and field name /EPO1/SNAM – name of service being used. Like with any other authorisation object, you can use transactions PFGC, SU01, SU02, SU03 to maintain and assign authorisation profiles and roles. When authorisation check fails, the error message is given back in callstatus parameter.

Example: You could create a communication SAP user, which is authorised only for 1 specific SAP Inbound service. In ICF you could then create an alias (or ICF service), where you enter this user and password. This way you get a trusted service, which can be used in your intranet without logon data required.

Next chapters:

Inbound (calling a SAP service)
The inbound section is guiding you through creation and setting up EPO runtime and SAP runtime services of the EPO XML Connector (providing a Web Service).

Outbound (calling an external service)
The outbound section describes how to create and use services for communication with web services “outside” of SAP. There are two options for implementing outbound services in the EPO XML Connector: EPO Client and SAP Client (consuming a Web Service).

3.3 INBOUND: EPO RUNTIME

EPO runtime provides a unique HTTP(s) handler, which is represented by the URL address sap_default_host/epo1soa/xmlhandler. XML messages can be posted to this URL. When the handler receives the request message it is able (in order) to store, XSLT transform, and process it. (Re)Processing is done by calling customer-developed function modules. This XML handler can store or log and XSLT transform the response message as well. The same thing can be done using files stored on your file system rather then HTTP(s) requests. In this case we call it EPO FILE runtime or FILE protocol.
A WSDL file can be created easily using the “WSDL create” functionality of the XML Transmitter (freeware from EPO Consulting).

The unique EPO runtime is one option creating an inbound integration with SAP. The other option for inbound integration is using the SOAP runtime of the EPO XML Connector. You can decide service by service, which option is appropriate. We recommend using EPO runtime as it returns far better error messages.

Overview: Steps for creating inbound integrations using EPO runtime:

- Define XML messages (request and response)
- Configure the service in the EPO XML Connector
- Write the custom function module for processing the request and response XML message (e.g. map the XML elements to the internal used BAPI).
- Create WSDL using XML Transmitter
- Test your integration

3.3.1 STATEFUL (SESSION) HANDLER

On the top of previously described handler (xmlhandler), there is also a stateful handler available (sap_default_host/epo1soa/apphandler), which you can use to create server sessions for EPO Runtime services. When the session is created after successful logon, server sends identification cookie (‘set-cookie’ header) which you need to use (‘cookie’ header) to be able to access the opened session. Sending any value in HTTP header named ‘exc-terminate’ will close the session. Read more info about stateful communication at help.sap.com.

3.3.2 USING HTTP(S)

The EPO runtime is implemented as a HTTP handler for SAP NetWeaver Application server (WAS). In the SAP service tree (transaction SICF) it is located in

sap_default_host/epo1soa/xmlhandler. The call of such an inbound service is done by posting (HTTP POST) a XML request message to this handler.

Example call: http://vepo2005:8000/epo1soa/xmlhandler/BAPI_BANK_GD_ERT?
Explanation of this call:
The host “vepo2005” and port “8000” are from the SAP WAS.
“/epo1soa/xmlhandler/” is the EPO runtime HTTP handler on SAP WAS.
The URI “BAPI_BANK_GD_ERT” is the service name, which must be configured.

3.3.2.1 IMPORTANT HTTP HEADERS

SOAPaction (optional): The operation of the service.
Hint: The operation of a service can also be defined in the XML of the service itself (first child element of <soap:Body>
Sap-client (optional): SAP logon client (Web logon, not an RFC logon)
Sap-language (optional): Logon language to SAP
ContentType: For example text/xml or text/plain
ContentLength: Normally automatically added
Method: POST

3.3.3 USING FILE, FTP

You can use file(s) as a runtime request(s) by setting up job which runs **/epo1/exc_fileruntime** program. This program checks specified service and configuration and then reads files from directory specified in service settings and processes them as requests similarly to HTTP requests. For ftp access you can use the same thing by mapping ftp site to your file system.

3.3.4 CREATING EPO RUNTIME - INTEGRATION GUIDE FOR HTTP PROTOCOL

3.3.4.1 DEFINE INTERFACES - REQUEST AND RESPONSE XML MESSAGES

Request XML definition example (BAPI_BANK_GD_ERT service)

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m0:BAPI_BANK_GETDETAIL xmlns:m0="urn:sap-com:document:sap:rfc:functions">
      <BANKCOUNTRY>AT</BANKCOUNTRY>
      <BANKKEY>00100</BANKKEY>
    </m0:BAPI_BANK_GETDETAIL>
  </soap:Body>
</soap:Envelope>
```

Response XML message example (BAPI_BANK_GD_ERT interface)

```
<?xml version="1.0" encoding="utf-8"?>
<asx:abap xmlns:asx="http://www.sap.com/abapxml" version="1.0">
  <asx:values>
    <BANKADDRESS>
```

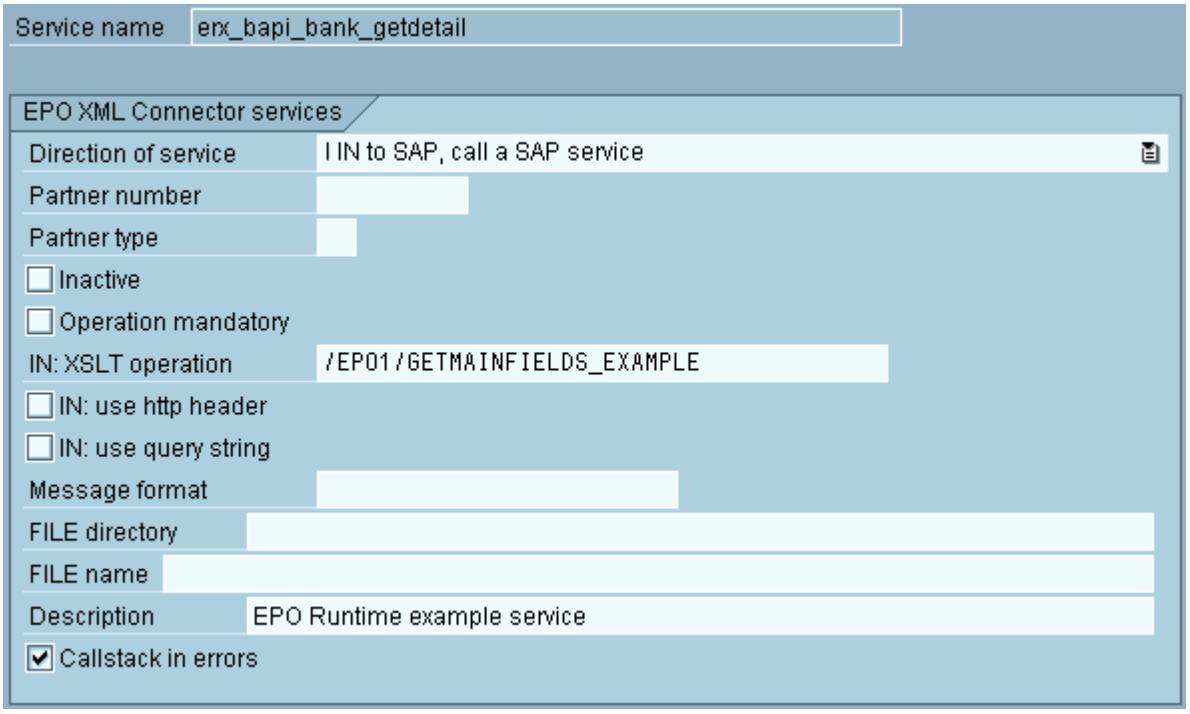
```
    <BANK_NAME>Oesterreichische Nationalbank</BANK_NAME>
    <REGION />
    <STREET />
    <CITY>1011 WIEN</CITY>
    <SWIFT_CODE />
    <BANK_GROUP />
    <POBK_CURAC />
    <BANK_NO>00100</BANK_NO>
    <POST_BANK />
    <BANK_BRANCH />
    <ADDR_NO />
   </BANKADDRESS>
   <BANKDETAIL>
    <CREAT_DATE>1999-04-01</CREAT_DATE>
    <CREATOR>&lt;UEBERNAHME&gt;</CREATOR>
    <METHOD />
    <FORMATTING />
    <BANK_DELETE />
   </BANKDETAIL>
   <RETURN>
    <TYPE />
    <ID />
    <NUMBER>000</NUMBER>
    <MESSAGE />
    <LOG_NO />
    <LOG_MSG_NO>000000</LOG_MSG_NO>
    <MESSAGE_V1 />
    <MESSAGE_V2 />
    <MESSAGE_V3 />
    <MESSAGE_V4 />
    <PARAMETER />
    <ROW>0</ROW>
    <FIELD />
    <SYSTEM />
   </RETURN>
  </asx:values>
 </asx:abap>
```

3.3.4.2 CREATE EPO RUNTIME SERVICE

The service creation is described in Definition of Services (Inbound & Outbound).

Area menu: *EPO XML Connector Configuration → Maintain Services EPO XML Connector*
Transaction: */EPO1/SERVICES12*

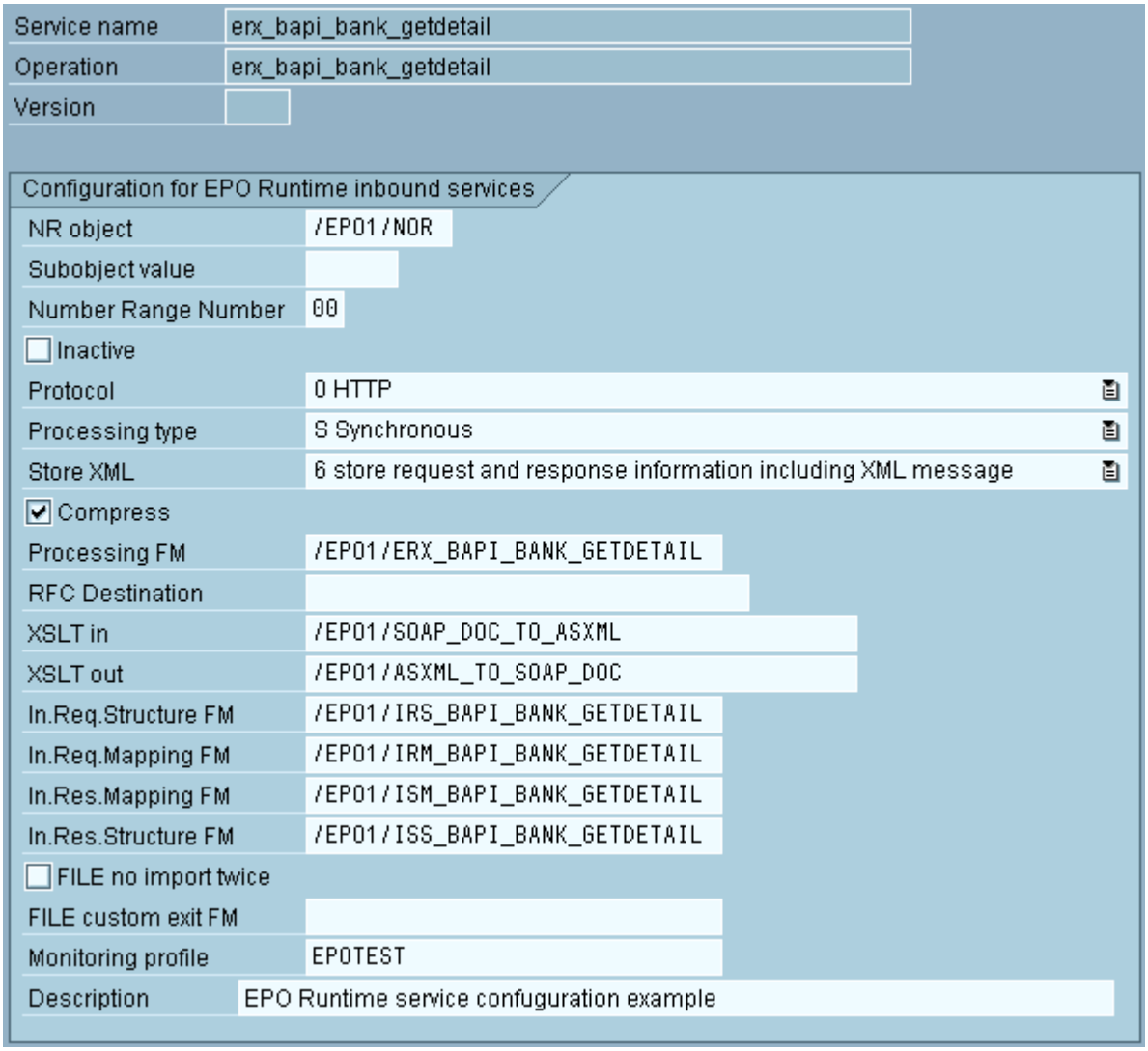
Image: EPO Runtime service example



3.3.4.3 CONFIGURE EPO RUNTIME SERVICE

Area menu: *EPO XML Connector Configuration → Inbound service configuration (SAP Services) → EPO runtime → In: EPO runtime maintain service configuration*
Transaction: */EPO1/EPORTIN12*

Image: EPO runtime configuration example



Service name: The name of service as it is set in service table described in section 3.1.

Operation: Operation of the service - you can create / use more operations of one service or leave it empty.

Version: Version of the service / operation. Using the version parameter you can implement changes to existing services and handle new and old versions of XML requests.

NR object: Number range object used for creating transaction identifiers (TransactionID). You can define your own object in transaction SNRO or use default "/EPO1/NOR".

Subobject value: Number range sub object value for the configuration.

No Range No: Number range number: It is used to create a unique "TransactionID" for each service call. It is used throughout the EPO XML Connector for requests and responses when storing. Creation of number ranges is described in section 2.2.

Protocol: Protocol used to acquire request XML. Default is HTTP. When you set this to FILE, you can get the request from file(s). In that case you will also need to fill the fields 'FILE directory' and 'FILE name' in service's settings.

Processing type: Synchronous – the processing is done when the request is received.
Asynchronous – nothing is done processing-wise but you can configure the connector to save the xml message and reprocess it later on (storing is described below in Store XML field).

Store XML: This field offers several possibilities for storing HTTP request and XML message:

- *0 Do not store any information:* Obviously, the connector does not store any information in this case. Suitable for synchronous processing when you do not want to keep any data. Do not use for asynchronous services or for disabling them.

- *1 Store request information: (only table /epo1/xmlhead):* The connector in this case stores everything except XML message itself, but for (in this inbound case) request only. This option you can use for logging HTTP requests to your service when processing synchronously, but beware, you won't be able to reprocess the request when using asynchronous processing.

- *2 Store request information including XML message:* The request in this case is stored including the XML message and you are able to reprocess it. This is the solution for asynchronous processing when you do not need to log or store responses. You can use this also when processing synchronously to be able to reprocess the request in error cases.

- *3 Store response information (only table /epo1/xmlhead):* Response logging could be the name for this option. The connector stores response information except the XML which is given out. The request message itself is not stored after processing.

- *4 Store response information including XML message:* Complete response storing is done when you choose this option. You can choose it to be able to check the responses of your service.

- *5 Store request and response information (only /epo1/xmlhead):* Use this option when you want to be able to check both request and response parts of transactions of your service but you don't care about the data transferred.

- *6 Store request and response information including XML message:* Finally, this is the "keep everything" choice. Use it when you require total control over your service.

Note: Depending on service usage, stored XML messages could occupy quite some database space. They are stored in database table /EPO1/XMLDATA.

Compress:	If you check this checkbox, the XML messages are compressed before storing (if configured to store).
Processing FM:	Processing function module: This is the function module which is called to process the request. Function module structure (interface) and development of those function modules is described below in detail.
RFC Destination:	The processing FM is called using RFC destination set in this field, otherwise the FM is called locally.
XSLT in:	XSLT transformation of request XML message. It takes part in request handling right after the request XML message is stored (if configured to do so) and before the message is processed (by calling processing function module).
XSLT out:	XSLT transformation of response XML message. This happens after the message is stored, before sending out.
In.Req.Structure FM *In.Req.Mapping FM* *In.Res.Mapping FM* *In.Res.Mapping FM*	Section 1.4.3 of this document describes these fields in detail.
FILE no import twice:	FILE protocol only. Set this to disable importing the same file twice for the configuration, which is done by storing filename in /epo1/files.
FILE custom exit FM:	FILE protocol only. User-exit function module which is called after the file is read and before it is processed. Interface of this function module can be found in example /epo1/file_in_user_exit. You can use this FM to rename the file after it has been successfully read into the EPO XML Connector. Also you can disable processing of the file, if renaming fails (e.g. because file is still written or changed).
Monitoring profile:	This field sets monitoring profile for the service. Monitoring is closely described in section 5.
Description:	Describe the version, operation, service or anything, but you can leave it empty if you don't need it.

3.3.4.4 CREATE PROCESSING FUNCTION MODULE

Processing function modules are used in the EPO runtime for

- receiving the XML request
- transforming it into ABAP data structures
- calling the function (BAPI, Call transaction BI, RFBI* program, IDOC, …)
- preparing the response
- transforming the response into XML

All processing function modules must have the same interface, thus you can copy an example function module. We strongly recommend doing so.
Use transaction SE37 or SE80 to implement it.
Your own processing function modules will be in the Z* or Y* name space.

Processing function module example (naming convention of example function modules for EPO Runtime is /epo1/erx*).

```
function /epo1/erx_bapi_bank_getdetail .
"----------------------------------------------------------------------
""Local Interface:
"  IMPORTING
"     VALUE(I_REQUESTXML) TYPE  XSTRING
"     REFERENCE(I_SERVICENAME) TYPE  /EPO1/SERVICE OPTIONAL
"     REFERENCE(I_OPERATION) TYPE  /EPO1/OPERATION OPTIONAL
"     REFERENCE(I_VERSION) TYPE  /EPO1/VERSION OPTIONAL
"     REFERENCE(I_TRANSACTIONID) TYPE  /EPO1/TRANSACTIONID OPTIONAL
"     REFERENCE(I_MESSAGEDIRECTION) TYPE  /EPO1/MESSAGEDIRECTION
"       OPTIONAL
"  EXPORTING
"     REFERENCE(E_XML) TYPE  XSTRING
"     REFERENCE(E_RESPONSEMESSAGE) TYPE  /EPO1/MESSAGE
"  CHANGING
"     REFERENCE(C_REQUESTSTATUS) TYPE  /EPO1/STATUS OPTIONAL
"     REFERENCE(C_REQUESTMESSAGE) TYPE  /EPO1/MESSAGE OPTIONAL
"     REFERENCE(C_FKEY1) TYPE  /EPO1/FKEY1 OPTIONAL
"     REFERENCE(C_FKEY2) TYPE  /EPO1/FKEY2 OPTIONAL
"     REFERENCE(C_FKEY3) TYPE  /EPO1/FKEY3 OPTIONAL
"     REFERENCE(C_FKEY4) TYPE  /EPO1/FKEY4 OPTIONAL
"----------------------------------------------------------------------

*&---------------------------------------------------------------------*
*& Company:   EPO Consulting
*
*& IP Rights: Intellectual Property Rights and all other rights are
*&            held by EPO Consulting
*&            Copying or Modifying this program is only allowed with
*&            written consent of EPO Consulting.
*& Author:    WK
*& Date:      March 2007
*&---------------------------------------------------------------------*

  data: xml_bankcountry type string.
  data: xml_bankkey     type string.
  data: l_bankkey type bapi1011_key-bank_key.
  data: l_bankcountry type bapi1011_key-bank_ctry.

  data: l_bank_address type bapi1011_address.
```

```
  data: l_bank_detail  type bapi1011_detail.
  data: l_return type bapiret2.


* Transform asXML to ABAP variables
  try.
*      CALL TRANSFORMATION id
*      SOURCE
*        XML           i_requestxml
*      RESULT
*        xmlelemtent1 = abapvariable1
*        xmlelemtent2 = abapvariable1
*      Tip 1: define abapvariable as strings. This avoids conversion
*             errors, when the XML element contents is too long
*      Tip 2: XML elements "tables" can passed to internal ABAP tables

      call transformation id
      source
        xml           i_requestxml
      result
        bankcountry = xml_bankcountry
        bankkey     = xml_bankkey.

    catch cx_root.
      c_requeststatus = '51'.
      c_requestmessage = 'XML wrong format'.
*      write your own error message here
      exit.
  endtry.

  if xml_bankcountry is initial or xml_bankkey is initial.
    exit.
  endif.

  l_bankcountry = xml_bankcountry.
  l_bankkey     = xml_bankkey.

  call function 'BAPI_BANK_GETDETAIL'
    exporting
      bankcountry  = l_bankcountry
      bankkey      = l_bankkey
    importing
      bank_address = l_bank_address
      bank_detail  = l_bank_detail
      return       = l_return.

* 1. Fill Status Fields ****************************************
  if l_return-type = 'E'.
    c_requeststatus = '51'.
*    write your own error message here
  else.
    c_requeststatus = '53'.         "processed successfully
  endif.
  c_requestmessage = l_return-message.
* **************************************************************

* use call transformation ID instead!
* Transform asXML to ABAP variables
  try.
*      CALL TRANSFORMATION id
*      SOURCE
*        XML           i_requestxml
*      RESULT
```

```
*         xmlelemtent1 = abapvariable1
*         xmlelemtent2 = abapvariable1
*       Tip 1: define abapvariable as strings. This avoids conversion
*              errors, when the XML element contents is too long
*       Tip 2: XML elements "tables" can passed to internal ABAP tables

      call transformation id
      source
        bankaddress   = l_bank_address
        bankdetail    = l_bank_detail
        return        = l_return
      result
        xml             e_xml.

    catch cx_root.
      c_requeststatus  = '51'.
      c_requestmessage = 'XML wrong format'.
*     write your own error message here
  endtry.

endfunction.
```

3.3.4.5 Create WSDL for Service using XML Transmitter

To be able to use a web service in other programming environments you will need to provide a WSDL. The XML Transmitter (freeware from EPO Consulting) provides “Create WSDL” functionality.
The steps needed to create a WSDL are described here.

1. Step.
Define a request message on client side and response message on the server side. You can “generate” the response message by testing your EPO XML Connector service (use POST). Messages must be in SOAP format, it means, they must have format from the schema definition xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/.

Schema definition template for the request:

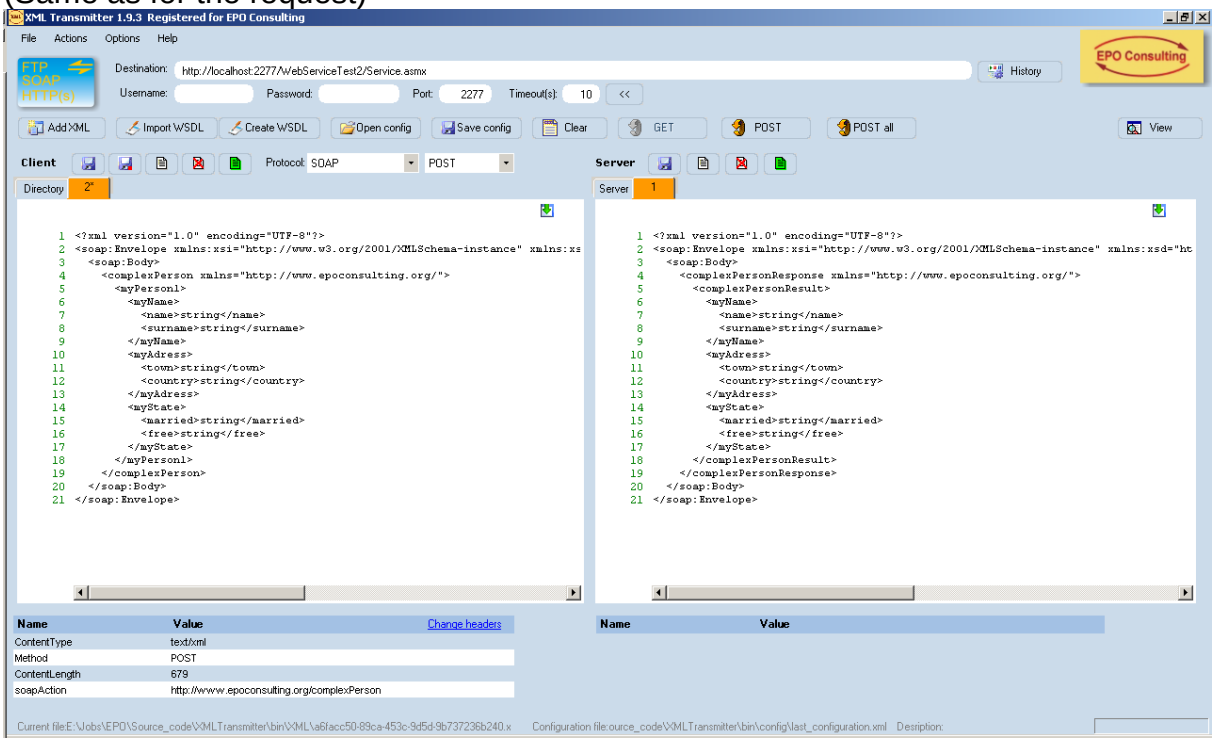
```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    < ... own structure ... />
  </soap:Body>
</soap:Envelope>
```

Schema definition for the response:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    < ... own structure ... />
```

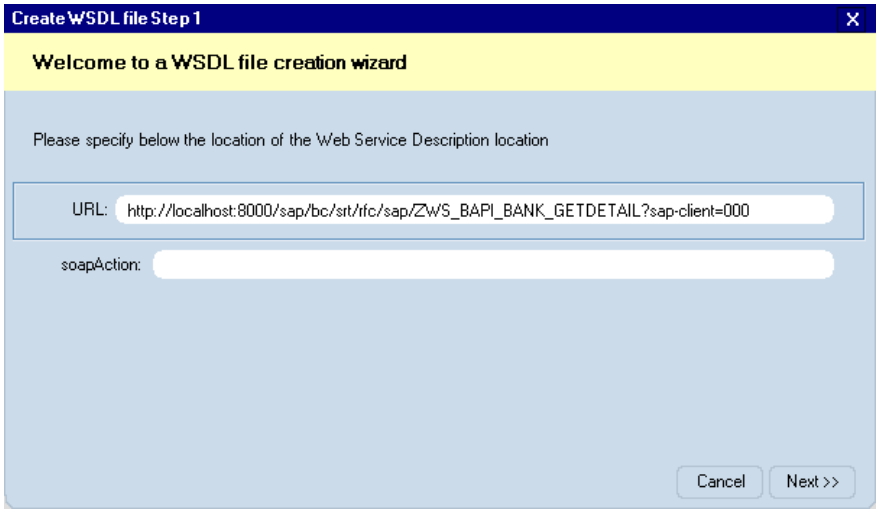
```
  </soap:Body>
</soap:Envelope>
```

(Same as for the request)

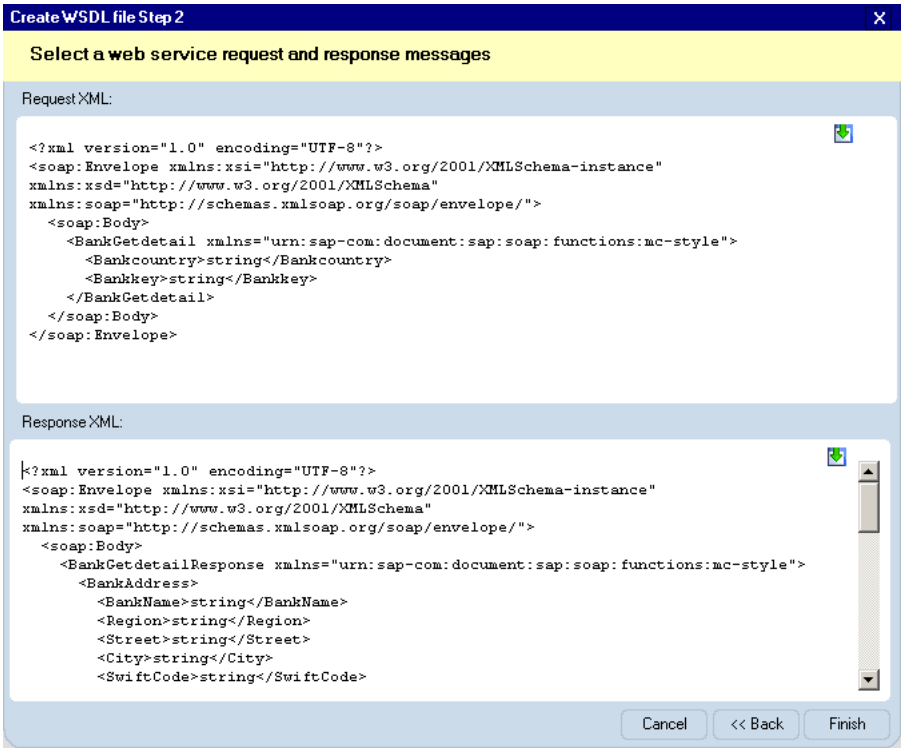


2. Step.

Make sure the URL and http headers are correct.
Then push the button Create WSDL file
Function “Create WSDL file” starts the wizard for the creation. In the first form window will be definition for the destination and definition for the SOAP action header.

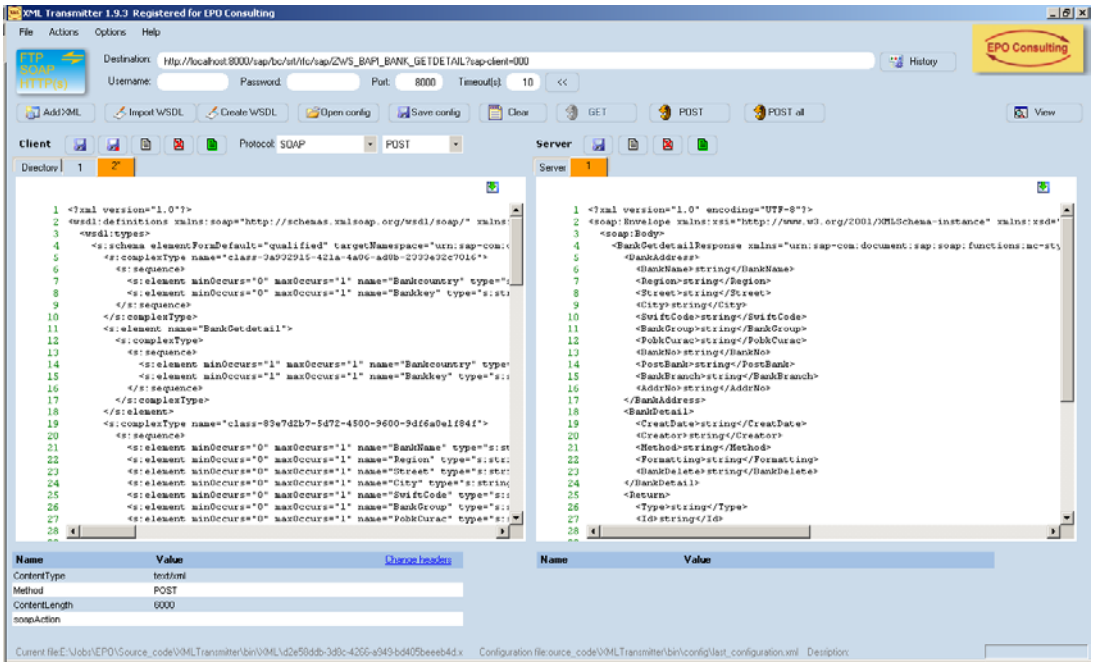


Second form shows definition for the request and response messages.



To finish the action press the “Finish” button.

The created WSDL file will be in the client side window. For checking the correctness of the WSDL file it is possible to start “Import WSDL” file. After import request and response messages will be created, which must be same as a source request and response messages.



3.3.5 CREATING EPO RUNTIME - INTEGRATION GUIDE FOR FILE PROTOCOL

Basically you can convert any HTTP runtime service (guide above) into FILE protocol one by changing the protocol in configuration of such service. The processing will then require the 'FILE directory' and 'FILE name' fields in service's settings to be filled with reasonable values. Also you can use 'FILE no import twice' check box in configuration to make such service not to process the same file twice when reading the same directory and/or you can rename, move or delete imported file in user exit function module. The interface of this user exit FM and example of how to rename the file using system command you can find in /EPO1/FILE_IN_USER_EXIT.

To automate such services, you can schedule program **/epo1/exc_fileruntime** as a SAP job.

3.3.6 TESTING AN EPO RUNTIME SERVICE

3.3.6.1 USING HTTP

In order to test an EPO runtime service, you must http post a request xml to the URL of the service. We recommend using XML Transmitter from EPO Consulting for testing such services.

Options for testing EPO runtime inbound services:

1. Post request XML messages from your integrated application
2. Post request XML messages from XML Transmitter
3. Post request XML messages from any other tool (SAP Java test tool, XML editor with POST functionality, …)
4. Upload a request XML file directly in SAP into the EPO XML Connector and process it (using upload and (re-)processing programs of the EPO XML Connector)
5. Use a stored XML message and (re-)process it (using the (re-)processing program of the EPO runtime of the EPO XML Connector).

Example: Testing service BAPI_BANK_GD_ERT with XML Transmitter:

"Configuration" (can be stored) with URL, user, password, http headers, XML request. With button "POST" the test will start.

Image: Testing EPO Runtime service using XML Transmitter – before post

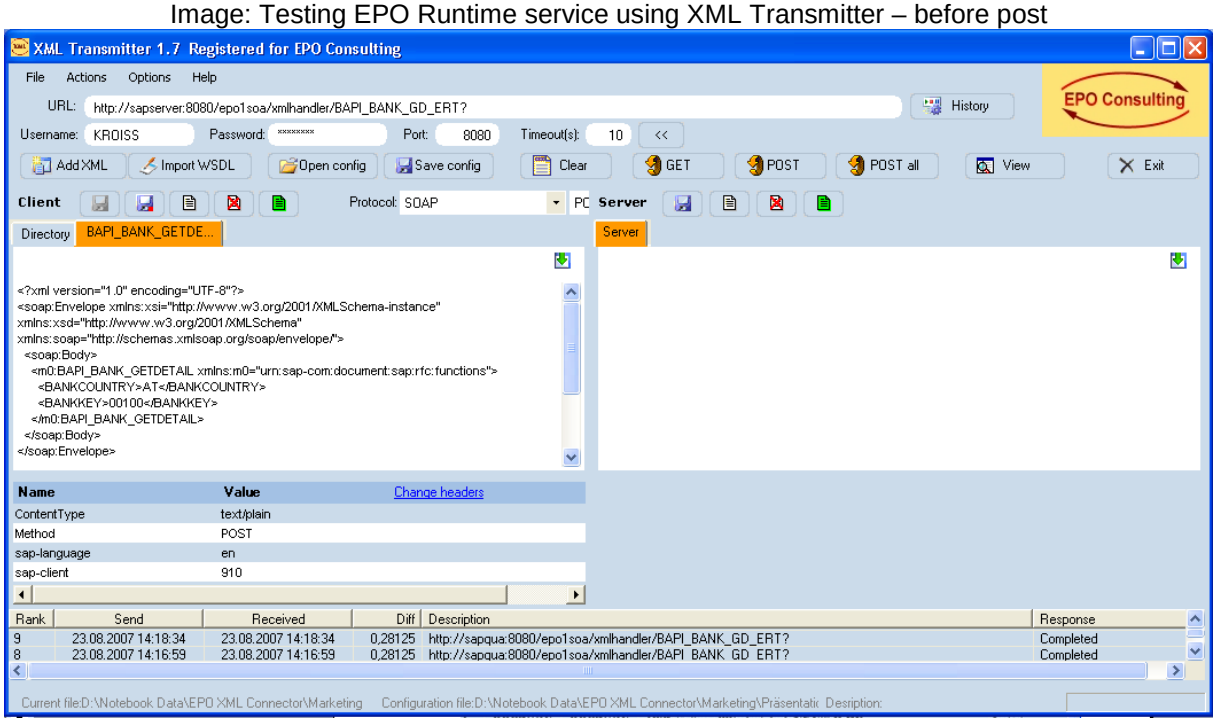
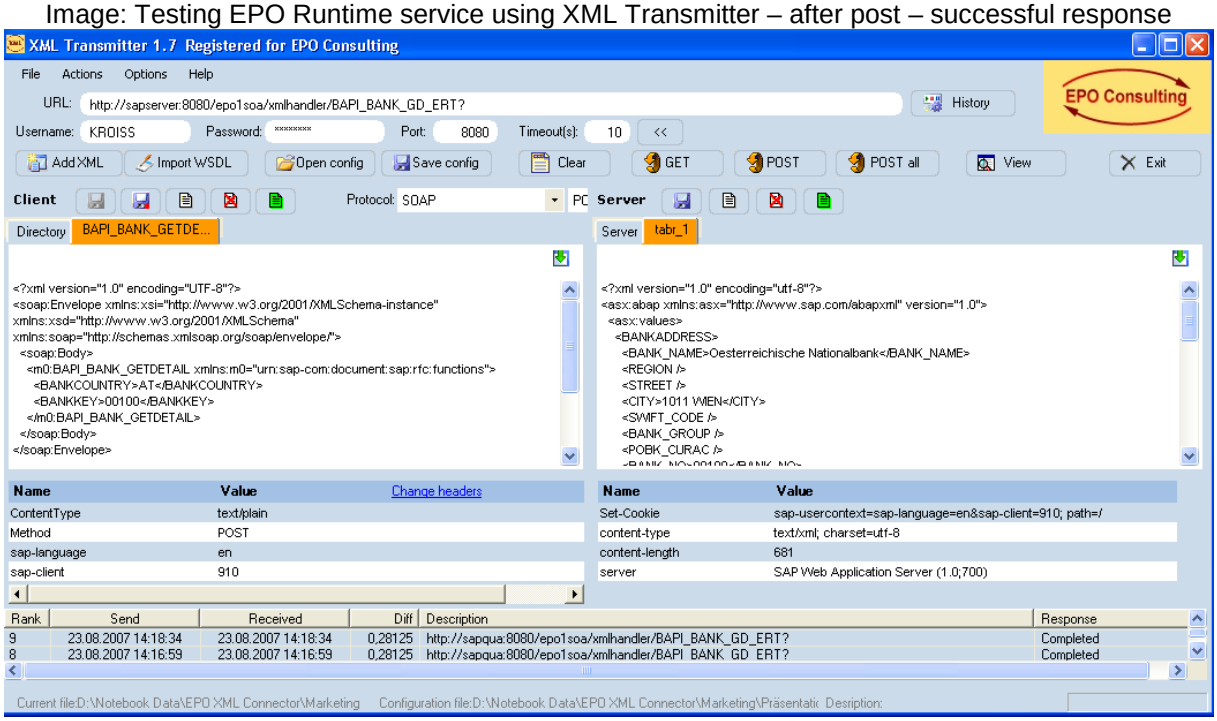


Image: Testing EPO Runtime service using XML Transmitter – after post – successful response



Debugging your service:

You can set an external break point in your processing function module. When posting a request XML, the SAP debugger will start. Make sure to set your “timeout” to a maximum. The SAP debugger will close on timeout.

Another easy way for testing is reprocessing the service with a stored message in SAP.

3.3.6.2 USING FILE, FTP

The file protocol can be tested by manually running /epo1/exc_fileruntime program (also in /EPO1/SOA menu *EPO XML Connector Configuration→ Inbound Service Configuration→ EPO Runtime → /EPO1/FILERT – In: FILE protocol runtime program*) and debugging it if necessary.

3.3.7 EPO RUNTIME ERROR XML MESSAGE

In case that you send a wrong XML request, use non-existing service (example) or something else is not what it should be, you will receive an error XML message generated by the EPO XML Connector with the following structure. You will only receive this message, if there is no response message from the processing function module.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <STATUSINFO>
      <CODE>900</CODE>
      <TYPE>E</TYPE>
      <SUBJECT>Service name not found in /EPO1/SERVICES</SUBJECT>
      <DESCRIPTION>ZEPO1_BAPI_BANK_GETDETAIL not configured in table
/EPO1/SERVICES</DESCRIPTION>
      <TRANSACTIONID>
      </TRANSACTIONID>
    </STATUSINFO>
  </soap:Body>
</soap:Envelope>
```

Note: Quite obviously this does not apply for FILE protocol, where there is no response sent back. You can see the job log for errors when using file protocol.

3.3.8 EPO RUNTIME EXAMPLE SERVICES

There are many different examples available for using EPO runtime. Examples are delivered as processing function modules for EPO runtime. All examples are delivered commented. You can copy it and create your own integration. If you want to use an example function module (e.g. for demonstration or first connection tests), you can uncomment it using the comment/uncomment program /EPO1/COMMENTUNCOMMENT (transaction /EPO1/TEMPLATESUNCOM, area menu: EPO XML Connector Administration).

All examples follow the naming convention /epo1/erx*.

You can find it using transaction SE37 or SE80.

List of processing function module examples:

Function group: /EPO1/ERX_EXAMPLES
/EPO1/ERX_BAPI_ACC_DOC_POST
/EPO1/ERX_BAPI_EMPLOYEE_GETD
/EPO1/ERX_BAPI_EMPLOYEE_GETD_T

/EPO1/ERX_FB01_RFBIBL00

Function group: /EPO1/ERX_EXAMPLES_MINISAP
/EPO1/ERX_BAPI_BANK_CREATE_T
/EPO1/ERX_BAPI_BANK_GETDETAIL

Function group: /EPO1/MBPM_PROCESSINGFM
/EPO1/ERX_MBPM_BAPI_ACC_DOC_PO
/EPO1/ERX_MBPM_BAPI_AGR_ASSIGN
/EPO1/ERX_MBPM_BAPI_BANK_CREA
/EPO1/ERX_MBPM_BAPI_BANK_GETL
/EPO1/ERX_MBPM_BAPI_USER_GETD

3.4 INBOUND: SAP RUNTIME

The SAP runtime takes advantage of the SAP standard SOAP web service runtime of the SAP NetWeaver Application server. You can publish any function module as web service in SAP standard and use it then in the SAP runtime of the EPO XML Connector.
In the SAP service tree (transaction SICF) it is located in sap_default_host/epo1soa/srthandler. The call of such an inbound service is done by posting (HTTP POST) a XML request message to this handler.

Example call: http://vepo2005:8000/epo1soa/srthandler/BAPI_BANK_GD?
Explanation of this call:
The host “vepo2005” and port “8000” are from the SAP WAS.
“/epo1soa/srthandler/” is the SAP runtime HTTP handler on SAP WAS.
The URI “BAPI_BANK_GD” is the service name, which must be configured.

Important http headers:
SOAPaction (optional): The operation of the service.
Hint: The operation of a service can also be defined in the XML of the service itself (first child element of <soap:Body>
Sap-client (optional): SAP logon client (Web logon, not an RFC logon)
Sap-language (optional): Logon language to SAP
ContentType: For example text/xml or text/plain
ContentLength: Normally automatically added
Method: POST

3.4.1 SAP RUNTIME (WEB SERVICE SOAP) - INBOUND

The SAP runtime provides a web service handler identical to SAP standard SRT (SOAP runtime), but it includes more features like enabling you to use asynchronous processing, full control over service processing, transformation of XML messages and file handling programs. Implementation of that integration is fairly simple and not much different to publish a SAP standard web service: You must publish a web service from a RFC-enabled function module or BAPI and put an alias (in SICF) under the srthandler service, release the alias (WSCONFIG) and configure the EPO XML Connector service. This enhances the standard SAP web service functionality with all functions of the EPO XML Connector (store, log, reprocess, XSLT transform,).

The SOA runtime is one option creating an inbound integration with SAP. The other option for inbound integration is using the unique EPO runtime of the EPO XML Connector. You can decide service by service, which option is appropriate. We recommend using EPO runtime as it returns far better error messages.

Steps for creating inbound integrations using EPO runtime:

- Publish a web service (a BAPI or RFC function module)
- Create an alias in SICF and release the alias in WSCONFIG
- Configure the service in the EPO XML Connector
- Test your integration

Note: Web Service SOAP runtime is not available on WAS 6.20.

3.4.2 CREATING SAP RUNTIME - INTEGRATION GUIDE

You will find all relevant transactions in the area menu /EPO1/SOA7.

3.4.2.1 CREATE A WEB SERVICE FROM A FUNCTION MODULE

In this chapter it is described, how to publish a web service in SAP standard. You can also refer to SAP help (http://help.sap.com).

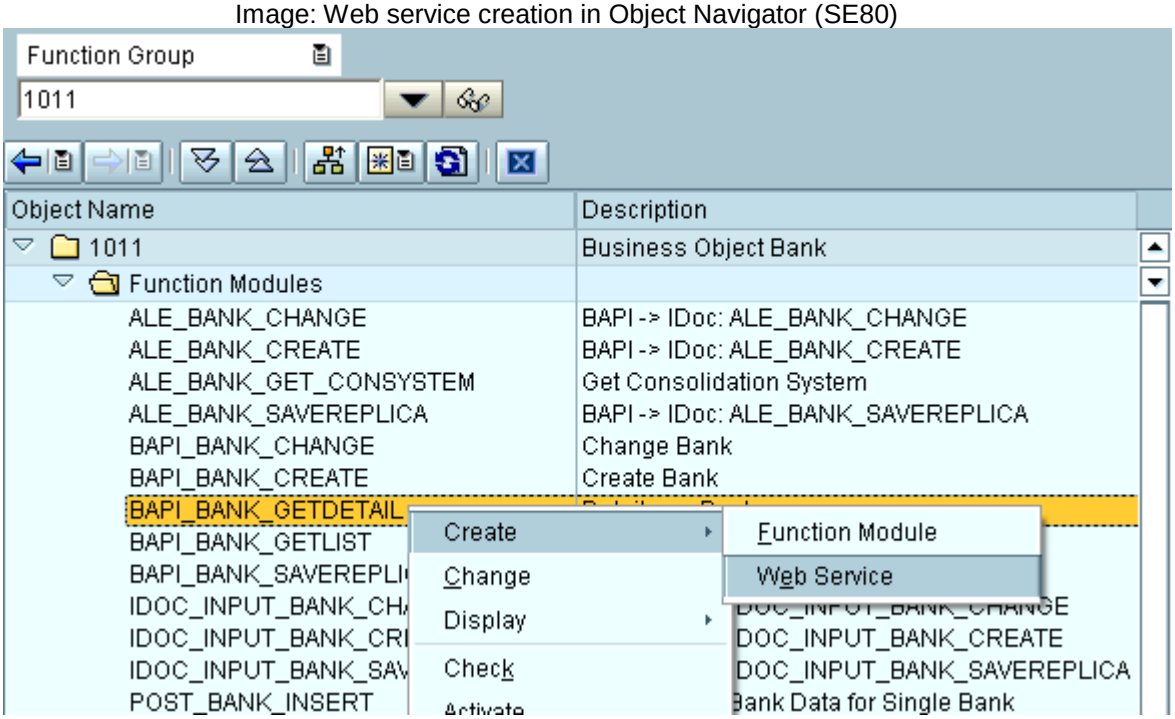
You can create Web service for RFC-enabled function modules, BAPIs and for function groups. The service definition is created using a wizard. Subsequently it can be checked and processed in the ABAP workbench.

1. In the Object Navigator (SE80), select the name of the package in which you want to create a Web service. From the context menu, choose *Create* → *Enterprise Services* → *Web Services* → *Web Service*.

   To create a Web service from a function group or function module, you can call the Creation Wizard from the Function Builder (SE37). Choose the function module, display it, and then choose *Utilities* → *More Utilities* → *Create* → *Enterprise Services* → *Web Services* → *Web Service*

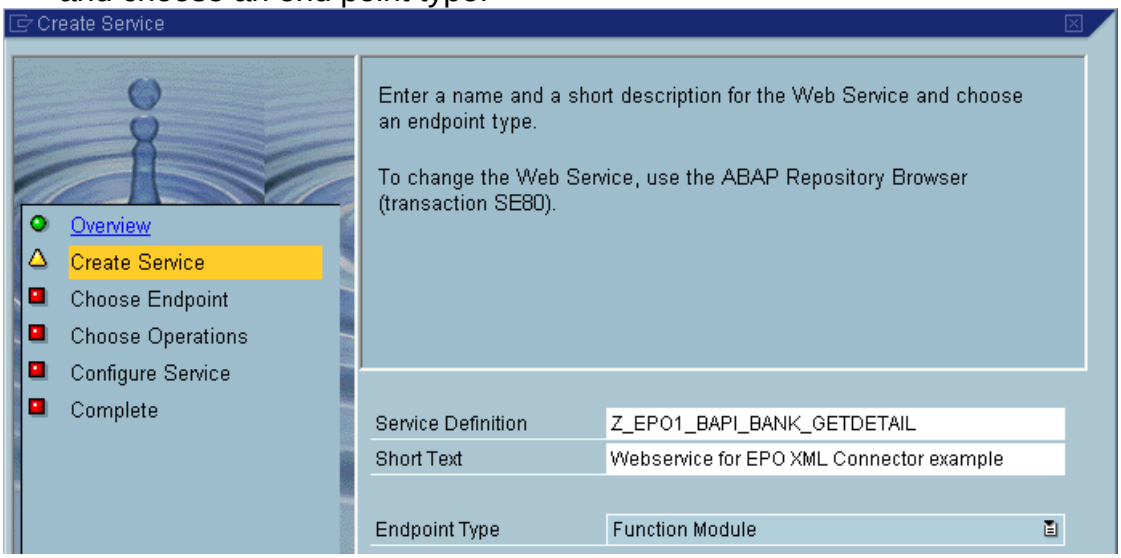
   Note: The function group must contain at least one RFC-enabled function module.

Image: Web service creation in Object Navigator (SE80)

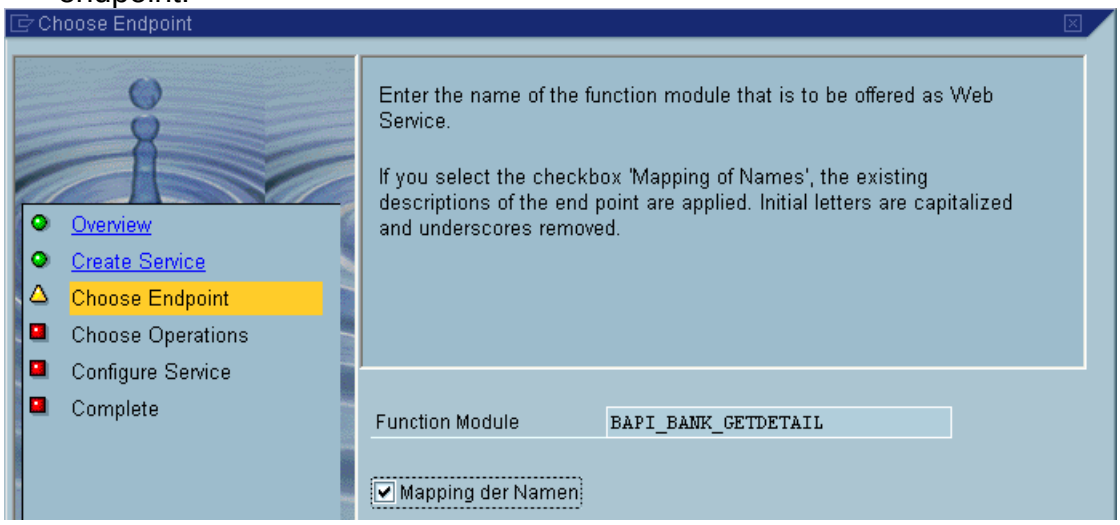


2. Perform the steps indicated in the wizard.

2.1 Create Service - Enter a name and description for the service definition and choose an end point type.



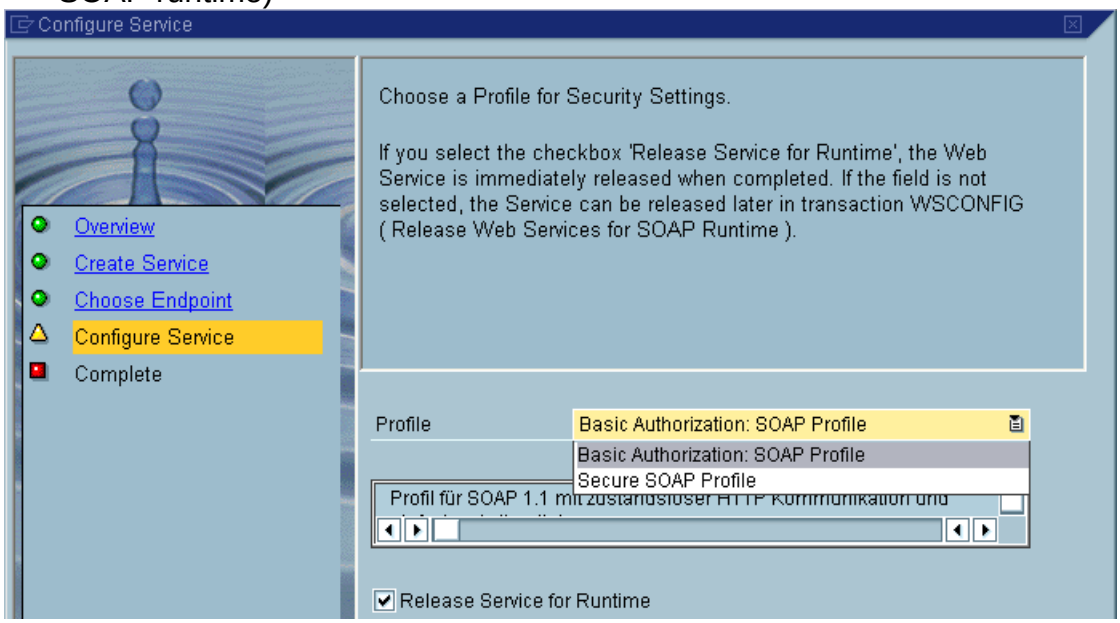
2.2 Choose Endpoint – Choose the object that you want to offer as a Web service. For business objects, enter the application. If the checkbox *Name Mapping* is checked, the wizard accepts the existing names for the end point. Initial letters are in uppercase and underscores are removed if this is not required, the service definition is created using the names in the endpoint.



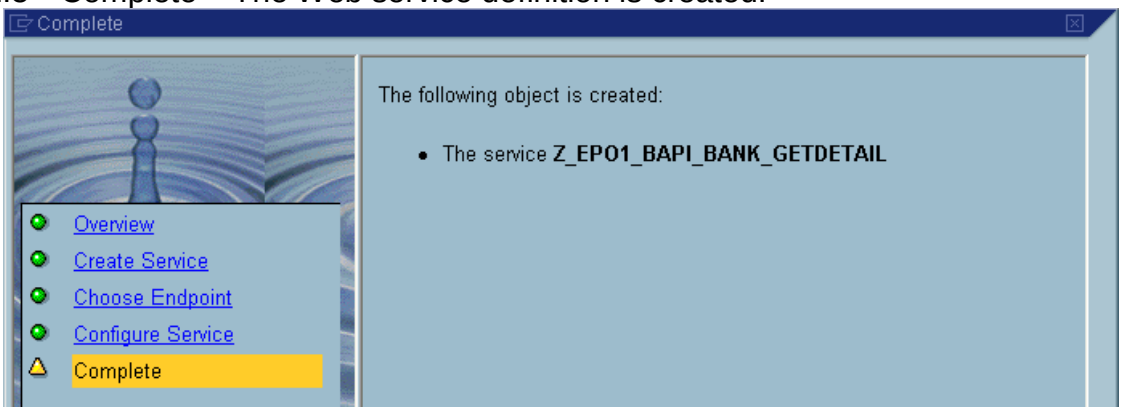
2.3 Choose Operations – For BAPIs and function groups, choose operations for which the Web service is to be created.

Note: There is no image in here because in this example service there is only one operation involved and that's why the wizard skipped this window.

2.4 Configure Services – The features you can assign here to the Web service relate to security of data transfer and type of communication. You can choose a predefined feature set from the profiles. By checking the "Release Service for Runtime" checkbox the Web service is release immediately when it is complete. Otherwise you need to release the Web service manually using WSCONFIG transaction (Release Web services for SOAP runtime)



2.5 Complete – The Web service definition is created.



3. Assign transport requests (including a Customizing request to configure the SOAP runtime) appropriate to the system configuration

When you finish Web service creation process, the service location in service tree should be
"Sap_default_host/sap/bc/srt/rfc/sap/Z_EPO1_BAPI_BANK_GETDETAIL".

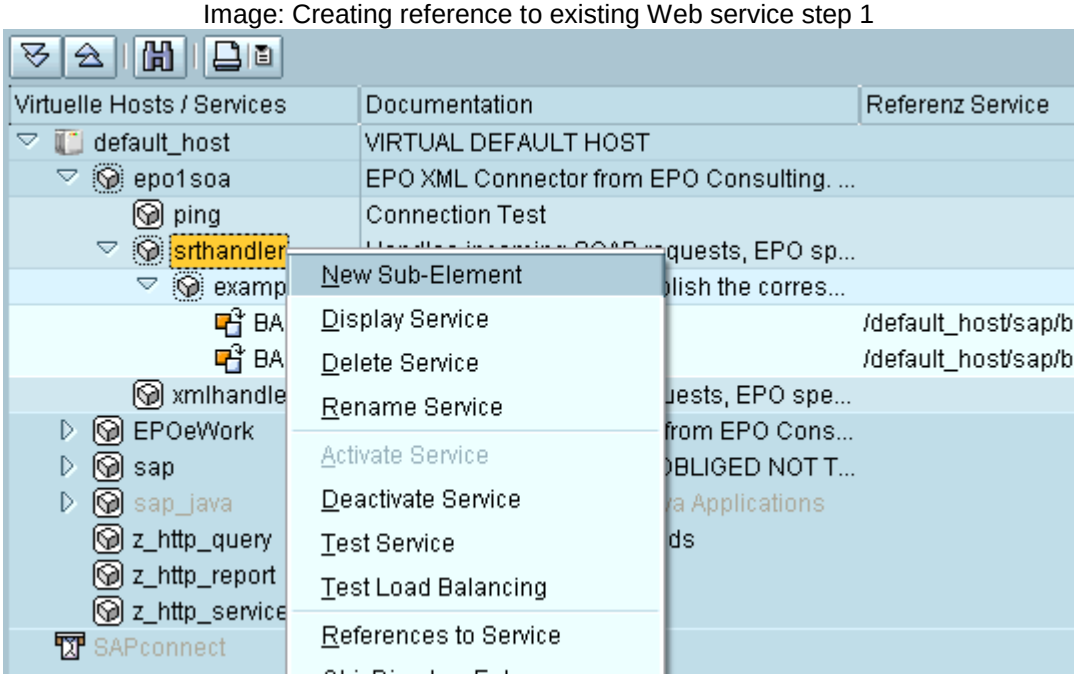
3.4.2.2 CREATE REFERENCE (ALIAS) TO THE WEB SERVICE UNDER THE SRTHANDLER

In order to use the SAP runtime handler of the EPO XML Connector you have to create a reference (an internal alias) to the service you created before under the “srthandler” service. You can create sub trees under the “srthandler” tree for organising your services.

How to create an internal alias service:

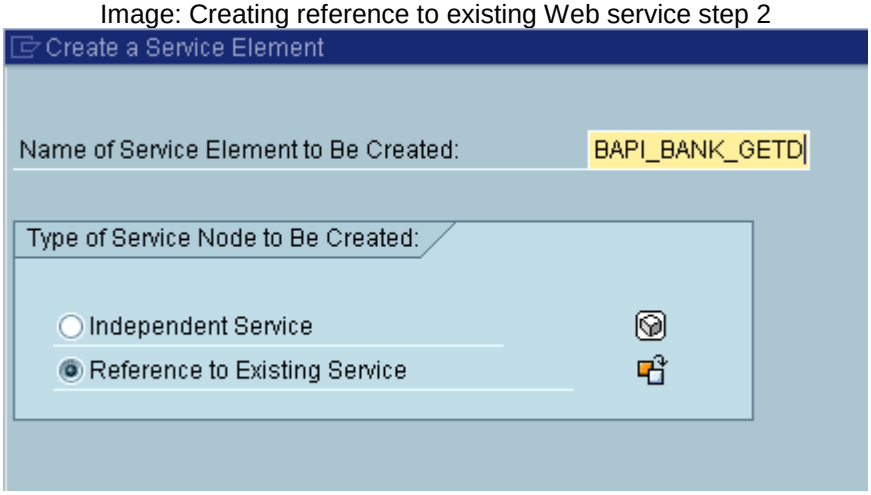
In transaction SICF (HTTP Service Hierarchy Maintenance (ICF)) you right click on the “srthandler” service in “epo1soa” and choose “New Sub-Element” (Image 10).
Tip: You could also use the wizard for creating services in SICF for creating this reference (alias). Menu: Service/Host – Wizard: Create Service

Image: Creating reference to existing Web service step 1



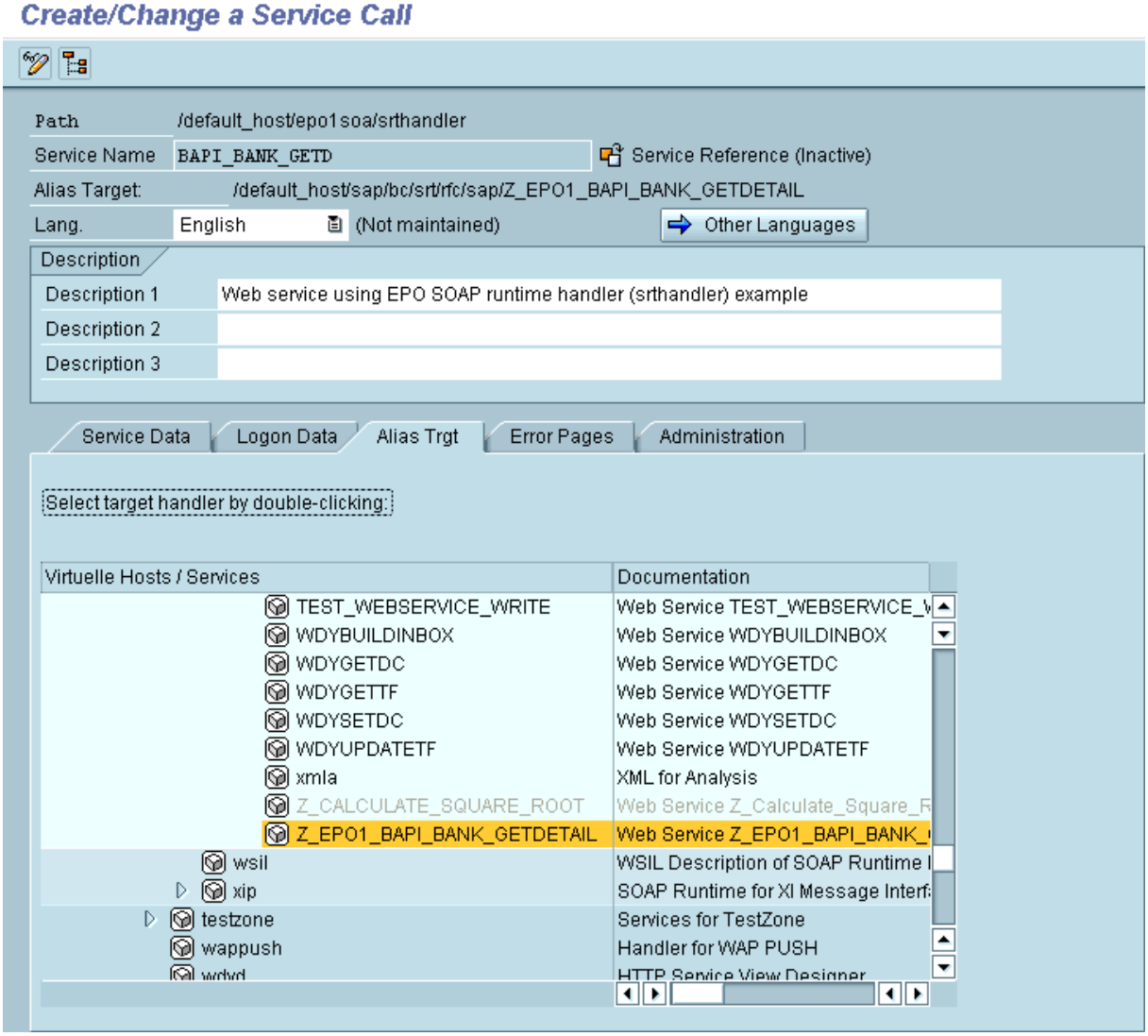
Then you name the service you are creating and more importantly you need to select “Reference to Existing Service” radio button.

Image: Creating reference to existing Web service step 2



The last step is to point the reference to the service you created. Click “Alias Trgt” tab and then find the service you wish to use. Services created in wizard as above are located in "*sap_default_host/sap/bc/srt/rfc/sap/".* You can configure the reference parameters the same way you can configure web services.

Image: Creating reference of Web service for EPO SOAP runtime handler

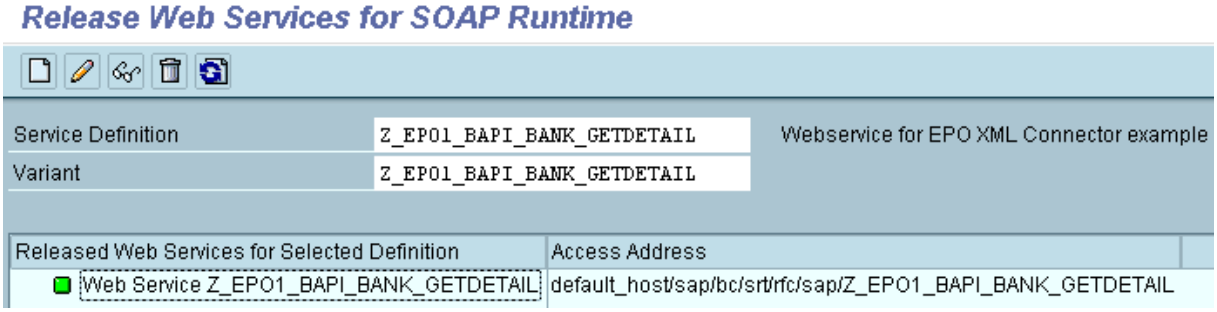


3.4.2.3 RELEASE THE REFERENCE USING TRANSACTION WSCONFIG

When creating Web service Z_EPO1_BAPI_BANK_GETDETAIL the release was done automatically by selecting “Release service for Runtime” checkbox in the creation wizard. For the reference you need to do the release manually.

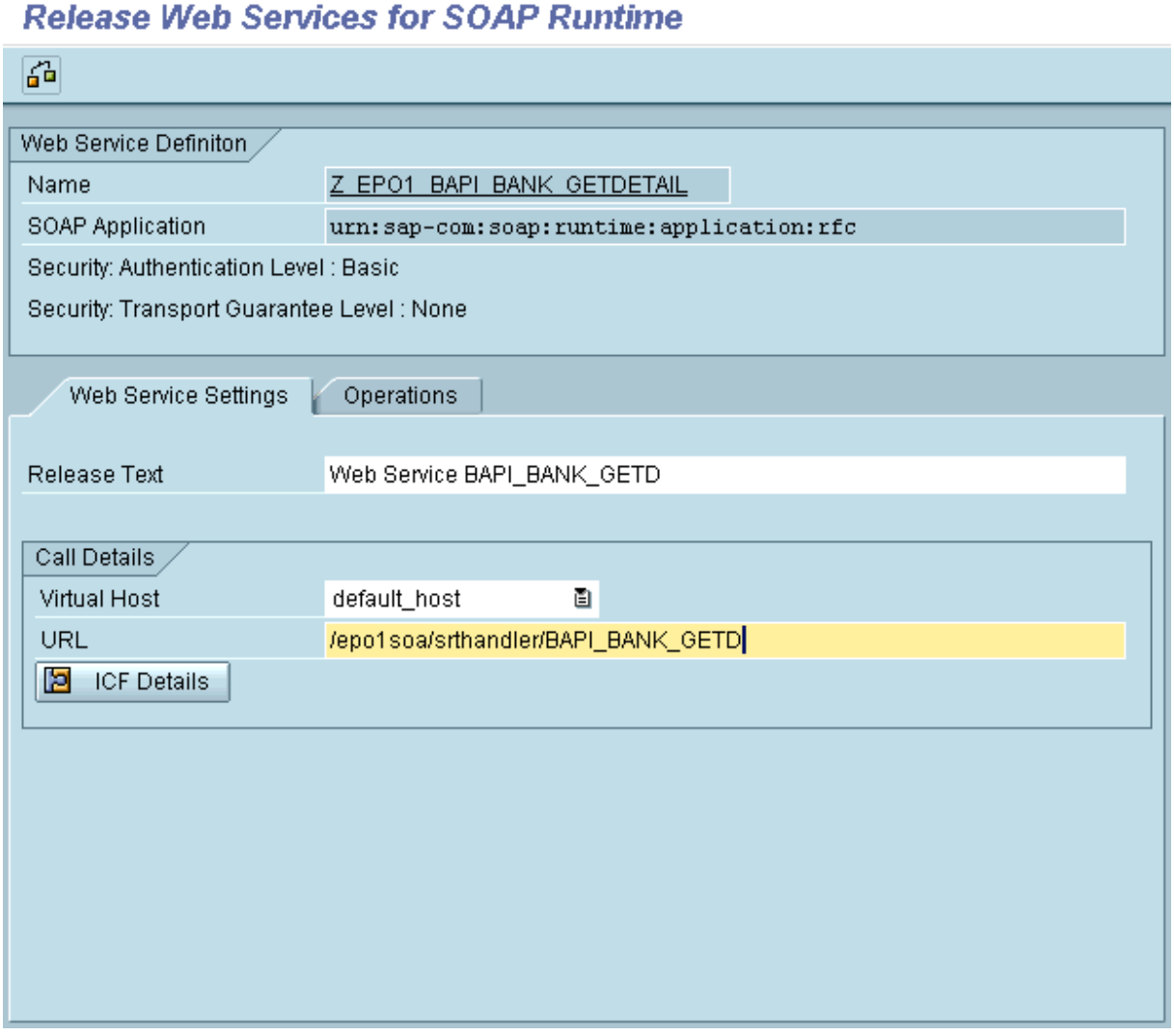
Transaction WSCONFIG:
Fill in the name of the Web service and variant (the same as Web service) into “Release Web Services for SOAP Runtime” program (transaction WSCONFIG), then press the “Create” button.

Image: Released Web service example



Then you need to change the URL to point to the reference you created before.

Image: Release reference to Web service example



Hint: If you forget to release the web service or the reference (alias), you will get an error message when testing your service “500 internal server error”.

3.4.2.4 CREATE SAP RUNTIME SERVICE FOR EPO XML CONNECTOR

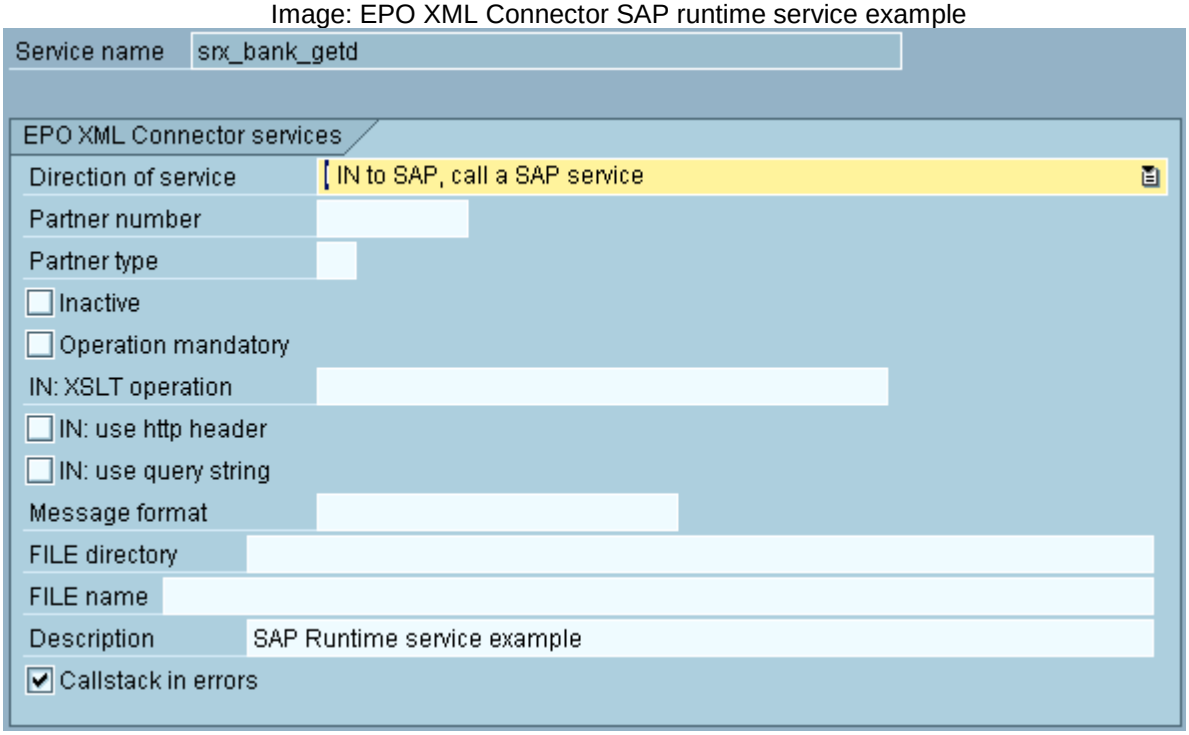
The service creation is described in section 3.1.

Area menu: *EPO XML Connector Configuration → Maintain Services EPO XML Connector*
Transaction: */EPO1/SERVICES12*

IMPORTANT: The service name MUST BE the name of the ALIAS (the reference) you created before in transaction SICF

For the example the alias was BAPI_BANK_GETD. It must be used as the name of the service in the EPO XML Connector. The service name is also part of the URL.

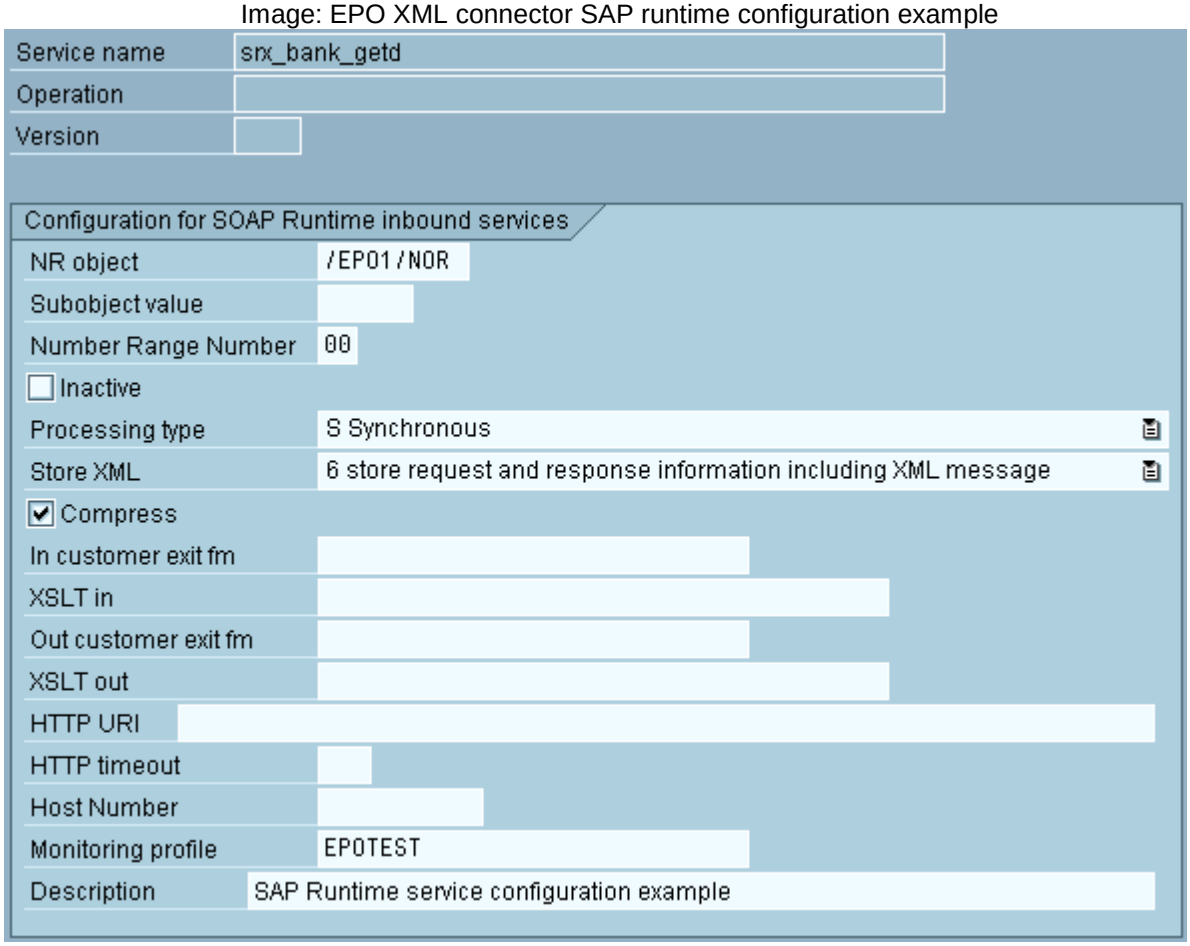
Image: EPO XML Connector SAP runtime service example



3.4.2.5 CONFIGURE SAP RUNTIME SERVICE

Area menu: *EPO XML Connector Configuration → Inbound service configuration (SAP Services) → SAP runtime (SAP Web Services) → In: Maintain SAP runtime web service configuration*
Transaction: */EPO1/WSIN12*

Image: EPO XML connector SAP runtime configuration example



Fields "Service name", "Operation", "Version", "NR object", "Subobject value", "Number Range Number", "Inactive", "Processing type", "Store XML", "Compress", "Monitoring profile" and "Description" are common for every EPO service configuration. For description of these fields please see section 3.2.3.3.

In customer exit fm: Function module which is called in pre-processing stage of the request message (inbound), right after the message is saved, before the "XSLT in" transformation takes place. As a template you can use /epo1/exit_requestxml function module provided in delivery.

XSLT in: XSLT transformation of request XML message. It takes part in request handling right after the "In customer exit fm" call and before the message is processed (by calling Web service).

Out customer exit fm: Function module called in post processing stage of the service after the response is stored and before the "XSLT out" transformation is done. You can copy template function module /epo1/exit_responsexml for this purpose.

XSLT out: XSLT transformation of response XML message. This happens after "Out customer exit fm" is called. Actually it is the last operation of the post processing method.

Path (Uri): The URI path to the service itself. It must be correct if you plan to use reprocessing functionality. The message is sent to the service using this URI when reprocessed. It is only used for reprocessing.

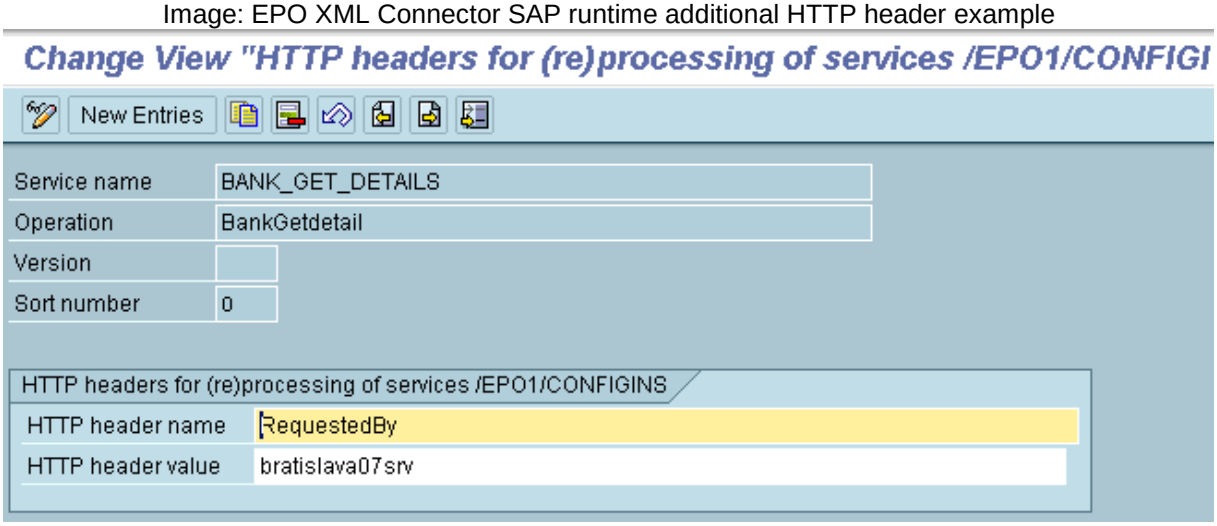
HTTP timeout: Timeout parameter for reprocessing of messages only. It's the amount of time in seconds for which the reprocessing program tries to send the request XML message and get the response to it.

Host number: SAP virtual host number. Default_host is used, if you leave it empty.

3.4.2.6 SET ADDITIONAL HTTP HEADERS FOR (RE-)PROCESSING

You can add any number of HTTP headers to the request message when (re)processing a stored message in SAP. Those HTTP headers are not used for synchronous services at the initial processing. They will be used only, when the request XML message is stored in SAP and the (re)processing is done with transaction /EPO1/WSINPROC = In: (Re) Process SAP runtime XML message.

Image: EPO XML Connector SAP runtime additional HTTP header example



3.4.2.7 WSDL OF THE WEB SERVICE

For deploying the web service on the integrated system, you will need the WSDL of the web service. Here it is described, how you get the WSDL in SAP standard. Please note, that you can get the WSDL only from the originally published web service. It is not possible to get the WSDL from the reference (alias) you created in the tree /epo1soa/srthandler/.

Another option to generate a WSDL is using the XML Transmitter. EPO Consulting XML Transmitter allows you to create a WSDL from a request and response XML message, which can be used directly in Microsoft .NET and Java developments. All details about this you can find in chapter 3.2.3.5.

Transaction /nWSADMIN

Open the tree in WSADMIN and select the web service. Press the button WSDL and generate the WSDL. You can choose between different options for the WSDL.

Image: Generating WSDL for SAP runtime service – WSDL button

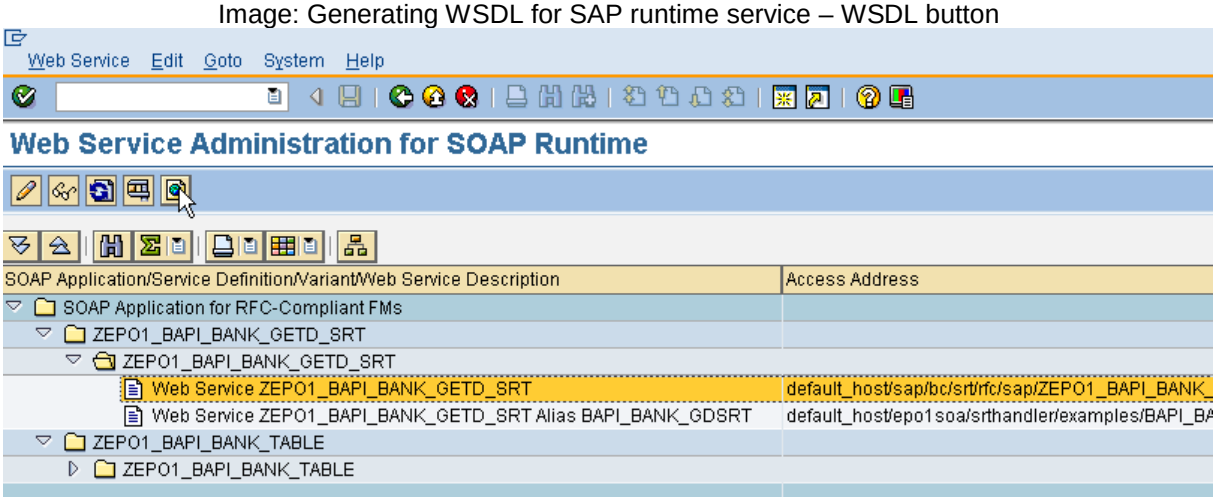
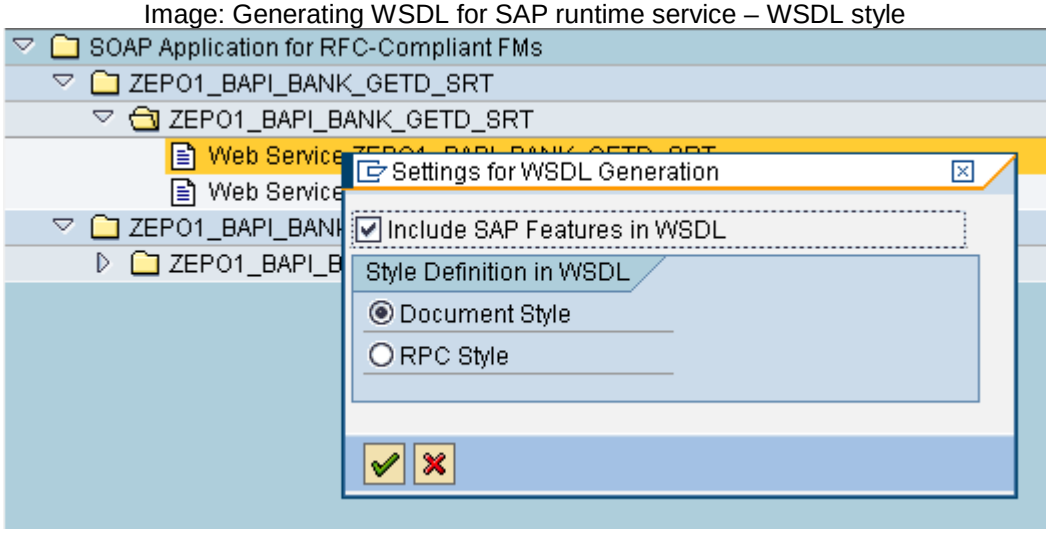
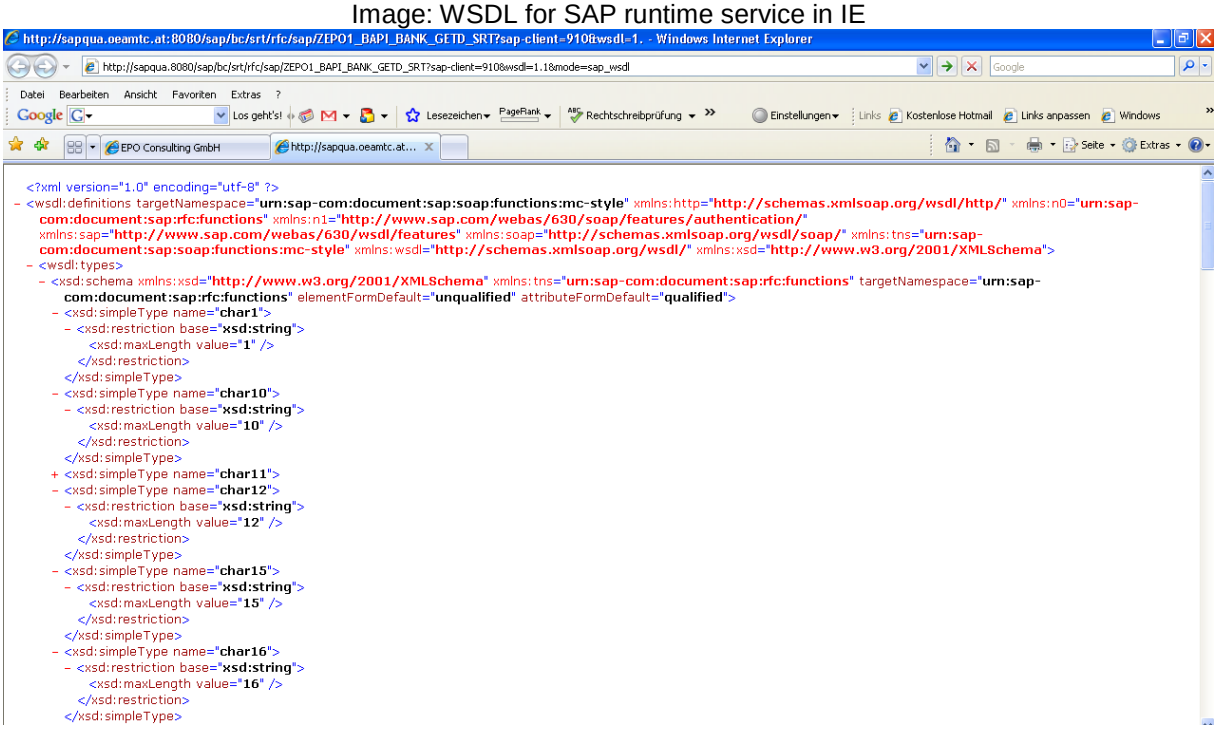


Image: Generating WSDL for SAP runtime service – WSDL style



This should open your web browser. You need to login using your SAP user and password. Then you can save the WSDL file from here (Menu: File – save as). Another option is to use the URL, which you can see in your web browser now.

Image: WSDL for SAP runtime service in IE



Note: For consuming this web service with the EPO XML Connector, you must replace the URL given in this WSDL with the URL of the EPO XML Connector sap_default_host/epo1soa/xmlhandler/<service name>? If you are using the URL from the WSDL, your web service should also work – only you loose all features of the EPO XML Connector (storing the message, applying a XSLT, (re-)processing etc.).

3.4.2.8 TESTING A SAP RUNTIME SERVICE

In order to test an SAP runtime service, you must http post a request xml to the URL of the service. We recommend using XML Transmitter from EPO Consulting for testing such services.

Options for testing SAP runtime inbound services:

1. Post request XML messages from your integrated application
2. Post request XML messages from XML Transmitter. We strongly recommend using the “Import WSDL” functionality to create a valid XML instance.
3. Post request XML messages from any other tool (SAP Java test tool, XML editor with POST functionality, …)
4. Upload a request XML file directly in SAP into the EPO XML Connector and process it (using upload and (re-)processing programs of the EPO XML Connector).
5. Use a stored XML message and (re-)process it (using the (re-)processing program of the EPO runtime of the EPO XML Connector).

Example: Testing service BAPI_BANK_GETD with XML Transmitter:

“Configuration” (can be stored) with URL, user, password, http headers, XML request. With button “POST” the test will start.

Image: Testing SAP Runtime service using XML Transmitter – before post

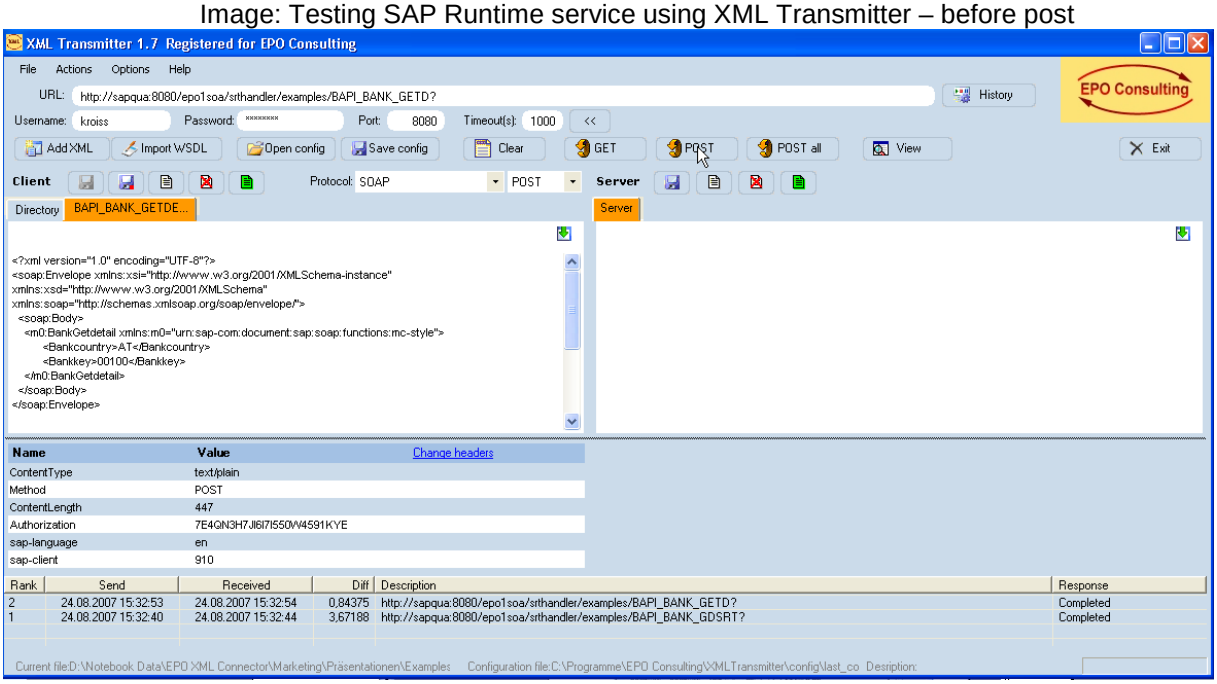
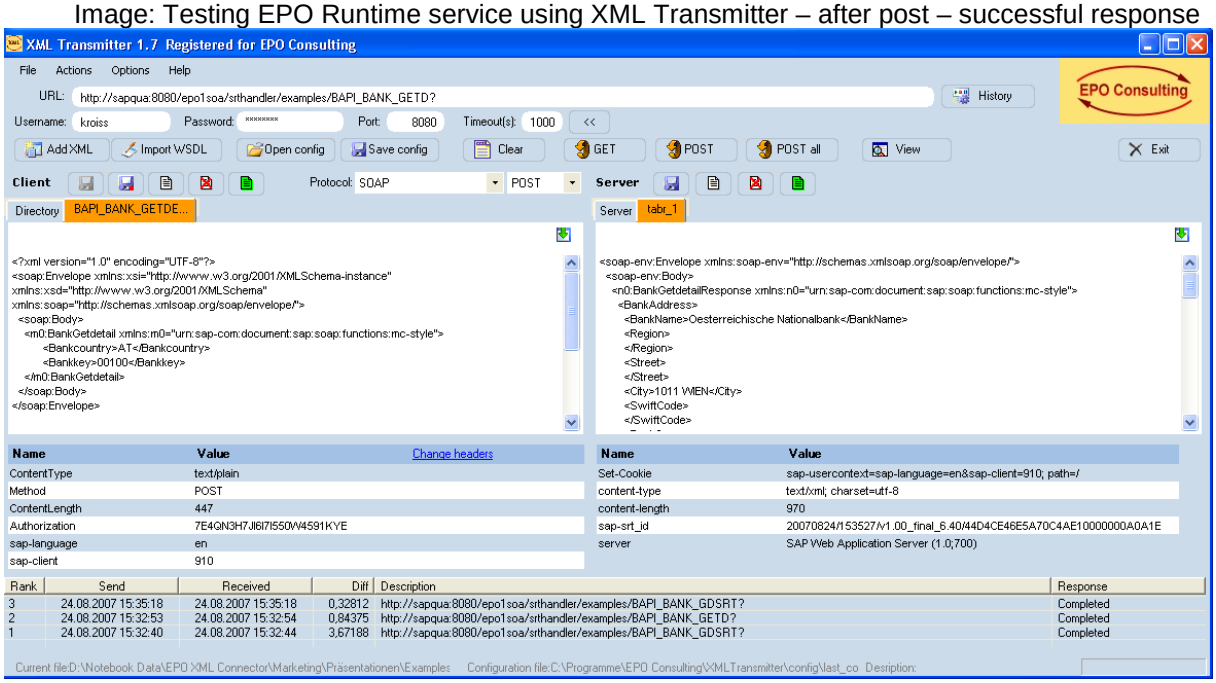


Image: Testing EPO Runtime service using XML Transmitter – after post – successful response



Common errors:

When testing your web service, you might encounter some error messages initially. Unfortunately you will get often the undistinguished error message "*The remote server returned an error: (500) Internal Server Error*.

This error message is raised by the SOAP runtime and it cannot be changed from the EPO XML Connector. The 2 most common reasons for this error message are:

- The URL (host, port and URI/tree) you are posting to, does not exists.
- The XML request message format is wrong (case sensitive, underscores are maybe removed). Create an example instance using the WSDL. (The XML Transmitter from EPO Consulting is tested with such WSDL.)

Tables of BAPIs/FM in WSDL and XML request:
Table parameters in function modules can be either import or export parameters. In the WSDL it is not defined, if a table parameter is an import or export parameter. When testing a SAP web service with table parameters you must always send the table XML elements with the request XML. For export tables these XML elements must be empty (e.g. <tablebapiret2/>).
Note: Table parameters should not be used anymore in function modules. Instead you should define a table type in the data dictionary and use this either in export or import of the function module interface.

Example for XML request of BAPI_USER_GET_DETAIL:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m0:UserGetDetail xmlns:m0="urn:sap-com:document:sap:soap:functions:mc-style">
      <Activitygroups />
      <Addcomrem />
      <Addfax />
      <Addpag />
      <Addprt />
      <Addrfc />
      <Addrml />
      <Addsmtp />
      <Addssf />
      <Addtel />
      <Addtlx />
      <Addttx />
      <Adduri />
      <Addx400 />
      <Extidhead />
      <Extidpart />
      <Groups />
      <Parameter />
      <Parameter1 />
      <Profiles />
      <Return />
      <Systems />
      <Uclasssys />
      <Username>KROISS</Username>
    </m0:UserGetDetail>
  </soap:Body>
</soap:Envelope>
```

Debugging your service:
Debugging of SAP web services is not useful. You might debug the customer exits of the EPO XML Connector, if you are using it.

3.4.2.9 SAP RUNTIME EXAMPLE SERVICES

There are no examples delivered with the EPO XML Connector for the SAP runtime. This is because you can publish any BAPI or RFC-enabled function module in SAP standard and use it directly with the SAP runtime of the EPO XML Connector.

3.5 OUTBOUND: EPO CLIENT

The EPO Client is an easy to use function module called "**/EPO1/EPOCLIENT**". When used for HTTP(s) protocol it dynamically creates a client proxy to be able to send given request XML message to a web service and receive response. Like all the other services you can use this one asynchronously, store or log and XSLT transform request and response messages. Using FILE protocol, it stores a file to the location specified in service's settings and with UM protocols you can send a SAP-Mail, E-Mail or other message to configured or dynamic list of recipients.

The unique EPO Client is one option creating an outbound integration with SAP. The other option for outbound integration is using a generated ABAP client proxy within the EPO XML Connector (see next chapter). You can decide service by service, which option is appropriate. We recommend using EPO Client as it works for any web service and is more flexible and easy to implement.

Usage of outbound EPO Client (compare with outbound generated ABAP client proxy)

- Integrations, where the XML format and structure is defined by a partner system (but does not have a WSDL or is not a full web service)
- Integrations, where SAP defines request and response XML messages (for outbound!)
- Integrations, where the request and response XML is relatively simple
- Integrations, where you do not want to use the overhead of generated ABAP proxies

Overview: Steps for creating outbound integrations using EPO client:

- Call the function module /epo1/epoclient (e.g. from your custom Z program)
- Configure the service in the EPO XML Connector
- Test your integration

3.5.1 USING HTTP(S)

EPO Client is simple to use function module /epo1/epoclient to which you pass the XML message you want to post to Web service. The Protocol field in service configuration is set to HTTP in this case.

3.5.2 USING FILE, FTP

You can use the same call as if using HTTP(s) protocol to make EPO client write a file to a location specified in service's fields. The Protocol field in service configuration must be set to FILE to make this happen. The file directory and name parameters are set in service's settings.

3.5.3 USING UM - SAP MAIL, E-MAIL VIA SCOT AND CUSTOMER DEFINED

Unified messaging (UM) protocols have been designed for sending messages using EPO Client function module to other common message end points – SAP Mail, E-Mail via SCOT and Customer defined (function module customer exit). XML data passed to the EPO Client FM could be sent within the message body or as an attachment. When sending attachment, the file naming routine from file protocol is used. File name and extension (message format) are taken from service settings and file name additions - client number (MANDT), transaction ID, date and time – are set based on setting in services configuration.

After sending the message, the success information or error message is given back to you in both e_callstatus and e_responsexml export parameters, e_responsexml contains transformed e_callstatus information.

The customer defined function module call is shown below; most of the parameters are taken from configuration as they are, recipients and CC recipients can be added dynamically from EPO Client FM import parameters; i_um_body, i_um_packing_list and i_um_att_cont are filled the same way like we use them with SO_OBJECT_SEND function module for first two UM protocols - SAP Mail and E-Mail via SCOT. The body already contains text from text object specified in configuration.

```
try.
    call function me->um_customexitfm
      exporting
        i_protocol        = me->protocol
        i_um_sender       = me->um_sender
        i_um_recipients   = me->um_recipients
        i_um_ccrecipients = me->um_ccrecipients
        i_um_subject      = me->um_subject
        i_um_textname     = me->um_textname
        i_um_textid       = me->um_textid
        i_um_contenttype  = me->um_contenttype
        i_um_attachtype   = me->um_attachtype
        i_um_body         = me->um_body
        i_um_packing_list = me->um_packing_list
        i_um_att_cont     = me->um_att_cont
        i_testmode        = me->testmode.
    exit.
  catch cx_root.                                        "#EC CATCH_ALL
    raise um_customexit_call.
endtry.
```

3.5.4 /EPO1/EPOCLIENT FUNCTION MODULE INTERFACE

Image: Function module /EPO1/EPOCLIENT interface

Import

- *i_requestxml* type xstring optional

  Request XML message can be left empty when reprocessing using i_transactionid. In all other cases the request XML (a valid XML string) must be given to this parameter.

- *i_servicename* type /epo1/service

  Service name from /epo1/services table. Please see section 3.4.4.1.

- *i_operation* type /epo1/operation optional

  Operation of the service is checked against service's configuration. This can be left empty, but in that case configuration with empty operation must exist in configuration table for used service. Please see section 3.4.4.2.

- *i_version* type /epo1/version optional

  Version of used service. Rules for this variable are the same as for i_operation.

- *i_fkey1 – 4* type /epo1/fkey1 – 4 optional

  The foreign key 1,2,3 and 4 are stored in header table with all the transaction information except the message itself, which is stored in data table, of course only if the service is set to store such information. These keys you can use to store additional information about specific transaction. Usage of such information can be variable e.g. logging, search capability, statistics etc.

- *i_reprocess* type flag optional

  Flag domain can have values ‘X’ or ‘ ‘. By setting this flag to ‘X’ you make the function module to reprocess stored XML message. You will need to set i_transactionid to TransactionID of transaction you wish to reprocess as well.

- *i_transactionid* type /epo1/transactionid optional

  TransactionID is unique identifier of EPO XML Connector transaction made from number range object (see section 2.2). You only need to set this parameter when you want to reprocess stored message by setting i_reprocess parameter to ‘X’.

- *i_nryear* type /epo1/nryear optional default '0000'

  Year parameter for number range object. Overrides actual date used by default.

- *i_path* type string optional

  This parameter overwrites ‘Path (URI)’ value taken from service configuration. See section 3.4.4.2.

- *i_httpaddress* type /epo1/httpheadertable optional

  Additional HTTP headers. This parameter is very useful when you need to set some HTTP header dynamically. These headers are added after the HTTP headers from static headers table (see section 3.4.4.3) so you can use it to overwrite static headers as well.

- *i_umrecipients* type /epo1/um_recipients optional

  Additional recipients for UM, semicolon separated list (string). These recipients are added after the recipients from EPO Client service configuration. Only used for UM protocols (SAP-Mail, E-Mail, or customer defined).

- *i_umccrecipients* type /epo1/um_ccrecipients optional

  Additional carbon copy recipients for UM. These CC recipients are added after the CC recipients from EPO Client service configuration. Only used for UM protocols (SAP-Mail, E-Mail, or customer defined).

Export

- ✓ *e_responsexml* type xstring

  Response XML message returned from the web service or the web server in error cases.

- ✓ *e_callstatus* type /epo1/callstatus

  Status of processing message with EPO XML Connector. It does only return the callstatus of the generated call in SAP (e.g. if connection cannot be established or path is wrong). It does not give any information, which is returned from the web service itself. For successful web service calls the callstatus code will be empty (check field callstatus-code).

  Structure description:

  - ✓ *code* type /epo1/statuscode (numc 3)
    callstatus with status code equal or greater than 200 is error message

  - ✓ *type* type /epo1/statustype (char 1)
    'E' = error message, 'I' = information, 'S' = success message, 'W' = warning message

  - *subject* type /epo1/statussubject (char 50)
    subject of callstatus message

  - *description* type /epo1/statusdesc (string)
    description of callstatus message

  - *TransactionID* type /epo1/transactionid (char 15)
    unique identifier of transaction raising message

- ✓ e_transactionid type /epo1/transactionid

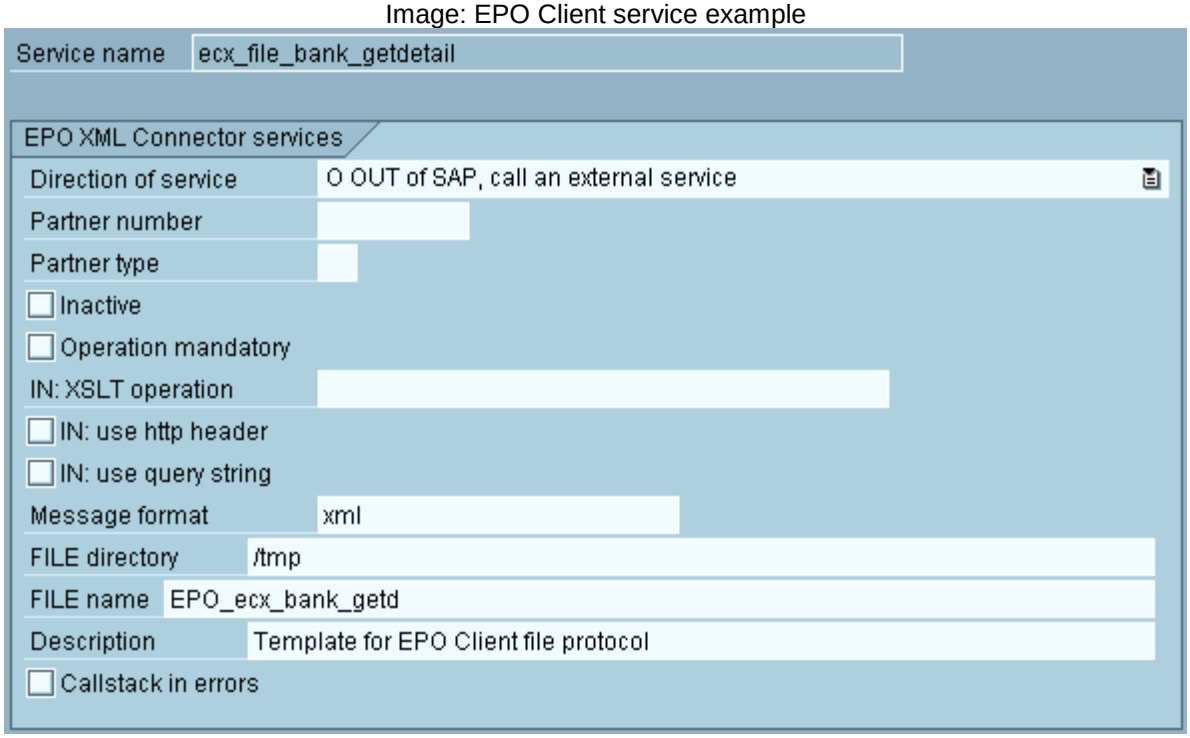
  Unique identifier of transaction being processed.

3.5.5 CREATING EPO CLIENT - INTEGRATION GUIDE

3.5.5.1 CREATE EPO CLIENT SERVICE

Description of service creation is in section 3.1. Image below shows an example of such service which can use FILE protocol as well.

Image: EPO Client service example



3.5.5.2 CONFIGURE EPO CLIENT SERVICE

Area menu: *EPO XML Connector Configuration → Outbound service configuration (external services) → EPO Client → Out: Maintain EPO Client service configuration*
Transaction: */EPO1/EPORTOUT12*

Image: EPO Client service configuration example for http

Service name	ecx_get_weather
Operation	get_weather
Version	

Configuration for EPO Client outbound services

NR object	/EPO1/NOR
Subobject value	
Number Range Number	00
☐ Inactive	
Protocol	0 HTTP
Processing type	S Synchronous
Store XML	6 store request and response information including XML message
☑ Compress	
☐ Special char to enti	
Source codepage	
XSLT out	
XSLT in	
Out.Req.Mapping FM	/EPO1/ORM_ACTUAL_WEATHER
Out.Req.Structure FM	/EPO1/ORS_ACTUAL_WEATHER
Out.Res.Structure FM	/EPO1/OSS_ACTUAL_WEATHER
Out.Res.Mapping FM	/EPO1/OSM_ACTUAL_WEATHER
HTTP host	www.webservicex.net
HTTP port	80
HTTP URI	http://www.webservicex.net/globalweather.asmx
HTTP content type	text/xml; charset=utf-8
HTTP proxy host	
HTTP proxy port	
HTTP SSL ID	
HTTP scheme	1 HTTP
HTTP timeout	
☐ FILE include MANDT	
☐ FILE include TransID	
☐ FILE include DATE	
☐ FILE include TIME	
FILE custom exit FM	
UM sender (SAP user)	
UM recipients	
UM CC recipients	
UM subject	
UM body text name	
UM body text ID	
UM content type	
UM attachement type	
UM customer exit FM	
Monitoring profile	EPOTEST
Description	

Fields "Service name", "Operation", "Version", "NR object", "Subobject value", "Number Range Number", "Inactive", "Processing type", "Store XML", "Compress", "Monitoring profile" and "Description" are common for every EPO service configuration. For description of these fields please see section 3.2.3.3.

Protocol:	HTTP or FILE. When set to HTTP, EPO Client sends request using HTTP(s) protocol to a Web Service, otherwise it stores it in a file. The directory and name of the file are set in service's fields.
Special char to enti:	Special character are converted to HTML (XML) entity character codes (e.g. „€" to „€" etc.) in messages.
Source codepage:	Code page being used for character conversion, when *Special char to enti* is set.
In customer exit fm:	Function module which is to be called in pre-processing stage of the request message (inbound), right after the message is saved, before the "XSLT in" transformation takes place. As a template you can use /epo1/exit_requestxml function module provided in delivery.
XSLT out:	XSLT transformation of request XML message.
XSLT in:	XSLT transformation of request XML message. It takes part in request handling right after the "In customer exit fm" call and before the message is processed (by calling Web service).
In.Req.Structure FM *In.Req.Mapping FM* *In.Res.Mapping FM* *In.Res.Structure FM*	Chapter 1.4.3 of this document describes these fields in detail.
HTTP host:	Host name (domain name) of server hosting the service.
HTTP port:	Port number on which the host server accepts messages. The default value is 80 for HTTP services.
HTTP URI:	The URI path to the service itself. The message is sent to the service using this URI when (re)processed.
HTTP Content type:	The HTTP header named 'Content-type' is given this value when this configuration is used. You will need to set it to 'text/xml' for calling all the Web services which receive XML message in HTTP body. Most of the Web services, particularly SOAP ones will not work if this header is not there or is set differently.
HTTP proxy host:	Name of proxy server to use for communication.

HTTP proxy port:	Port number of proxy server.
HTTP SSL ID:	By specifying the SSL client identity, you define the client certificate with which the SAP system logs on to the HTTP server.
HTTP Scheme:	HTTP or HTTPS for non-secure or secure HTTP communication.
HTTP timeout:	Parameter for (re)processing of messages. It's the amount of time in seconds for which the (re)processing program tries to send the request XML message and get the response.
FILE include MANDT:	FILE protocol only. Processing program will append SAP Client number (SY-MANDT) to file name.
FILE include TransID:	FILE protocol only. Same as above but TransactionID is appended.
FILE include DATE:	FILE protocol only. Date (SY-DATUM) appended to file name.
FILE include TIME:	FILE protocol only. System time (SY-UZEIT) appended to file name.
FILE custom exit FM:	Name of customer (Z) function module which is called after the file is written and closed. You can use this FM to rename / move the file when stored successfully.
UM sender (SAP User):	UM protocols only. SAP user name used as sender for UM messages.
UM recipients:	UM protocols only. Semicolon separated list of recipients.
UM CC recipients:	UM protocols only. Semicolon separated list of carbon copy recipients.
UM subject:	UM protocols only. Subject text for message.
UM body text name:	UM protocols only. Text for body of UM message. This is name of text object created using transaction SO10 (SAP standard texts).
UM body text ID:	UM protocols only. Text ID of a SAP standard text object (SO10).
UM content type:	UM protocols only. TXT (text/plain) or HTM (text/html) content type for UM message body.

UM attachment type: UM protocols only. XML data attachment can be 'inline' – within body of message or 'as attachment' – file attached to the message.

UM customer exit FM: UM protocols only. Name of customer exit function module for implementing other or own message sending system.

3.5.5.3 SET ADDITIONAL HTTP HEADERS IF NEEDED

Area menu: *EPO XML Connector Configuration → Outbound service configuration (external services) → EPO Client → Out: Maintain EPO Client HTTP headers*
Transaction: */EPO1/EPORTOUTH12*

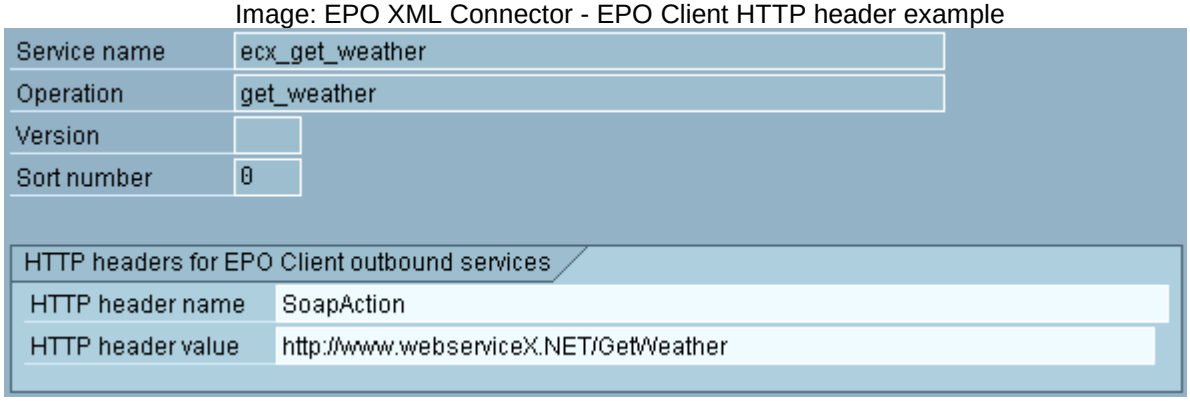
You can add any number of HTTP headers to the request message. When the Web service is called – the request message is to be sent – the headers are used in HTTP communication, therefore they can be extracted and used on the other side.

All the fields of this dialog should be self explanatory but the 'Sort number'. It is the last part of the key of table holding all the headers and so you need to use it if you are adding more than one header to the service.

HTTP header 'SOAPaction'

This header is used only when calling SOAP Web service. This header must contain the name of operation as it is listed in WSDL description of the Web service.

Image: EPO XML Connector - EPO Client HTTP header example



3.5.5.4 CREATE PROGRAM TO CALL /EPO1/EPOCLIENT FUNCTION MODULE

Finally you have to call the function module for sending the request XML message. So you need to create an ABAP program. The service in this example is calling the same Web service as the example in the next chapter (generated proxy client), so that you can see the differences between these two implementations.

Note: We have created XML request string in outbound request structure function module by calling XSLT transformation /epo1/tecx_out_actual_weather, which you can find (including all the FMs used here) in /EPO1/EXC_REPOSITORY package installed with EXC.

Example program for calling EPO Client function module

```
*&---------------------------------------------------------------------*
*& Report  /EPO1/ECX_ACTUAL_WEATHER                                    *
*&---------------------------------------------------------------------*
*& Company:   EPO Consulting                                           *
*& IP Rights: Intellectual Property Rights and all other rights are    *
*&            held by EPO Consulting.                                  *
*&            Copying or Modifying this program is only allowed with   *
*&            written consent of EPO Consulting.                       *
*& Author:    MH                                                       *
*& Date:      September 2008                                           *
*& Desc.:     Call external Web service using EPO Client               *
*&---------------------------------------------------------------------*
*& >> In order to use this example you need to create EPO Client
*&    service using following settings (selection screen default):
*&
*& Service name:          ecx_get_weather
*& Direction of service: OUT of SAP, call an external service
*& IN: XSLT operation:
*& IN: use http header:  Checked
*& Message format:       xml
*& Description:          EPO Client service example
*&
*& >> The service configuration is:
*&
*& Service name:    ecx_get_weather
*& Operation:       get_weather
*& Version:
*&
*& Processing type:      Synchronous
*& Store XML:            store request and response information
*&                       including xml message
*& Number Range Number:  00
*& Compress:             Checked
*& Special char to enti: Not checked
*& XSLT out:
*& XSLT in:
*& Hostname:             www.webservicex.net
*& Port:
*& Path (URI):           http://www.webservicex.net/globalweather.asmx
*& Content type:
*& Proxy hostname:
*& Proxy port:
*& SSL ID:
*& HTTP scheme:          HTTP
*& Timeout:
*& Description:          Get actual weather using EPO Client - example
*&
*& >> Also you need to add special HTTP header for SOAP
*&
*& Service name:    ecx_get_weather
*& Operation:       get_weather
*& Version:
*& Sort number:     0
*&
*& HTTP header name:  SoapAction
*& HTTP header value: http://www.webserviceX.NET/GetWeather
*&---------------------------------------------------------------------

REPORT  /epo1/ecx_actual_weather.

DATA: l_requestxml        TYPE xstring,
      l_responsexml       TYPE xstring,
```

```
      ls_requestxml       TYPE string,
      ls_responsexml      TYPE string,
      ls_city             TYPE string,
      ls_country          TYPE string,
      l_callstatus        TYPE /epo1/callstatus,
      l_convo             TYPE REF TO cl_abap_conv_out_ce,
      l_convi             TYPE REF TO cl_abap_conv_in_ce,
      ls_location         TYPE string,
      ls_time             TYPE string,
      ls_wind             TYPE string,
      ls_visibility       TYPE string,
      ls_skyconditions    TYPE string,
      ls_temperature      TYPE string,
      ls_dewpoint         TYPE string,
      ls_relativehumudity TYPE string,
      ls_pressure         TYPE string,
      ls_status           TYPE string,
      l_weather_response  TYPE /epo1/ecx_weather_response.

SELECTION-SCREEN BEGIN OF BLOCK sel WITH FRAME TITLE text-001.
PARAMETERS: p_ctry(50) TYPE c DEFAULT 'SLOVAKIA',
            p_city(50) TYPE c DEFAULT 'BRATISLAVA'.
SELECTION-SCREEN SKIP 1.
PARAMETERS: p_serv TYPE /epo1/service   DEFAULT 'ecx_get_weather',
            p_oper TYPE /epo1/operation DEFAULT 'get_weather',
            p_vers TYPE /epo1/version.
SELECTION-SCREEN END OF BLOCK sel.

* 1. Call outbound request mapping function module
CALL FUNCTION '/EPO1/ORM_ACTUAL_WEATHER'
  EXPORTING
    i_cityname    = p_city
    i_countryname = p_ctry
  IMPORTING
    e_cityname    = ls_city
    e_countryname = ls_country
  CHANGING
    c_callstatus  = l_callstatus.

* 2. Call outbound request structure function module
CALL FUNCTION '/EPO1/ORS_ACTUAL_WEATHER'
  EXPORTING
    i_cityname    = ls_city
    i_countryname = ls_country
  IMPORTING
    e_xmlrequest  = l_requestxml
  CHANGING
    c_callstatus  = l_callstatus.

* 3. EPO Client processing - send request, receive response
IF l_callstatus-code < 500.
  CALL FUNCTION '/EPO1/EPOCLIENT'
    EXPORTING
      i_requestxml  = l_requestxml
      i_servicename = p_serv
      i_operation   = p_oper
      i_version     = p_vers
    IMPORTING
      e_responsexml = l_responsexml
      e_callstatus  = l_callstatus.
ENDIF.

* 4. Call outbound response structure function module
```

```
CALL FUNCTION '/EPO1/OSS_ACTUAL_WEATHER'
  EXPORTING
    i_responsexml      = l_responsexml
  IMPORTING
    e_location         = ls_location
    e_time             = ls_time
    e_wind             = ls_wind
    e_visibility       = ls_visibility
    e_skyconditions    = ls_skyconditions
    e_temperature      = ls_temperature
    e_dewpoint         = ls_dewpoint
    e_relativehumidity = ls_relativehumudity
    e_pressure         = ls_pressure
    e_status           = ls_status
  CHANGING
    c_callstatus       = l_callstatus.

* 5. Call outbound response mapping function module
CALL FUNCTION '/EPO1/OSM_ACTUAL_WEATHER'
  EXPORTING
    i_location         = ls_location
    i_time             = ls_time
    i_wind             = ls_wind
    i_visibility       = ls_visibility
    i_skyconditions    = ls_skyconditions
    i_temperature      = ls_temperature
    i_dewpoint         = ls_dewpoint
    i_relativehumidity = ls_relativehumudity
    i_pressure         = ls_pressure
    i_status           = ls_status
  IMPORTING
    e_weather_response = l_weather_response
  CHANGING
    c_callstatus       = l_callstatus.

IF l_callstatus-code < 500.

* In this exapmle we just view the response on screen
  WRITE:/'Location.....:', l_weather_response-location,/
         'Time.....:', l_weather_response-time,/
         'Wind.....:', l_weather_response-wind,/
         'Visibility.....:', l_weather_response-visibility,/
         'Skyconditions.....:', l_weather_response-skyconditions,/
         'Temperature.....:', l_weather_response-temperature,/
         'Dew Point.....:', l_weather_response-dewpoint,/
         'Relative Humidity.:', l_weather_response-relativehumidity,/
         'Pressure.....:', l_weather_response-pressure,/
         'Status.....:', l_weather_response-status.

ELSEIF l_responsexml IS NOT INITIAL.

  CALL FUNCTION 'DISPLAY_XML_STRING'
    EXPORTING
      xml_string      = l_responsexml
      title           = 'Server response directly displayed'
      starting_x      = 5
      starting_y      = 5
    EXCEPTIONS
      no_xml_document = 1
      OTHERS          = 2.
  IF sy-subrc <> 0.
    MESSAGE l_callstatus-description
      TYPE  'S'
```

```
      DISPLAY LIKE l_callstatus-type.
  ENDIF.

ELSE.
  MESSAGE l_callstatus-description
    TYPE  'S'
    DISPLAY LIKE l_callstatus-type.
  EXIT.
ENDIF.
```

3.5.5.5 TESTING AN EPO CLIENT SERVICE

Testing of outgoing integration (calling 3rd party Web service) is nothing more than running and debugging the program you wrote.

To display the request and/or the response for testing purposes, you can use either SAP function module DISPLAY_XML_STRING in your program (like we did), which displays XML message in SAP or you can configure the service to store request and/or response and then use /EPO1/MESSAGESLIST program to display those messages (Area menu *EPO XML Connector Data Maintenance* ▱ */EPO1/MESSAGESLIST – List and view stored messages*). The /EPO1/MESSAGESLIST program uses the DISPLAY_XML_STRING function module, so you don't need to call it in your report.

3.6 OUTBOUND: SAP CLIENT

The SAP Client function provides an extension and simplification to the SAP standard for generated ABAP client proxies. ABAP proxies allow you to consume external web services.
You can generate an ABAP proxy using a WSDL in SAP standard and use this ABAP proxy in the EPO XML Connector. All functions of the EPO XML Connector are available then (store, reprocess, XSLT transform,). Two customer exits allow you to access request and response XML data and add your custom functions.

SAP client is SAP standard – generated ABAP client proxy, by which you can call "outside" Web services. You can use this standard together with EPO XML Connector functionality via the method of class /epo1/cl_proxyconnector - /epo1/sapclient and reprocessing program /epo1/proxy_reprocess.

The SAP Client is one option creating an outbound integration with SAP. The other option for outbound integration is using the unique EPO Client of the EPO XML Connector. You can decide service by service, which option is appropriate. We recommend using EPO Client as it works for any web service and is more flexible and easy to implement.

Usage of outbound SAP Client (compare with outbound EPO Client)

- Integrations, where you can generate an ABAP proxy from a provided WSDL

Overview: Steps for creating outbound integrations using SAP Client:

- Generate the ABAP proxy using a WSDL
- Configure the service in the EPO XML Connector
- Write program for calling the SAP Client (the generic method of the EPO XML Connector)
- Test your integration

Note: SAP Client is not available on WAS 6.20.

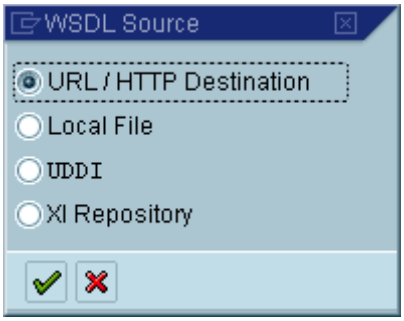
3.6.1 CREATING SAP CLIENT - INTEGRATION GUIDE

3.6.1.1 GENERATE ABAP CLIENT PROXY

To do this you need to start Object Navigator (transaction SE80), Repository Browser View, select "Package" in drop-down combo box and open or create new package, where the proxy objects will be added into. Next right-click on package name and select *Create → Enterprise Service / Web Service → Proxy Object* in the menu which appears. The proxy creation wizard starts afterwards. The example below does not describe all possibilities of proxy creation process. You can find full documentation for SAP 700 on
http://help.sap.com/saphelp_nw70/helpdata/en/ba/d4c23b95c8466ce10000000a114084/content.htm.

1. *WSDL Source*

   In the first window you can choose the location of service's WSDL (Web Service Description Language) file / generator.

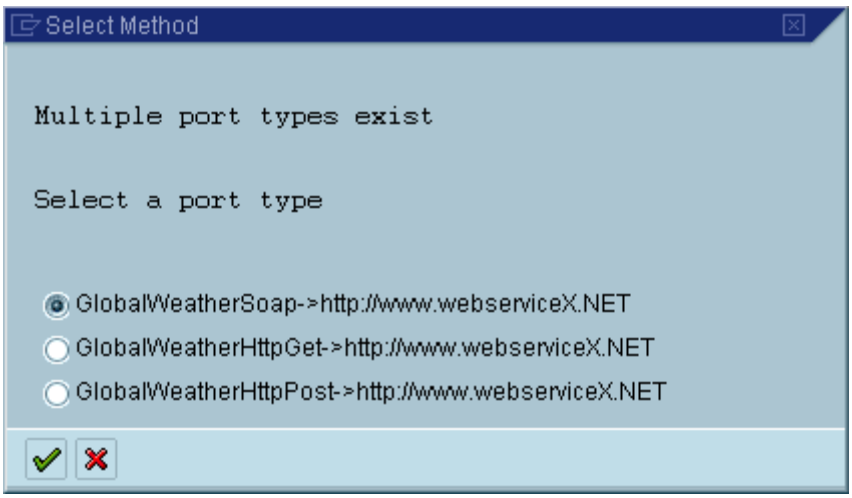


2. *Choose URL or HTTP destination*

   The URL we used is the same as in EPO Client service example – http://www.webservicex.net/globalweather.asmx?wsdl.

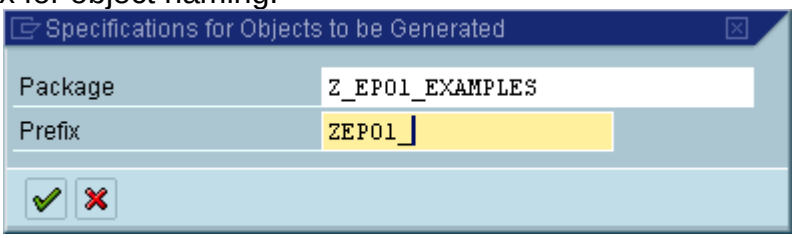
3. *Select Method*

   This step applies only for the Web services having multiple access methods.



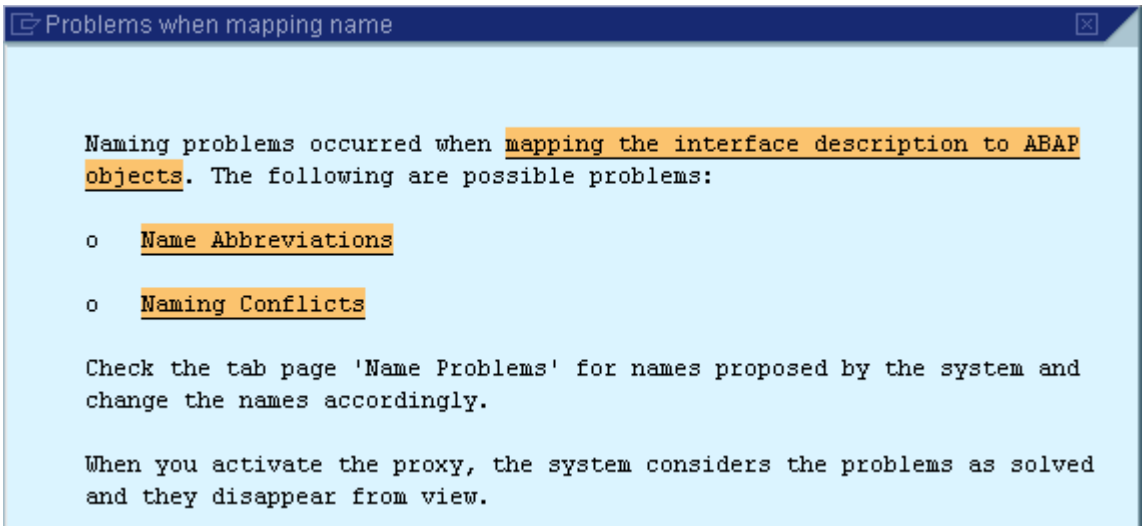
4. *Specifications for Objects to be Generated*

   Here you input the package you want the client proxy object to be generated into and the prefix for object naming.



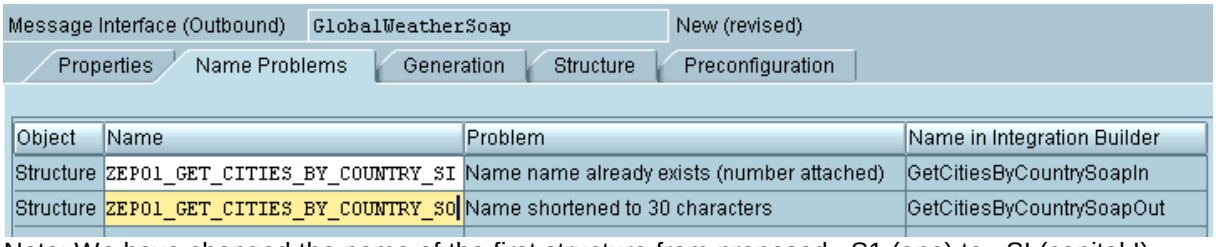
5. *Problems with mapping name*

   You can see on the image below that some of the generated names in the object were too long or in conflict with existing ones. In the next step we correct this problem.



6. *Correct the naming problems*

   When all the object components are generated you click on the “Name Problems” tab to see list of names to be changed. The wizard did the corrections automatically, but still you might need to change the names to fulfil your needs.



Object	Name	Problem	Name in Integration Builder
Structure	ZEP01_GET_CITIES_BY_COUNTRY_SI	Name name already exists (number attached)	GetCitiesByCountrySoapIn
Structure	ZEP01_GET_CITIES_BY_COUNTRY_SO	Name shortened to 30 characters	GetCitiesByCountrySoapOut

Note: We have changed the name of the first structure from proposed _S1 (one) to _SI (capital I). The example program which calls this proxy object uses this name in the source code so you need to change it the same way for the example to work.

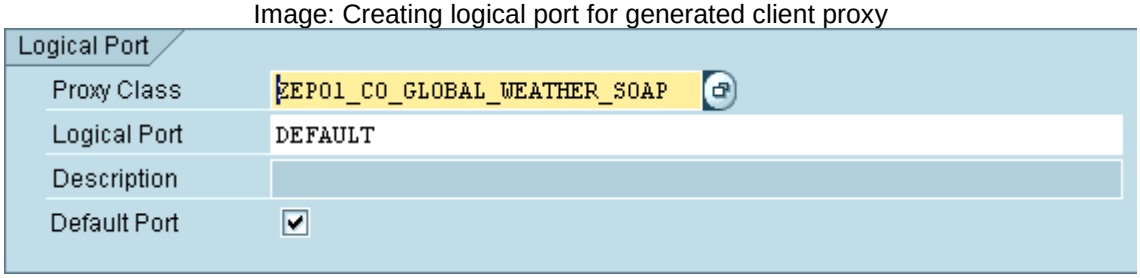
7. *Activate client proxy*

   Obviously the client proxy you created will not work if you don’t activate it.

3.6.1.2 Create Logical Port for Generated Client Proxy

Runtime features of the proxy you generated need to be assigned to it by so-called "Logical Port" using transaction LPCONFIG. Full documentation can be found on http://help.sap.com/saphelp_nw04/helpdata/en/16/285d32996b25428dc2eedf2b0eadd8/content.htm.

Image: Creating logical port for generated client proxy



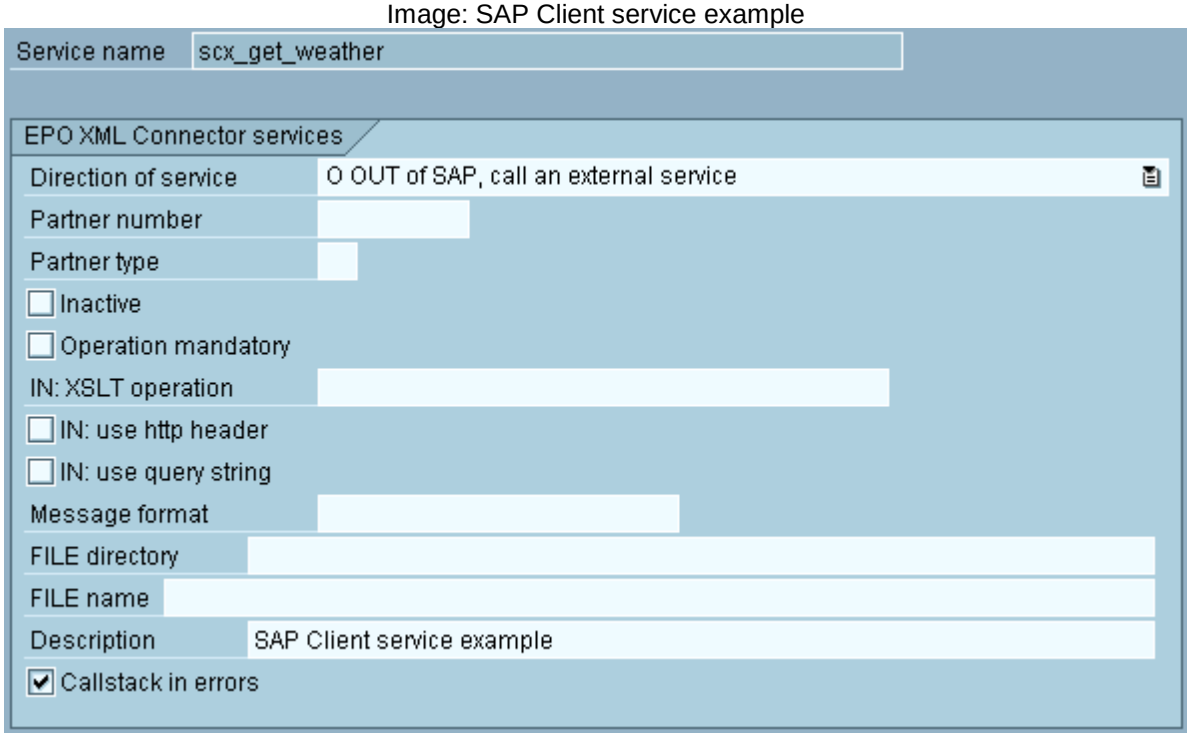
Note: We created default logical port changing nothing in the proposed attributes.

Please do not forget to activate the Logical Port you created.

3.6.1.3 Create SAP Client service

Description of service creation is in section 3.1. Image below shows an example of such service.

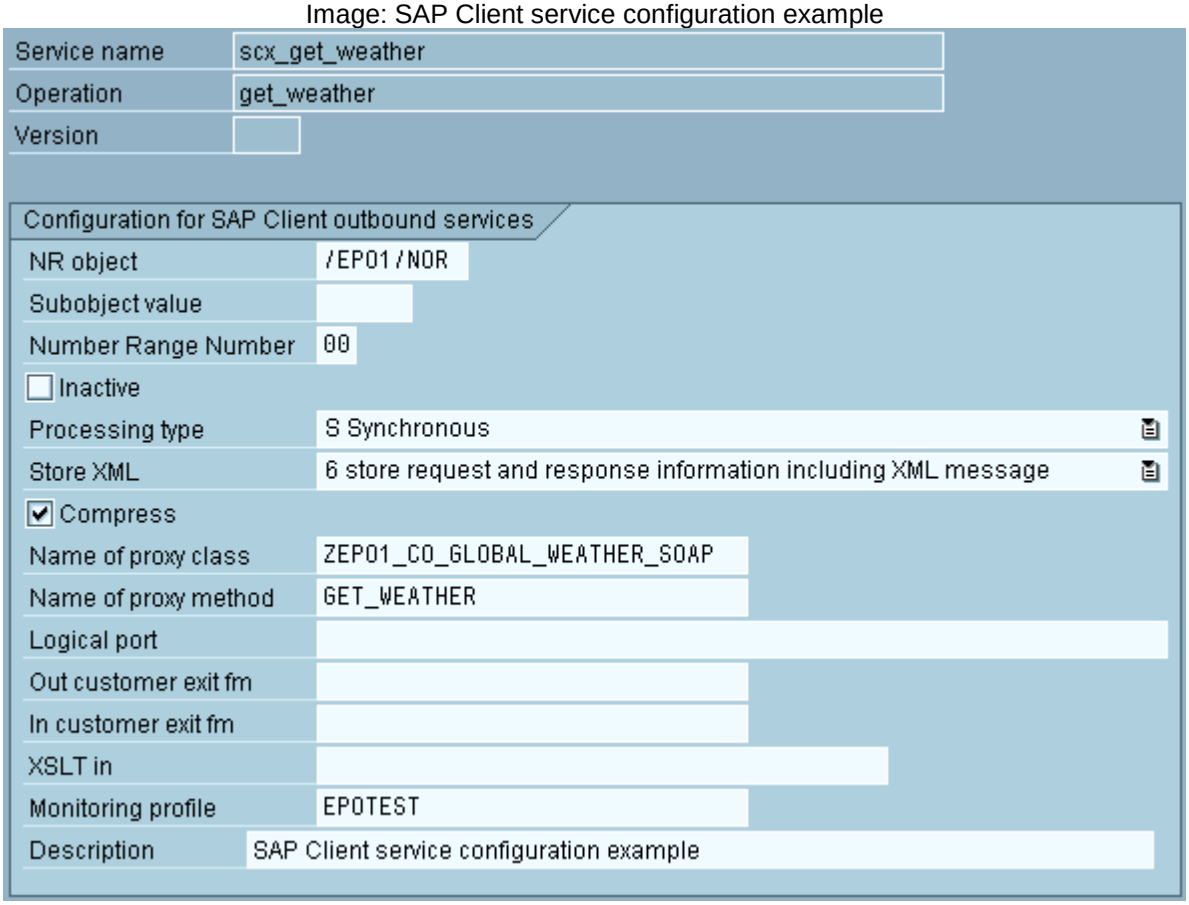
Image: SAP Client service example



3.6.1.4 CONFIGURE SAP CLIENT SERVICE

Area menu: *EPO XML Connector Configuration → Outbound service configuration (external services) → Generated ABAP Proxy Client → Out: Create/Change Proxy configuration*
Transaction: */EPO1/WSOUT12*

Image: SAP Client service configuration example



Fields "Service name", "Operation", "Version", "NR object", "Subobject value", "Number Range Number", "Inactive", "Processing type", "Store XML", "Compress", "Monitoring profile" and "Description" are common for every EPO service configuration. For description of these fields please see section 3.2.3.3.

Name of proxy class: Class name created by generation of client proxy. This class name is used for reprocessing of messages using /epo1/reprocessproxy program.

Name of proxy method: Method of the proxy class. The method in this case is the operation of the Web service to be called.

Logical port: Logical port name to use when the client proxy is called. You can leave it empty if you want to use the default logical port.

Out customer exit FM: Function module which is called in pre-processing stage of the request message (outbound), right after the message is saved, before the "XSLT in" transformation takes place.

In customer exit FM: Function module which is called in post-processing stage of the response message (inbound), right after the message is saved.

XSLT in: XSLT transformation of response XML message. It takes part in request handling right after the “In customer exit FM”.

3.6.1.5 CREATE PROGRAM TO CALL SAP CLIENT

Call the EPO XML Connector method in your program. The method name is “/epo1/sapclient” and it belongs to class “/epo1/cl_proxyconnector”. The service in the example below is calling the same Web service as the example of EPO Client so you can see the differences between these two implementations.

Example program for calling EPO client method

```
*&---------------------------------------------------------------------*
*& Report  /epo1/scx_actual_weather                                    *
*&---------------------------------------------------------------------*
*& To uncomment this example please use /EPO1/COMMENTUNCOMMENT program*
*&---------------------------------------------------------------------*
*& To use this example you will need to generate Client Proxy object. *
*& WSDL from URL: http://www.webservicex.net/globalweather.asmx?wsdl   *
*& Prefix ZEPO1_ ( Class name zepo1_co_global_weather_soap ).          *
*& Please change naming abbreviations to                               *
*& zepo1_get_cities_by_country_si for input structure and              *
*& zepo1_get_cities_by_country_so for output structure of method       *
*& get_cities_by_country which is used in another example.             *
*&---------------------------------------------------------------------*
*& Company:   EPO Consulting                                           *
*& IP Rights: Intellectual Property Rights and all other rights are    *
*&            held by EPO Consulting.                                  *
*&            Copying or Modifying this program is only allowed with   *
*&            written consent of EPO Consulting.                       *
*& Author:    MH                                                       *
*& Date:      May 2007                                                 *
*& Desc.:     comment/uncomment reports                                *
*&---------------------------------------------------------------------*

report  /epo1/scx_actual_weather.

data: l_ro            type ref to cx_root,
      ls_error        type string,
      l_requestxml    type xstring,
      l_responsexml   type xstring,
      l_transactionid type /epo1/transactionid,
      l_configouts    type /epo1/configouts,
      l_return        type bapiret2.

selection-screen begin of block sel with frame title text-001.
parameters: p_ctry(50) type c default 'SLOVAKIA',
            p_city(50) type c default 'BRATISLAVA'.
selection-screen end of block sel.

data: output type zepo1_get_weather_soap_out .
data: input type zepo1_get_weather_soap_in .
```

```
input-city_name = p_city.
input-country_name = p_ctry.

* EPO Connector processing
call method /epo1/cl_sapclient=>/epo1/sapclient
  exporting
    i_service         = 'GET_WEATHER_PROXY'
    i_operation       = 'GET_WEATHER'
*    i_version         =
    i_proxy_input     = input
*    i_reprocess       =
  importing
    e_proxy_output    = output
    e_return          = l_return
  changing
    c_transactionid   = l_transactionid
*    c_request_asxml   =
    c_response_asxml  = l_responsexml
*    c_request_message =
    c_fkey1           = p_city
    c_fkey2           = p_ctry
*    c_fkey3           =
*    c_fkey4           =
    .

* e_return (BAPIRET2) error handling
if l_return-type = 'E' or l_return-type = 'A'.
  call transformation id
    source error = l_return
    result xml l_responsexml.
else.

endif.

* view response
CALL FUNCTION 'DISPLAY_XML_STRING'
  EXPORTING
    xml_string      = l_responsexml
    title           = text-002
    starting_x      = 5
    starting_y      = 5
  EXCEPTIONS
    no_xml_document = 1
    OTHERS          = 2.
```

3.6.1.5.1 /EPO1/SAPCLIENT METHOD INTERFACE

Import

- *i_service* type /epo1/service
- *i_operation* type /epo1/operation optional
- *i_version* type /epo1/version optional

  EPO XML Connector service name, operation and version parameters are used to find the generated client proxy method to call, which is set in service's configuration.

- *i_proxy_input* type any optional

  Request data to be send to Web service. The input data must have the structure of the input parameter of the generated proxy method.

- *i_reprocess* type flag optional

  This parameter is here for reprocessing program only. Please use /epo1/proxy_reprocess program for reprocessing stored messages.

Export

- *e_proxy_output* type any

  Generated ABAP client proxy output parameter. This is the response in the format defined by generated method. If you want to access the response in the XML format you need to use another parameter – c_response_asxml

- *e_return* type bapiret2

  Return parameter for error handling.

Changing

- *c_transactionid* type /epo1/transactionid

  Although you might not need to use this variable, you have to provide it for the method call for it is used inside for handling. It will carry the unique transaction identifier for the transaction created by the call of this method.

- *c_request_asxml* type xstring optional

  Serialized request message data (by transformation id) must fit the generated proxy input data structure. This is the second way of how to pass the data to this method. Note that if you use both i_proxy_input and i_request_asxml, the method will raise an error.

- *c_request_status* type /epo1/status optional

Giving variable to this parameter, you will be able to see the status code of request message after it was processed.

- *c_request_message* type /epo1/message optional

  You can find useful information about processing of request when you use this parameter.

- *c_fkey1 – c_fkey4* type /epo1/fkey1 - /epo1/fkey4 optional

  These four foreign keys are stored with each message. You can set or read any of them in here or in user exit function modules. You can use them to store any data you need to store with every message. The example of such usage would be using one of the keys for storing some information you don't provide to the 3rd party web service and then using the key for searching.

3.6.1.6 TESTING A SAP CLIENT SERVICE

Testing of SAP Client service is exactly the same as the testing of EPO Client service so please see the section 3.4.4.5.

4 GENERIC FUNCTION MODULE CALL (IMPLEMENTED IN EPO RUNTIME)

General function module call (GFMC) is special functionality of EPO XML Connector to allow you using specific EPO Runtime service for calling any SAP function module accessible within you SAP system. In other words, you can directly call any SAP function module from outside of SAP using xml messages. A standard, W3-conform WSDL is provided (same on SAP systems worldwide). An excerpt of more than 3000 WSDL for SAP BAPI's is published on www.epoconsulting.com.

First you must setup the service (one time only).

4.1 SET UP: SERVICE FOR GENERIC FUNCTION MODULE CALL

☞ **Precondition 1: EPO XML Connector installation**
Installation is standard with transaction SAINT (installation) and SPAM (support packages). There is no difference to SAP standard software. Please refer to the installation documentation.

☞ **Precondition 2: SICF Service /epo1soa/xmlhandler must be active**
Area menu: EPO XML Connector Administration → HTTP Service Hierarchy Maintenance (ICF)
Transaction: SICF
For details see section 2.3.

☞ **Precondition 3: Number range must be set up**
If you want to store request or response messages, you need a number range for the TransactionID.
Area menu: EPO XML Connector Configuration → Maintain number range for EPO XML Connector messages
Transaction: /EPO1/NOR.
For details see section 2.2.

☞ **Precondition 4: Valid license key must be installed for production systems**
Area menu: EPO XML Connector Administration → Load license key for EPO XML Connector
Transaction: /EPO1/SETLICENSE
For details see section 2.1.

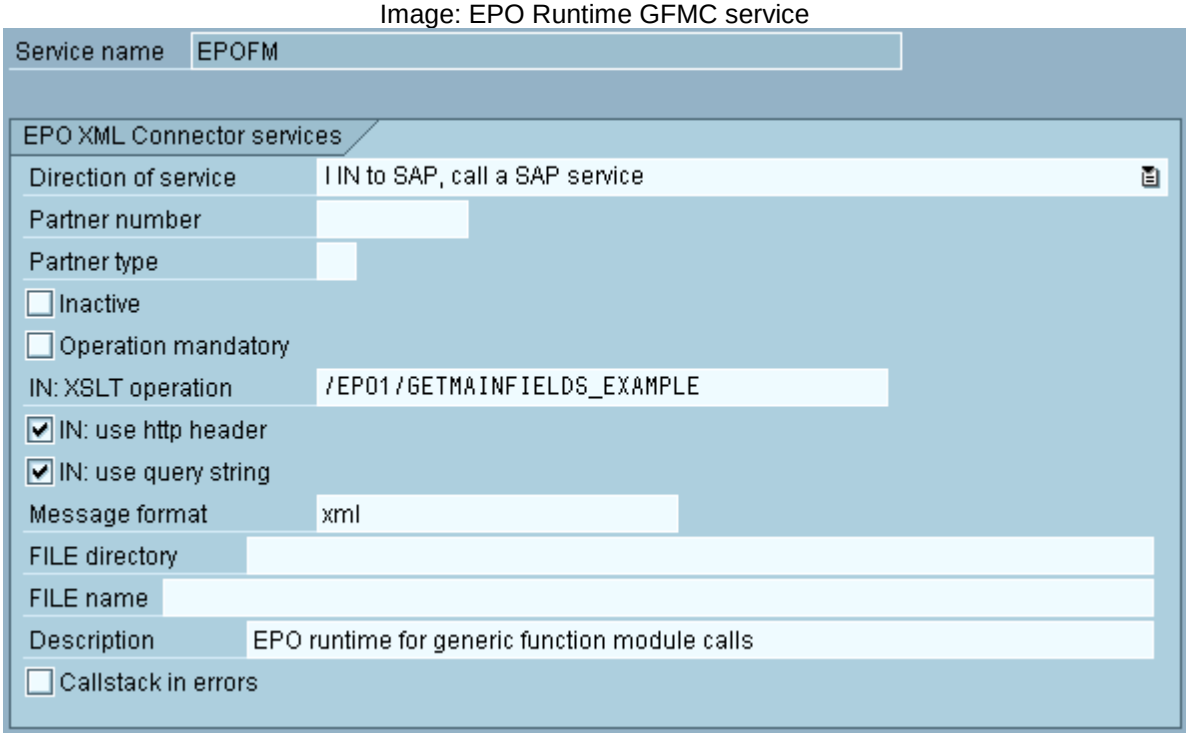
☞ **Step 1: Set up a service**

You need to create a specific EPO Runtime service (inbound) for the GFMC. The image below shows such a service. EXC services are documented in section 3.1.

Area menu: *EPO XML Connector Configuration → Maintain Services EPO XML Connector*
Transaction: */EPO1/SERVICES12*

Hint: We use EPOFM as service name. You could use any other name.

Image: EPO Runtime GFMC service



Operation mandatory: Notice that 'Operation mandatory' is not checked, which enables you to call any SAP function module without creating specific configurations for specific operations (operation = function module name). Only 1 configuration with empty operation is needed as shown in the example below. We strongly recommend checking the 'Operation mandatory' field for production systems, so that not any non-configured function module call would work.

IN: XSLT operation: The used XSLT /EPO1/GETMAINFIELDS_EXAMPLE tries to read the first xml element in the <soap:Body>. If it is found, it is used as “Operation” (operation = SAP function module name).

IN: use http header: If “Operation” was not found or is not checked with the “IN: XSLT operation”, the http header SOAPaction is used.

IN: use query string: If “Operation” was not found with the previous 2 settings, it will be read from the query string (&operation=[function module name]). This setting is necessary when using GET (not POST), for example for producing the WSDL (see below).

☞ **Step 2: Set up the configuration for the service**

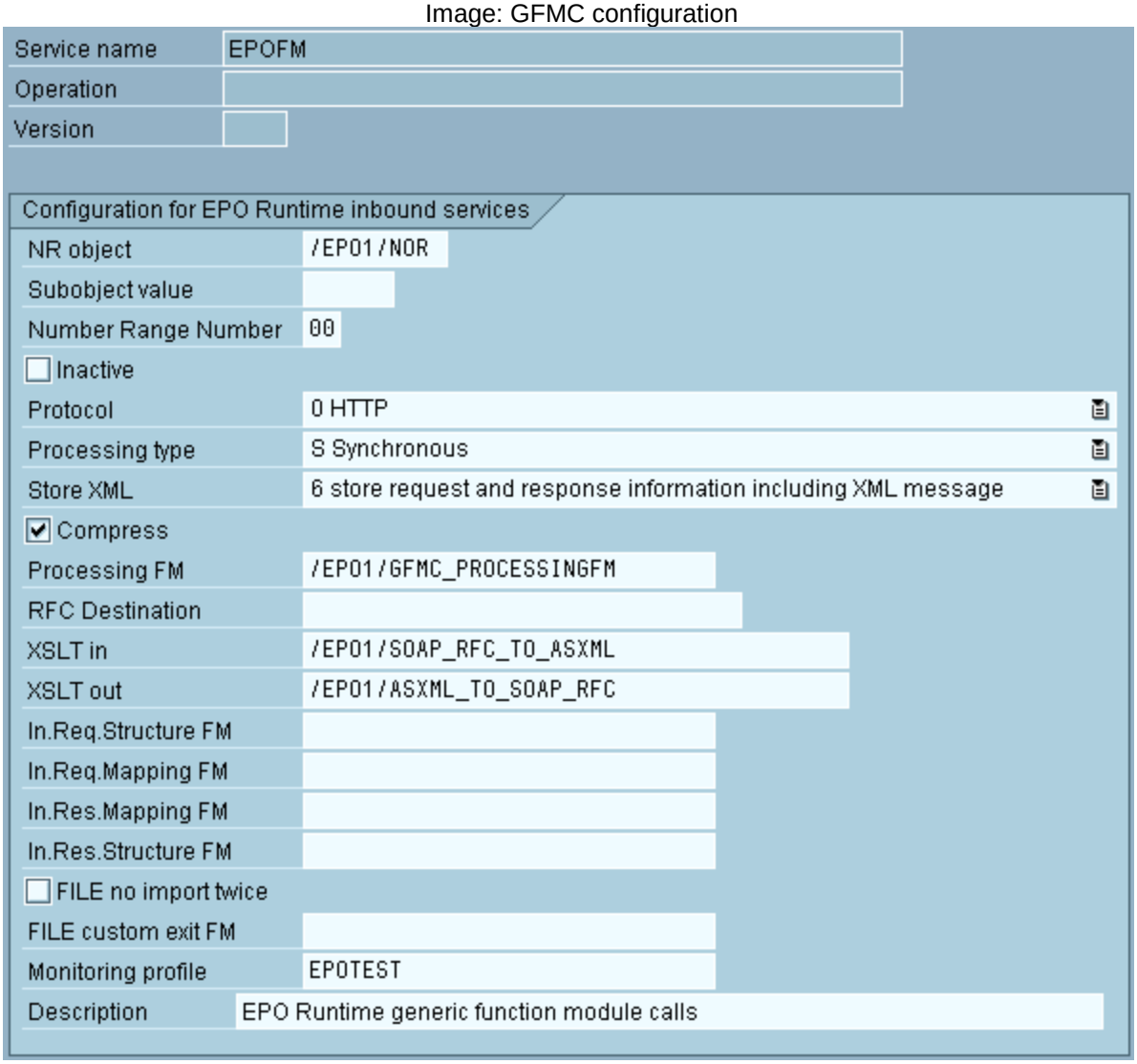
Area menu: *EPO XML Connector Configuration → Inbound service configuration (SAP Services) → EPO runtime → In: Maintain EPO runtime service configuration*
Transaction: */EPO1/EPORTIN12*

The GFMC service uses “operation” parameter passed to it as the name of the SAP function module, which is called then.

Here it is important that you use:

- **/EPO1/GFMC_PROCESSINGFM** as the processing function module. It encapsulates all functionality for calling generically any SAP function modules. It also holds the functionality inside for generating WSDL’s.
- **/EPO1/SOAP_RFC_TO_ASXML** as the XSLT in. It transforms incoming SOAP requests into ABAP XML (ASXML).
- **/EPO1/ASXML_TO_SOAP_RFC** as the XSLT out. It transforms ABAP XML (ASXML which is produced with CALL TRANSFORMATION ID) into SOAP.

Image: GFMC configuration



Notice: This is the configuration as we use it on our test & development systems. It has empty operation field, which enables any SAP function module to be called. In other words, it is used when there is no specific setting for a SAP function module.
The precondition for this entry to work is that “operation mandatory” is not set for the service (see above).
The opposite situation would be having entries with specific FM names in the operation field.

Notice: Like in any other EXC service you need a number range (and the NR object) for creation of TransactionIDs – unique transaction identifiers. The protocol is set to HTTP although you could use FILE protocol if you wish to use files for request messages (e.g. upload files in development). Synchronous processing type means that the call is made at the time when request arrives, opposite to asynchronous, where the request would be only stored and processing would be done later either manually or by a scheduled job. This configuration also stores the request and response information and compressed (compress check box) data into EXC tables (Store XML field). All other fields of this dialogue are described in chapter 3.2.3.3.

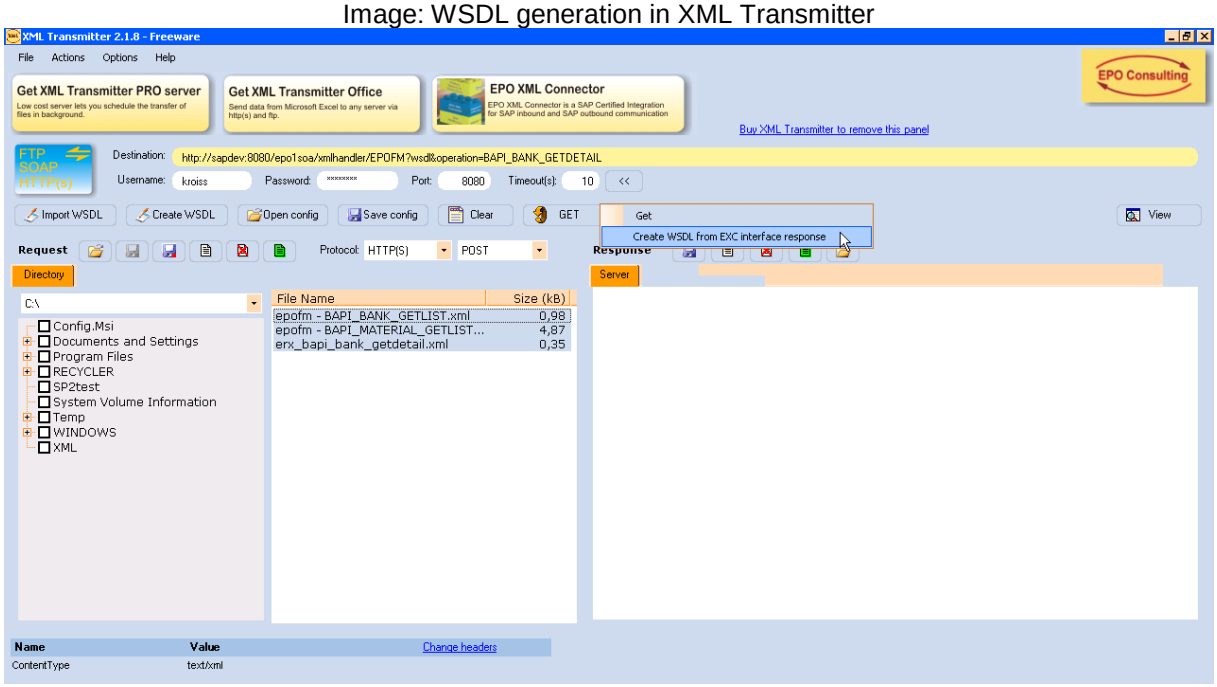
With this setup you are able to call hundreds of thousand of SAP function modules. Let's produce WSDL's and call the SAP function modules.

4.2 WSDL GENERATION IN XML TRANSMITTER FOR SAP FUNCTION MODULE

After you created and configured the GFMC service, you can create WSDL files in our freeware tool named XML Transmitter. You will fill in the URI and click GET, submenu "Create WSDL from EXC interface response". The function module for which you want to create the WSDL is specified as operation for GFMC service. Please note the "..?wsdl.." part of query string following the GFMC service name, which makes it create interface response for XML transmitter. If you want to use function module which includes namespace in its name like /EPO1/EXC_STORE_REQUEST you will need to replace the '/' character with '_-', so it will become _-EPO1_-STORE_REQUEST. The reason behind this is that '/' character cannot be used in xml element names.

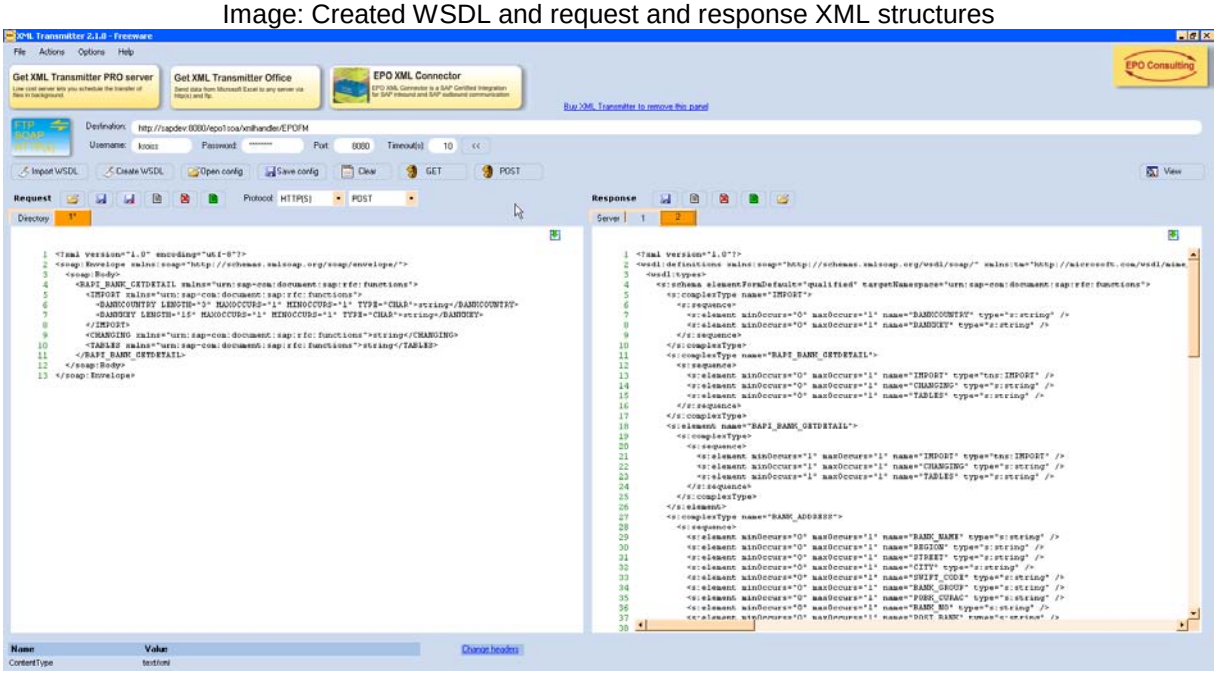
URI example for FM BAPI_BANK_GET_DETAIL:
http://saptest:8080/epo1soa/xmlhandler/**EPOFM**?wsdl&operation=BAPI_BANK_GETDETAIL

Image: WSDL generation in XML Transmitter



The XML Transmitter will then open three new tabs; on client side (left) you will see request xml structure, on server side (right) there will be response xml structure and the WSDL created.

Image: Created WSDL and request and response XML structures



After this step you can save the WSDL and use it for creation of proxy object to call the specified function module.

Notice: The WSDL will be the same on any SAP system as long as the SAP function module interface does not change (Release independent).

4.2.1 Structure of the EXC GFMC WSDLs:

The structure of any WSDL will have the SAME top level XML elements:

For Request it will have 1 or 2 or 3 of those xml elements:

<IMPORT> containing all IMPORT parameters of a SAP function module. XML element names will be same as function module parameters.

<CHANGING> containing all CHANGING parameters of a SAP function module. XML element names will be same as function module parameters.

<TABLES> containing all TABLES parameters of a SAP function module. XML element names will be same as function module parameters.

For Response it will have 1 or 2 or 3 of those xml elements:

<EXPORT> containing all IMPORT parameters of a SAP function module. XML element names will be same as function module parameters.

<CHANGING> containing all CHANGING parameters of a SAP function module. XML element names will be same as function module parameters.

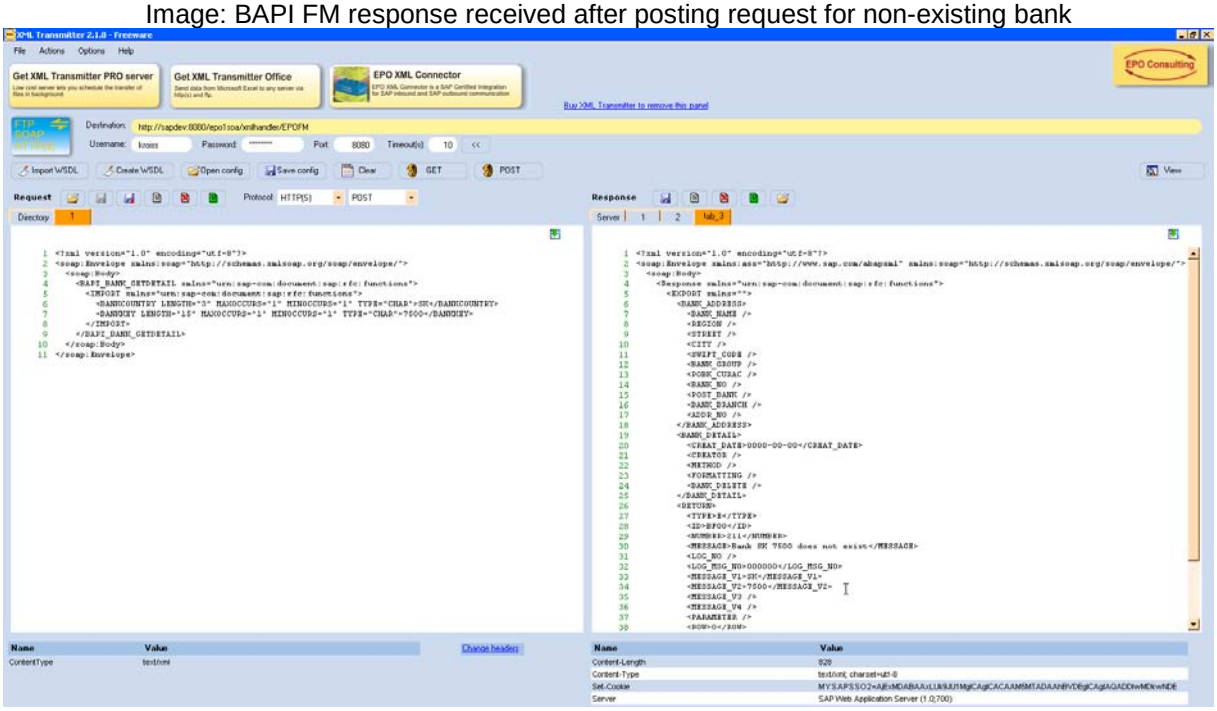
<TABLES> containing all TABLES parameters of a SAP function module. XML element names will be same as function module parameters.

Notice: In the WSDL all types will be set to string. However in the request and response XML, which is created together with the WSDL in the XML Transmitter, you can see all type definitions of all SAP fields (=xml elements with same name). At runtime you must make sure, that all sent data confirms to the SAP fields. For example dates must be sent as yyyymmdd (e.g. 20090430). If you want to send different formats or cannot use the EXC GFMC runtime, we recommend using EPO runtime instead. EPO runtime means, that you are using your own processing fm (opposite to using /EPO1/GFMC_PROCESSINGFM). Trick: You can copy the mapping program from the GMFC runtime and use it in your own processing fm.

4.3 Testing the WSDL (the SAP function) in XML Transmitter

You can POST the request xml message, which is generated together with the WSDL in XML Transmitter to test the SAP function right away from this point. Of course, you must fill it first with correct data. The image below shows response of BAPI_BANK_GETDETAIL function module to my request for bank which does not exist on our development system.

Image: BAPI FM response received after posting request for non-existing bank



SOAP Fault message response in error cases for EXC GFMC runtime:
In error cases the response you will receive will be standard SOAP fault message with the error described in <detail> element using /epo1/callstatus structure.

Example of SOAP fault GFMC response:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:asx="http://www.sap.com/abapxml"   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Fault>
        <faultcode>EPO1_EXC_ERROR</faultcode>
        <faultstring>EPO XML Connector Error</faultstring>
        <detail>
            <CALLSTATUS>
                <CODE>551</CODE>
                <TYPE>E</TYPE>
                <SUBJECT>XML transformation error</SUBJECT>
                <DESCRIPTION>Exception in XSLT processor: No valid source context supplied</DESCRIPTION>
                <TRANSACTIONID></TRANSACTIONID>
            </CALLSTATUS>
        </detail>
    </soap:Fault>
</soap:Envelope>
```

5 PUBLIC FUNCTION MODULES - REQUEST/RESPONSE API

The API function modules allow you to store message of any kind into EXC tables. Its intended use is to include existing interfaces of any kind (file up- and downloads, IDOC interfaces...) into the EPO XML Connector.

Existing interfaces are then automatically extended with

- monitoring features
- storing metadata information (who, when, what,...)
- storing the interface transaction data in binary form and
- reprocessing options

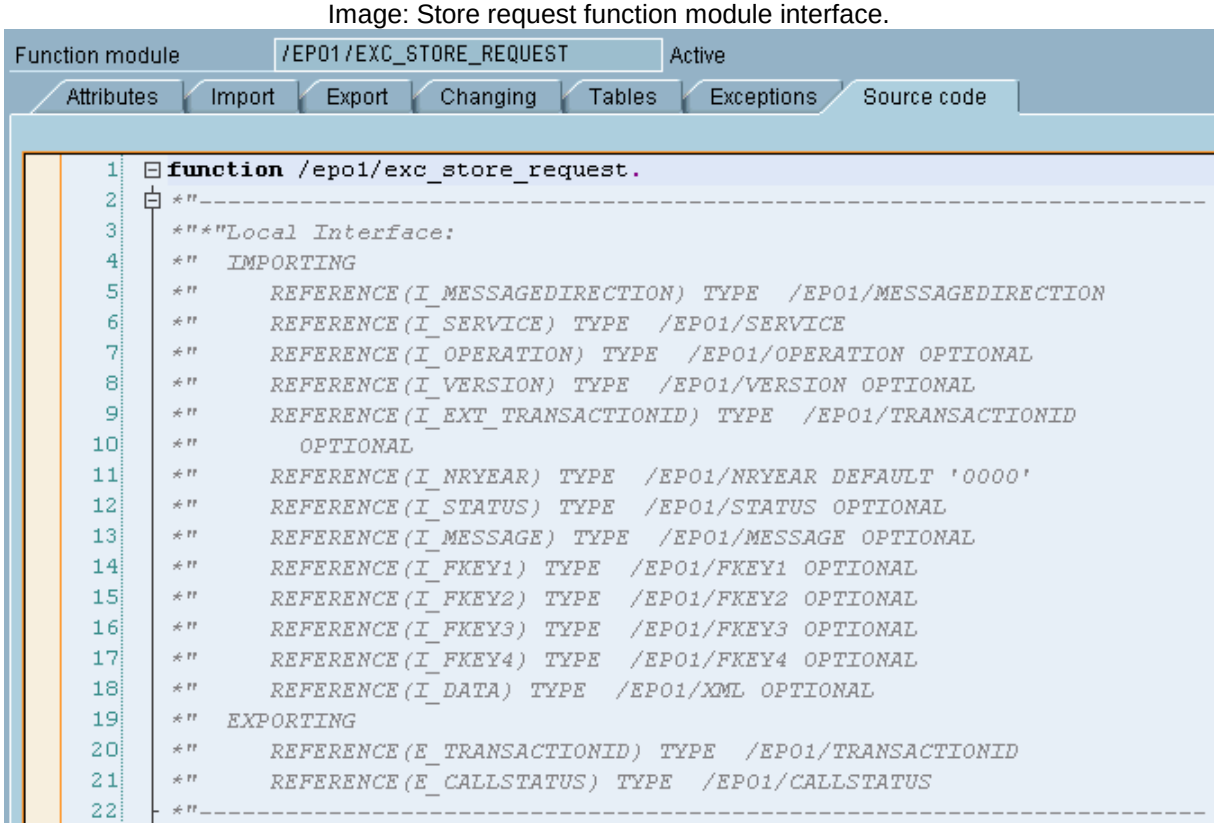
5.1 /EPO1/EXC_STORE_REQUEST INTERFACE

The function stores request (only) message into EXC tables. As you can see on the image, the interface is straightforward, all the parameters has already been described in this documentation. The function checks the existence of specified service and configuration and creates (if configured) specified header (/EPO1/XMLHEAD) and/or data (/EPO1/XMLDATA) records. In case of any error you can use E_CALLSTATUS to describe it.

Precondition: Service must be set up in EPO XML Connector

Notice: Monitoring messages are not sent at runtime. You must use the EPO XML Connector monitor to produce monitoring messages.

Image: Store request function module interface.

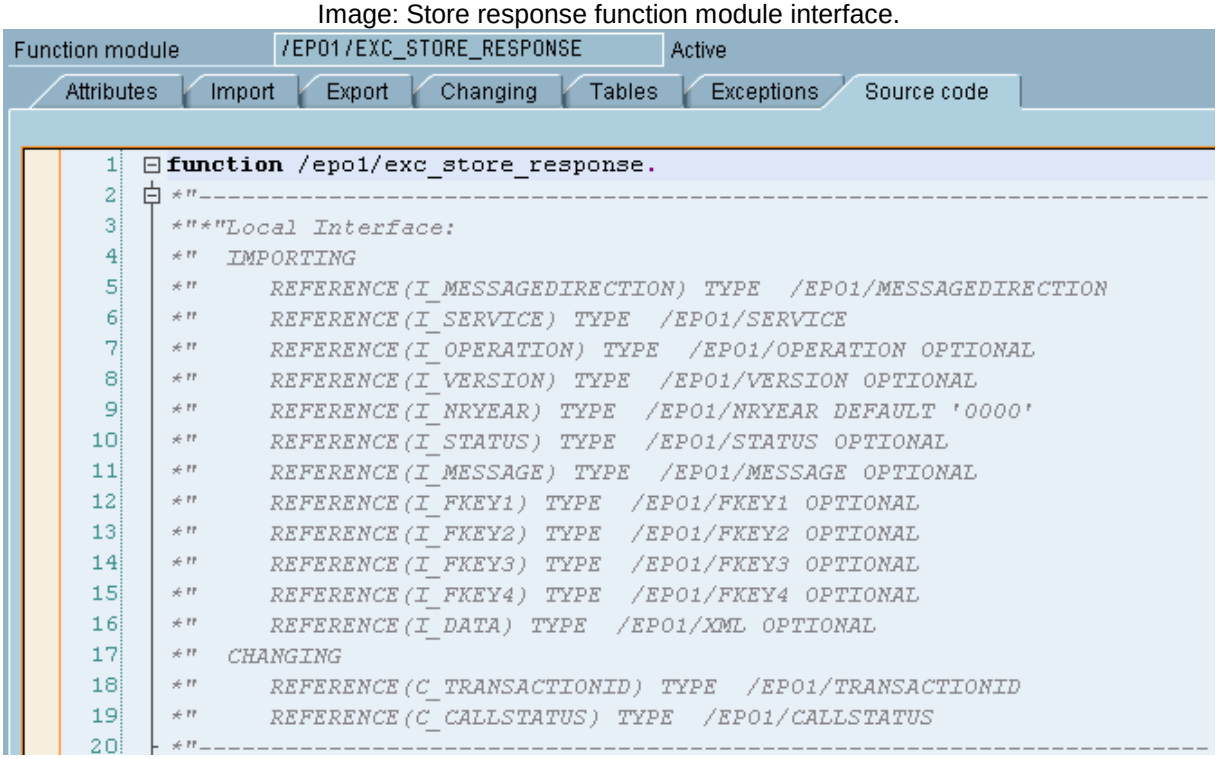


5.2 /EPO1/EXC_STORE_RESPONSE INTERFACE

The function is exactly the same as the previous one, but this one permits you to store response message only. You can use it for http(s) protocols for example.

Notice: The transaction ID parameter is a CHANGING parameter. Therefore you can use the transaction ID which was generated when storing the request.

Image: Store response function module interface.



6 MONITORING FUNCTIONALITY

Monitoring functionality of EPO XML Connector allows you to monitor all interfaces from and to your SAP systems. You can monitor all interfaces implemented within the EPO XML Connector implicitly (EPO runtime, SAP runtime, EPO Client, SAP Client). And you can also monitor all other interfaces, by just adding one of the two API function modules (see chapter 5) to your existing interfaces.

EXC monitoring sends internal (SAP Office), external (E-Mail) or custom (user exit) messages when “something goes wrong”. In detail this means you can send monitoring messages, when transactions finish with statuses which correspond to statuses set in a monitoring profile. A monitoring profile can be assigned to any EXC service.

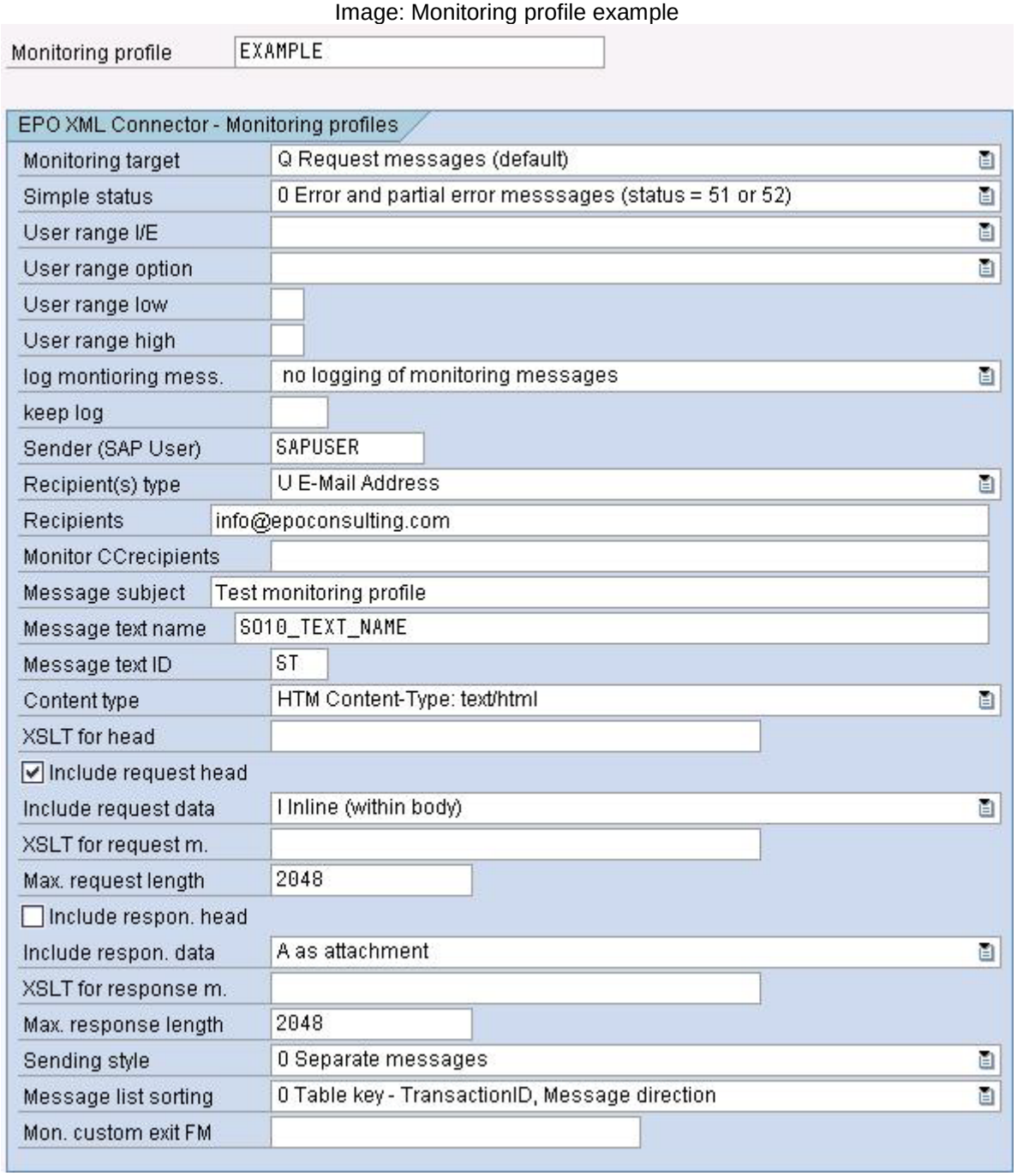
The functionality is executed always when processing a transaction with monitoring profile set in service's configuration (at runtime). Or you can also manually run /EPO1/MONITOR program (image below) to send monitoring message(s) for selected set of transactions, or you can schedule a job to run this program (scheduled).

6.1 SETTING UP MONITORING PROFILES

Monitoring profiles hold information about when and how a monitoring message should be sent. Monitoring profiles can subsequently be assigned to each service configuration or they can be used in the EPO XML Connector monitor.

Area menu: *EPO XML Connector Configuration -> Monitoring ->Create/Change monitoring profile*
Transaction: */EPO1/MONITOR12*

Image: Monitoring profile example



Description of monitoring profile parameters:

- Monitoring profile — Name of the profile, which you fill into the service configuration

- Monitoring target — Monitoring message is sent when status of an EXC message meets the status set in monitoring profile. Here you decide what status the monitoring is looking at; request (default) or response message status.

  Note: Only request message status is set automatically in error cases so unless you set status of response message in user exit, the request message status is the only viable option here.

- Simple status — Status values for which the monitoring message will is sent. Options here are:
  - Error and partial error messages (status = 51 or 52)
  - Error messages only (status = 51)
  - Stored messages (status = 50)
  - Successful messages (status = 53)
  - All messages (any status)
  - User defined status range (user range)

- User range I/E — Include or exclude user range
- User range option — User range relational operators (BT, CP, EQ, etc...)
- User range low — Status value
- User range high — Status value

  Using these four fields you can define range of statuses for the profile, which is used when 'Simple status' field is set to 'User defined status range (user range).

- Log monitoring mess. — If you enable logging of monitoring messages here, the log records are created in /EPO1/MON_LOG table. The logged information is : Service, operation, version, creation date and time, monitoring profile name, monitoring source (R-runtime, M-manual), the SUBRC error number (when sending of the message failed), error message, TransactionID and recipient type.

- Keep log — Number of days for keeping the log records. When empty, all records are kept, otherwise the log records older then specified are deleted.

- Sender (SAP User) — SAP user name used as a sender of monitoring message.

- Recipient(s) type — According to this value, the monitoring message is send to:
  - SAP User (SAP Office)

- E-Mail address
- EXC customer exit (to be programmed individually)

In case of customer exit, function filled in 'Mon. custom exit FM' field of profile is called.

- **Recipients** — Semicolon (;) separated list of recipients of the monitoring message. Sap user names, e-mail addresses or custom recipients list for the user exit FM.
- **Monitor CCrecipients** — Same as above, but these are “carbon copy” recipients.
- **Message subject** — Text for subject of the monitoring message (string).
- **Message text name** — Text for body of the monitoring message. This is name of text object created using transaction SO10 (SAP standard texts).
- **Message text ID** — Text ID of a SAP standard text object (SO10).
- **Content type** — TXT (text/plain) or HTM (text/html) content type for message body. Make sure you set it to HTM you want the message body to properly show html, which you pass to it.
- **XSLT for head** — XSLT transformation, which you can use for formatting /epo1/xmlhead structure (either request or response dependent on 'Monitoring target' setting of the profile.
- **Include request head** — When checked, the body of message will contain information from /epo1/xmlhead table for request message of the monitored transaction.
- **XSLT for request m.** — The transformation specified here you can use for formatting the request head (/epo1/xmlhead structure) information set to be included in field above.
- **Include request data** — You can also include the request data into body attach it as a file to the monitoring message.
  Options are:
  - when this field is empty, no request data are sent
  - as attachment – data file attached to the message
  - inline (within body) – data included in message body
- **Max. request length** — Maximum size in bytes of the included request data. When empty or zero, all the data are included, otherwise the string is shortened to the size specified here.
- **Include respon. head**
- **Include respon. data**

  These fields are equivalents of the four fields above, but for response head and data information.

- XSLT for response m.
- Max. response length

- Sending style — Options for sending set of transactions in one monitoring message. This field only applies to /EPO1/MONITOR program. In opposite, when monitoring runs on transaction being processed, there is always only one message per one transaction possible.

  You can choose to send:
  - Separate messages
  - Message list per service
  - Message list per service and operation
  - Message list per service, operation and version
- Message list sorting — In case you selected message list in field above, you can choose the sorting of the included monitoring messages here.

  Sorting options:
  - Table key (TransactionID, Message direction)
  - Service, operation, version
  - Creation date, creation time, changed data, changed time
  - Sorting 1 and 2 – Service, operation, version, creation date, creation time, changed data, changed time
- Mon. custom exit FM — Here you can specify your function module to be executed when you set 'Recipients type' field to 'EXC customer exit (to be programmed individually)'. This is the option for you to program your own sending of the monitoring messages.

6.1.1 MONITORING CUSTOM EXIT FUNCTION MODULES

Instead of using one of the build-in monitoring message function (SAP office mail, E-mail), you can implement your own monitoring message function. This can be used to send monitoring messages either

- with other techniques like fax, SMS, ftp,...
- or to link it with another monitoring system.

The call of the custom exit function code:

```
CALL FUNCTION me->profile-moncustomexitfm
  EXPORTING
    i_mon_profile      = me->profile          " type /epo1/mon_profs (structure)
    i_service          = l_service            " type /epo1/service
    i_operation        = l_operation          " type /epo1/operation
    i_version          = l_version            " type /epo1/version
```

```
  i_transactionid      = l_transactionid                " type /epo1/transactionid
  i_request_head       = me->request_head               " type /epo1/xmlhead (table)
  i_request_data       = me->request_data               " type /epo1/xmldata (table)
  i_response_head      = me->response_head              " type /epo1/xmlhead (table)
  i_response_data      = me->response_data              " type /epo1/xmldata (table)
  i_message_body       = me->message_body               " type /epo1/mon_message_body (table)
  i_testmode           = i_testmode                     " type /epo1/testmode
IMPORTING
  e_mon_status         = l_mon_status                   " type /epo1/status
  e_mon_message        = l_mon_message                  " type /epo1/message
CHANGING
  c_recipients_type    = me->profile-recipients_type    " type /epo1/mon_recipients_type
  c_recipients         = me->profile-recipients         " type /epo1/mon_recipients
  c_cc_recipients      = me->profile-cc_recipients      " type /epo1/mon_cc_recipients
  c_mon_subject        = me->profile-subject.           " type /epo1/mon_subject
```

6.2 EPO XML CONNECTOR MONITOR

The EPO XML Connector monitor allows you to select messages according to their status and

- list messages in test run
- or to send monitoring message in “Production mode”.

Area menu: *EPO XML Connector Configuration → Monitoring → EPO XML Connector monitor*
Transaction: */EPO1/MONITOR*

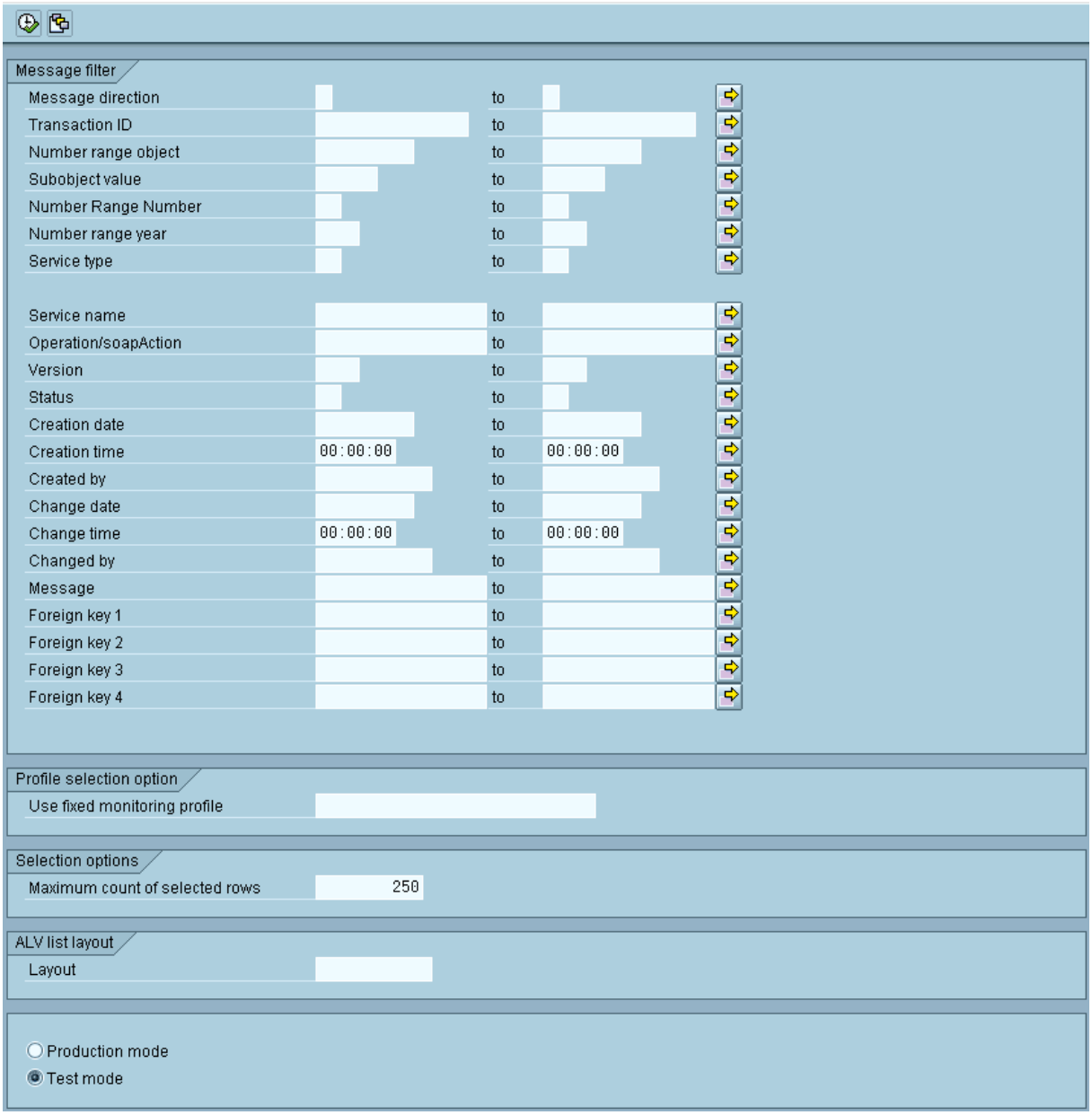
There are 2 main options for selecting messages:

a) “Use fixed monitoring profile” left empty
b) “Use fixed monitoring profile” filled with a monitoring profile

With option a) only services configured with a monitoring profile will be selected. Sending of monitoring messages is done according to the different monitoring profiles.
With option b) all messages will be selected and sending of monitoring messages is done according to this monitoring profile.

Image: Monitoring program selection screen

EPO XML Connector Monitor



7 MESSAGE DATA MAINTENANCE (INBOUND & OUTBOUND)

During runtime of any service each message sent or received in SAP with the EPO XML Connector will get a unique transaction ID. Request and response messages of a particular service call will get the same transaction ID, but different message direction information (I or O). If a service is configured to store messages, those messages can be obtained, viewed and edited with the data maintenance transaction of the EPO XML Connector.

All functions are available for all kind of messages. You can find them in our area menu in section *EPO XML Connector Data Maintenance* and they are divided into two sections for better orientation – *Inbound XML messages (Call of SAP Services)* and *Outbound XML messages (Call of external Services)*.

7.1 (RE) PROCESS A STORED XML REQUEST MESSAGE

Processing or reprocessing of XML messages manually can be done via our menu functions.

Inbound services (Incoming requests to a SAP services – EPO Runtime and SAP Runtime services):

EPO XML Connector Data Maintenance → Inbound XML messages →
/EPO1/EPORTINPROC – In: (Re) Process EPO Runtime XML message
/EPO1/WSINPROC – In: (Re) Process SAP Runtime XML message

Outbound services (Outgoing request to an external service – EPO Client and SAP Client services):

EPO XML Connector Data Maintenance → Inbound XML messages →
/EPO1/EPORTOUTPROC – Out: (Re) Process EPO Client XML message
/EPO1/WSOUTPROC – Out: (Re) Process SAP Client XML message

Each of these transactions call different programs to (re)process a message using a different technique, but they all have got the same selection screen, where you select the message(s) for processing or reprocessing.

The only condition needed for (re)processing of a message is that its status code needs to be less than 53 (processed successfully). However, if you need to reprocess such message again you can change the status code manually, using the function from administration menu (or your SAP administrator can) – EPO XML Connector Administration → */EPO1/SETSTATUS – Set status of XML message manually.*

7.2 DOWNLOAD AND EDIT A STORED XML MESSAGE

There are two functions in our area menu enabling you to download an XML message as a file(s). Every record in the database tables is represented by a single file. You can store the file(s) to the server or local machine directory, which you will

need to input in selection screen of the chosen program. There is a common selection screen in each of the functions where you will be able to choose which message you want to download. The files are saved in binary format which keeps all the encoding information intact. The names of the files are generated automatically using “epo”, client, TransactionID and Messagedirection with the following structure: “EPO_<client>_<transactionid>_<messagedirection>.<suffix>”, where the suffix is taken from “Message format” field of configuration of corresponding service. The file name then will look like this: epo_910_000000000000054_I.XML” for incoming message of transaction 54. The client, transaction id and message direction from the file name are used when uploading / updating the XML message. So you don’t really want to change them, if you want to successfully update the same record in the database. However this function is here for you to be able to edit the content of the file – XML message itself. You can use any XML editor for these purposes; many of them are available on the internet.

Note: Using the outbound function for incoming service, you are able to download and therefore edit a response XML message, but can not upload and update the response using the upload function. You will also see the message “This is response message” in the message column of the ALV list displayed after you downloaded such file.

The functions are:

Inbound services (Incoming requests to a SAP services – EPO Runtime and SAP Runtime services):

EPO XML Connector Data Maintenance → Inbound XML messages → /EPO1/DOWNLOADXMLIN – In: Download XML message

Outbound services (Outgoing request to an external service – EPO Client and SAP Client services):

EPO XML Connector Data Maintenance → Outbound XML messages → /EPO1/DOWNLOADXMLOUT – Out: Download XML message

7.3 UPLOAD A STORED XML MESSAGE

After having downloaded and edited a file (previous paragraph) you can upload it back and update the database tables. You can upload the file from the server or local machine by choosing the right option in the selection screen and selecting the right file by browsing in local machine files, or typing the path and file name of the server file. The file name must correspond to the structure created by the downloading program (see previous paragraph). The information from it is used for finding the record, which needs to be updated. This function uploads the selected file and updates the record in the database tables. It cannot be used to insert a new record in any case. You will not be able to upload any file which contains a response XML message using these functions (The responses are answers from the services, so it does not make any sense to change them).

Functions for uploading files:

Inbound services (Incoming requests to a SAP services – EPO Runtime and SAP Runtime services):

EPO XML Connector Data Maintenance → Inbound XML messages → /EPO1/UPLOADXMLIN – In: Upload XML message

Outbound services (Outgoing request to an external service – EPO Client and SAP Client services):

EPO XML Connector Data Maintenance → Outbound XML messages → /EPO1/UPLOADXMLOUT – Out: Upload XML message

7.4 File upload of a new XML request message (instead of using HTTP)

Please use this program only for ad-hoc uploads. For uploads of files from the SAP server as a regular service, you must use program /epo1/exc_fileruntime (Transaction can be found in Configuration - EPO runtime part of area menu).

Reading the previous two paragraphs, you know that you can download, edit, upload and update the XML message to/from files. The functions described here enable you to insert new request message into the database tables. They will not update any record in any case. Unlike the uploading functions, these ones do not need the specific file name; they will upload any file, which you choose in the selection screen, but you will have to assign the file to the right service by inputting service's name, operation and version, so that it can be (re)processed afterwards. When running the upload program you will tell the EPO XML Connector to create a new transaction id and insert a new request message to the database tables. It will not process the request automatically, you will need to do it manually using (re)processing function (please see section 7.1).

The functions for inserting new record from a file are:

Inbound services (Incoming requests to a SAP services – EPO Runtime and SAP Runtime services):

EPO XML Connector Data Maintenance → Inbound XML messages → /EPO1/UPLOADXMLIN – In: Upload XML message

Outbound services (Outgoing request to an external service – EPO Client and SAP Client services):

EPO XML Connector Data Maintenance → Outbound XML messages → /EPO1/UPLOADXMLOUT – Out: Upload XML message

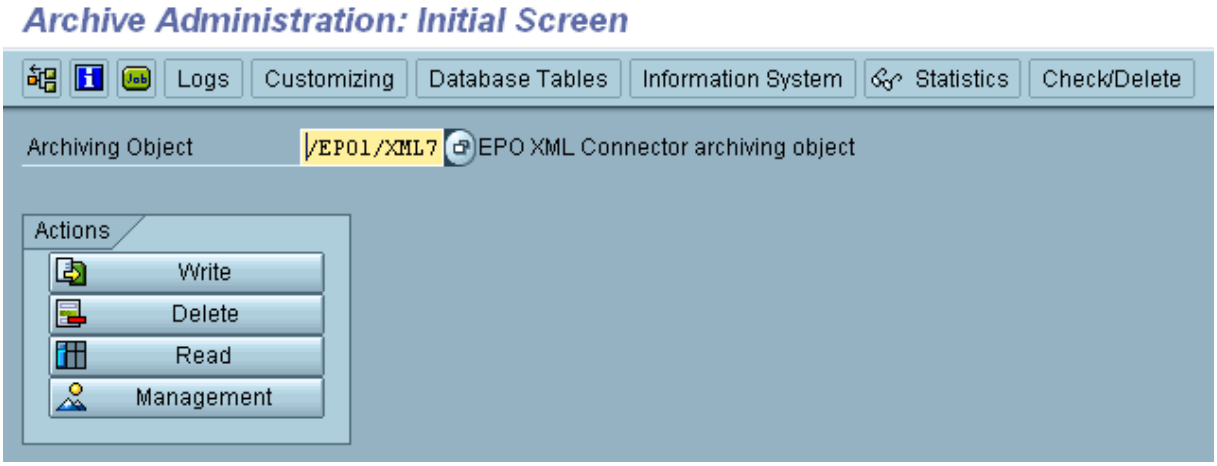
Note: The concept of the EPO XML Connector allows inserting any file format (not just XML). If it is not an XML format, you will need the EPO runtime with a processing function module to process the message.

8 ADMINISTRATION OF THE EPO XML CONNECTOR

8.1 ARCHIVING XML MESSAGES

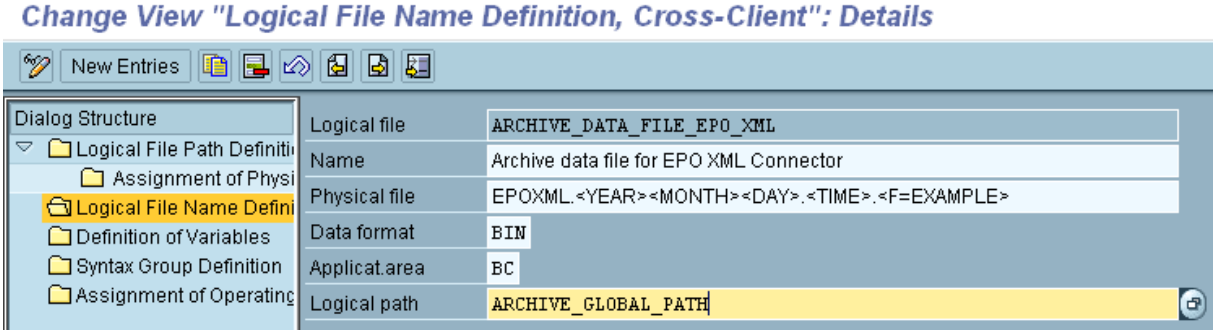
For archiving (write, delete and read archiving routines) of XML messages we created the /epo1/xml7 (/epo1/xml in WAS 6.20) archiving object, which you can use in SAP archive administration – transaction SARA. You can call the transaction from our area menu (*EPO XML Connector Administration → Archiving XML messages → /EPO1/SARA7 – Archive Administration for object /EPO1/XML7*), which will open the SARA transaction with the object name already filled in, so you don't need to remember it.

Image 30: SARA – Archive Administration using object /EPO1/XML7



The only thing you will most likely need to change for the archiving object is the Logical path in the Logical file name definition, so it corresponds with your archiving system setup (image 31). As you probably know, you can do this in transaction FILE, which you can also access from our area menu (*EPO XML Connector Administration → Archiving XML messages → FILE – Cross-client file names / paths*).

Image 31: Logical file name definition for object /EPO1/XML7



To let you have a look at the /epo1/xml7 archiving object definition there is the AOBJ transaction in our menu (*EPO XML Connector Administration → Archiving XML messages → AOBJ – Archiving object definition*)

8.2 AD-HOC DATA OPERATIONS

Please do not mismatch these operations with services. They are here to provide data access in case there is a need to manually alter the data or database.

8.2.1 DOWNLOAD XML MESSAGES TO DIRECTORY

Rather then downloading stored XML messages by direction as it is proposed in data maintenance area menu, you can download all selected messages (based on selection) to a single directory either to the server or local machine. The files are stored in binary format to keep the encoding. The names of the files are created automatically from the 'epo', client, TransactionID and direction of the message (Messagedirection). The suffix each file is set to the message format, which is configured with the service (please see section 3.1). Error messages occurring during download, if any, you can view in the ALV list field "message" after running the program.

8.2.2 UPLOAD XML MESSAGES FROM DIRECTORY

With this function you can upload all files, which were downloaded before, from a single directory either from the server or local machine. This procedure only updates records in the database tables, it does not insert anything in any case and that is why uploading requires existing records in the database tables. The key is TransactionID and Messagedirection, which is stored in the file name, so the file name structure must correspond to the structure created by downloading program – this is checked exactly when uploading files and if the file name structure is not correct, files are not uploaded. These files are treated as wrong files, so you will not be able to see them anywhere in the ALV list. All other errors are written to the message field of corresponding records, but only in local table – they are not stored to the database – and you can see them with all the other stored fields (except the XML message of course) in the ALV list displayed after the procedure.

Note: When this function runs, it always uploads all the correct files in the selected directory and then it applies the filter you have entered in the selection screen of the program, so it can take some time when you have many files, but you want to upload just one or couple of them.

8.2.3 INSERT NEW XML MESSAGES FROM DIRECTORY

Please use this program only for ad-hoc uploads. For uploads of files from the SAP server, as a regular service, you must use program /epo1/exc_fileruntime (Transaction can be found in Configuration - EPO runtime part of area menu).

File handling functionality would not be complete without having the ability to insert new records to the database tables. Similar to uploading it uploads all files from a selected directory, either from the server or local machine. It creates new TransactionIDs for each of the uploaded files and inserts them into the database tables. You need to assign the files to the existing service by inputting service name, operation and version in the selection screen. It does not update any record in any case and it does not check the file name, so all the files are inserted. It does not

process the requests automatically. You will need to (re)process them afterwards (please see section 7.1). Errors opening and/or reading files are written to the message field of the local table – not inserted to the database – and displayed with all the other fields in the ALV list after running this program.

8.3 OTHER ADMINISTRATIVE FUNCTIONS

/EPO1/SETSTATUS – Set status of XML message manually

You can change the status code of a XML message using this program. It allows you to reprocess a message, which has already been processed, by setting the code to the number less than 53.

SMICM – ICM monitor
SICF – HTTP service hierarchy maintenance

Standard SAP transactions you will need to use administering EPO XML connector.

/EPO1/SETLICENSE – Load license key for EPO XML Connector

Utility for loading the license key file for EPO XML Connector

/EPO1/TEMPLATESUNCOM – Uncomment / comment template programs

Here you can find a program to uncomment or comment the template programs (examples), which are delivered with the EPO XML Connector.

All template programs delivered with the EPO XML Connector are commented out. Note: This is to avoid syntax errors in your system, because you might not have installed all SAP components which are used in the templates.

If you want to use a template program, you must first uncomment it. Often this is not necessary, because you will need to create a copy of a template program in the Z* or Y* namespace (see developing of new services).

You can uncomment or comment all template programs in one single run.

The comment / uncomment program can only be used for template programs with the EPO XML Connector. No other programs can be commented or uncommented.

9 INTEGRATION SOLUTIONS BASED ON EXC

9.1 xHTML INTERACTIVE FORMS

This solution enables you to generate prefilled xHTML forms out of SAP. It is directly built-in into the EPO XML Connector.
It is similar to Adobe pdf interactive forms, but instead of using pdf forms you can use xHTML forms. It is also similar to BSP (SAP Business Server Pages) with the advantage of being much simpler and again a clear separation between front-end and back-end.

9.1.1 xHTML OUTPUT FOR EPO RUNTIME AND CLIENT

The title saying it, this part of xml processing applies to EPO Runtime and EPO Client services only. The idea is to be able to serve HTML form prefilled with data from SAP. To give full control to you, the prefilling is done in customer programmed function modules, where you call pre-defined XSLT transformations. Example XSLT transformation are delivered with the EPO XML Connector (used in example prefill function modules):

- /EPO1/EXC_CHANGE_XHTML_ATTRIB replaces or adds attributes and its values in existing elements.
- /EPO1/EXC_CHANGE_XHTML_ELEMENT replaces value in existing element of the template.

The xHTML template must be stored in SAP MIME repository (transaction SE80, MIME Repository button). It has to be in xHTML format to be able to do XSLT transformations. You can find xHTML specification on http://www.w3.org/TR/xhtml1/.

The prefill function module call is placed before outbound XSLT transformation in both EPO runtime and EPO Client.

EPO runtime and EPO Client both check whether there is a xHTML configuration set for the called service, configuration and version. If it is set, it reads the configured MIME object and passes it together with XML data to the prefill function module.

9.1.2 xHTML OUTPUT CONFIGURATION

Area menu: *EPO XML Connector Configuration → Maintain EPO xHTML configuration*
Transaction: */EPO1/XHTML12*

Image: EPO xHTML output configuration example

Service name, Operation, Version: EPO Runtime or Client service identification for xHTML output processing

MIME Object URL: xHTML template object URL (SE80, MIME Repository)

Prefill FM: Customer programmed function module for template transformations

HTTP content type: EPO Runtime only, custom HTTP header value for response, text/xml (EPO Runtime default) if left empty

9.1.3 xHTML Output Prefill FM Interface

Below is the interface of template function module, placed in /EPO1/EXC_XHTML package which you can use for filling the xHTML template with data from SAP.

```
function /epo1/exc_xhtml_prefill.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_SERVICE) TYPE  /EPO1/SERVICE
*"     REFERENCE(I_OPERATION) TYPE  /EPO1/OPERATION
*"     REFERENCE(I_VERSION) TYPE  /EPO1/VERSION
*"     REFERENCE(I_XML) TYPE  /EPO1/XML
*"     REFERENCE(I_MIME) TYPE  /EPO1/XML
*"  EXPORTING
*"     REFERENCE(E_XML) TYPE  /EPO1/XML
*"     REFERENCE(E_CALLSTATUS) TYPE  /EPO1/CALLSTATUS
*"----------------------------------------------------------------------
```

I_SERVICE, I_OPERATION, I_VERSION: EPO Runtime or Client service identification

I_XML: For EPO Runtime this is response XML data after being saved (if configured) and before outbound XSLT transformation (if configured) is applied, for EPO Client it is request XML data passed to /EPO1/EPOCLIENT FM, after being saved (if configured), before outbound XSLT transformation (if configured).

I_MIME: The xHTML template read from MIME Repository (SE80)

E_XML: Parameter for filled or transformed template to pass it back to EPO Runtime or EPO Client processing

E_CALLSTATUS: Error information in /EPO1/CALLSTATUS format. If you fill this information and leave E_XML empty it is transformed to simple HTML structure containing the information, if you want to create different error structures, do so in E_XML and leave this empty

9.1.4 xHTML Output Prefill FM Creation

First step is getting data needed for filling the template from imported XML data. In the template we call transformation id to give us username to read the data using BAPI.

```
" get username from i_xml
call transformation id
  source xml i_xml
  result username = l_username.
```

Then we call the BAPI to get user details, but obviously you will use this part to get the data you need. You can see simple error handling here, which gives back callstatus which is then in turn transformed to HTML message.

```
" get user details
call function 'BAPI_USER_GET_DETAIL'
  exporting
    username = l_username
  importing
    address  = l_address
  tables
    return   = lt_return.

" error handling
if l_address is initial.
  " first error to callstatus
  loop at lt_return into wa_return
    where type = 'E' or
          type = 'A'.

    clear e_xml.
    call function '/EPO1/EXC_CSMSG'
      exporting
        i_bapiret2   = wa_return
        i_parameter1 = 'BAPI_USER_GET_DETAIL'
      importing
        e_callstatus = e_callstatus.
    exit.
  endloop.
endif.
```

Last part is transforming the xHTML template using XSLT transformations to prefilled xHTML form. Before this bit there is also filling of these two internal tables, you can find whole code in /EPO1/EXC_XHTML_PREFILL function module, which you can also use as a template for your own ones.

```
" xHTML form template
e_xml = i_mime.

" apply attribute value changes
loop at lt_attributes into wa_attributes.
  call transformation /epo1/exc_change_xhtml_attrib
    parameters
      element   = wa_attributes-element
      id        = wa_attributes-id
      attribute = wa_attributes-attribute
      value     = wa_attributes-value
    source xml e_xml
    result xml e_xml.
endloop.

" apply element value changes
loop at lt_elements into wa_elements.
  call transformation /epo1/exc_change_xhtml_element
    parameters
      element   = wa_elements-element
      id        = wa_elements-id
      value     = wa_elements-value
    source xml e_xml
    result xml e_xml.
endloop.
```

9.2 METASTORM BPM / SAP INTEGRATION

Integration of Metastorm BPM (former name e-Work) with SAP was previously done with the Metastorm BPM Connector. With the release of the EPO XML Connector integration of Metastorm BPM is covered by this general, more powerful connector. The Metastorm BPM Connector is retired.
There is a separate, in-detail-documentation available for integrating Metastorm BPM with SAP. EPO Consulting also provides a product called B2B WS Integrator for Metastorm BPM, which enables web service calls (SAP Inbound) for Metastorm BPM.

9.2.1 INTEGRATION OF METASTORM BPM USING WEB SERVICES

Metastorm has released new, improved Web Service functionality with Metastorm Release 7.5 SR1. We have successfully tested the EPO XML Connector with these Web Services. There is a full documentation with examples available which can be requested from EPO Consulting or downloaded from our website. Here follows only a short documentation.

9.2.2 INTEGRATION OF METASTORM BPM USING FILE INTERFACES

You can also use the file interface functionality of the EPO XML Connector. In principle it means creating a file in Metastorm BPM and loading it into SAP or vice versa.

9.2.3 USING B2B INTEGRATOR FOR METASTORM BPM

The B2B Integrator from EPO Consulting is a software tool allowing you to create Web Service interfaces (SAP Inbound) with Metastorm BPM.

The B2B Integrator is a software tool, which needs to be installed on the Metastorm BPM Server.
When creating Metastorm BPM processes the B2BIntegrator library must be attached to processes. This library automates the call of SAP Web Service calls (in the Integration Wizard of the Metastorm designer).

9.3 MS EXCEL / SAP INTEGRATION

Integration of MS Excel with SAP is brought to a new level by the EPO XML Connector. There are several options available to set up any MS Excel to SAP integration. The technique used is always calling a web service out of Excel, which is provided by SAP with the power of EXC (SAP Inbound).

Option 1: Calling a SAP EXC web service with GET in VB
Option 2: Calling a SAP EXC web service with POST in VB
Option 3: Calling a SAP EXC web service with using XML Transmitter Office add-in (a product developed by EPO Consulting)

Option 4: Calling a SAP EXC web service using .NET programming options.

EPO Consulting will deliver examples and pre-packed solutions in the coming months.

10 APPENDIX 1: USING XML TRANSMITTER FOR TESTING AND DEPLOYING WEB SERVICES

When developing your web services using EPO XML Connector you will need to test them. For this purpose we have created a special application which we deliver together with the EPO XML Connector – XML Transmitter. We are using it heavily ourselves developing and testing the connector, so we think it can help you with creation and testing of your own services.

The application enables you to create HTTP request, SOAP or not, post it to the server and see the response in the same window. You can type in your XML messages or create them from WSDL description; you can use XSLT transformations on them; you can even set your own special HTTP headers using this program. The application window is divided into left and right side, where the left side contains all the information about the request you are posting and the right side which shows the response after it has been received. Everything you change on posting side you can store to the configuration file and use it over and over again. You can see some screenshots of this program in section 3.2.5 - Testing an EPO runtime service and section 3.3.2.8 - Testing a SAP runtime service.

Create WSDL functionality
A very useful function of the XML Transmitter is “Create WSDL”. It enables you to generate a WSDL file from a request XML, response XML and the URL (URI). You can use this functionality not just with SAP web services.

The XML Transmitter has got many more functions, which we are not going to describe in here. It can be found in its own documentation.

11 APPENDIX 2: DEMONSTRATION OF ABAP SERIALISATION

As of Release 6.10 ABAP contains the statement CALL TRANSFORMATION that allows you to transform ABAP data to XML and vice versa. There is simple demo program in your EXC installation, which shows how this transformation can be used. The program selects data from T000 and CVERS tables, transforms it into asXML, which is displayed afterwards. Then it uses the same transformation, but in the other direction, to process the asXML back to local table T000, which is displayed as ALV-lists. Then it does the same with table CVERS. The transformation of asXML to tables (T000 and CVERS in this example) could be also done in one step.

AsXML is the XML representation of SAP ABAP programming language structures.

The demo program source

```
*&---------------------------------------------------------------------*
*& Report  /EPO1/Z_TRANSFORMATIONID
*&---------------------------------------------------------------------*
*& Company:   EPO Consulting                                           *
*& IP Rights: Intellectual Property Rights and all other rights are    *
*&            held by EPO Consulting.                                  *
*&            Copying or Modifying this program is only allowed with   *
*&            written consent of EPO Consulting.                       *
*& Author:    MH                                                       *
*& Date:      March 2008                                               *
*& Desc.:     Example of TRANSFORMATION ID                             *
*&---------------------------------------------------------------------*
REPORT  /epo1/z_transformationid.

TYPE-POOLS: slis.
TABLES: t000,cvers.
DATA: lt_t000  TYPE TABLE OF t000,
      lt_cvers TYPE TABLE OF cvers,
      l_return TYPE bapiret2,
      as_xml   TYPE xstring.

SELECTION-SCREEN BEGIN OF BLOCK t000 WITH FRAME TITLE text-001.
SELECT-OPTIONS: so_mandt FOR t000-mandt.
SELECTION-SCREEN END OF BLOCK t000.

SELECTION-SCREEN BEGIN OF BLOCK cvers WITH FRAME TITLE text-003.
SELECT-OPTIONS: so_comp FOR cvers-component.
SELECTION-SCREEN END OF BLOCK cvers.

START-OF-SELECTION.

  SELECT * FROM t000  INTO TABLE lt_t000
    WHERE mandt IN so_mandt.

  SELECT * FROM cvers INTO TABLE lt_cvers
    WHERE component IN so_comp.

END-OF-SELECTION.

  " transform the tables to asXML
  TRY.
      CALL TRANSFORMATION id
        SOURCE
          t000  = lt_t000
```

```
          cvers = lt_cvers
        RESULT XML as_xml.
    CATCH cx_root.
      " no error handling in this example
      EXIT.
  ENDTRY.

  " empty the tables
  REFRESH: lt_t000,lt_cvers.

  " view the asXML
  PERFORM view_xml USING as_xml.

  " the following two tranformations could be done in one step, but in
  " this exapmle we do it in two

  " transform asXML to t000 table
  TRY.
      CALL TRANSFORMATION id
        SOURCE XML as_xml
        RESULT t000 = lt_t000.
    CATCH cx_root.
      " no error handling in this example
      EXIT.
  ENDTRY.

  " ALV list table of banks
  PERFORM list_t000.

  " transform asXML to cvers table
  TRY.
      CALL TRANSFORMATION id
        SOURCE XML as_xml
        RESULT cvers = lt_cvers.
    CATCH cx_root.
      " no error handling in this example
      EXIT.
  ENDTRY.

  " ALV list table of banks
  PERFORM list_cvers.

*&---------------------------------------------------------------------*
* View XML message
*&---------------------------------------------------------------------*
FORM view_xml USING lp_xml TYPE xstring.

  DATA: ls_xml  TYPE string,
        lo_conv TYPE REF TO cl_abap_conv_in_ce,
        li_len  TYPE i.                                    "#EC NEEDED

  li_len = XSTRLEN( lp_xml ).
  lo_conv = cl_abap_conv_in_ce=>create( input = lp_xml ).
  lo_conv->read( IMPORTING data = ls_xml len = li_len ).

  CALL FUNCTION 'PAYLOAD_DISPLAY'
    EXPORTING
      p_payload = ls_xml
      p_title   = text-002.

ENDFORM.                    "view_xml

*&---------------------------------------------------------------------*
```

```
* ALV list table cvers
*&---------------------------------------------------------------------*
FORM list_t000.
  DATA: wx_t000 TYPE t000,                                  "#EC *
        feldcat TYPE slis_t_fieldcat_alv.

  REFRESH feldcat.
  " get field catalog from WX_T000 structure
  CALL FUNCTION 'REUSE_ALV_FIELDCATALOG_MERGE'
    EXPORTING
      i_program_name     = sy-cprog
      i_internal_tabname = 'WX_T000'
      i_inclname         = sy-cprog
    CHANGING
      ct_fieldcat        = feldcat
    EXCEPTIONS
      OTHERS             = 3.
  IF sy-subrc <> 0.
    " no error handling in this example
    EXIT.
  ENDIF.

  " ALV list lt_banklist
  CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
    EXPORTING
      it_fieldcat = feldcat
    TABLES
      t_outtab    = lt_t000
    EXCEPTIONS
      OTHERS      = 2.
  IF sy-subrc <> 0.
    " no error handling in this example
    EXIT.
  ENDIF.
ENDFORM.                                                    "list_t000

*&---------------------------------------------------------------------*
* ALV list table cvers
*&---------------------------------------------------------------------*
FORM list_cvers.
  DATA: wx_cvers TYPE cvers,                                "#EC *
        feldcat TYPE slis_t_fieldcat_alv.

  REFRESH feldcat.
  " get field catalog from WX_CVERS structure
  CALL FUNCTION 'REUSE_ALV_FIELDCATALOG_MERGE'
    EXPORTING
      i_program_name     = sy-cprog
      i_internal_tabname = 'WX_CVERS'
      i_inclname         = sy-cprog
    CHANGING
      ct_fieldcat        = feldcat
    EXCEPTIONS
      OTHERS             = 3.
  IF sy-subrc <> 0.
    " no error handling in this example
    EXIT.
  ENDIF.

  " ALV list lt_cvers
  CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
    EXPORTING
      it_fieldcat = feldcat
```

```
    TABLES
      t_outtab    = lt_cvers
    EXCEPTIONS
      OTHERS      = 2.
  IF sy-subrc <> 0.
    " no error handling in this example
    EXIT.
  ENDIF.
ENDFORM.                                                    "list_cvers
```

12 APPENDIX 3: XSL TRANSFORMATIONS (XSLT)

Since Release 6.10 of the SAP Web Application Server (SAP Web AS), XSL Transformations (XSLT) have been integrated in ABAP via the CALL TRANSFORMATION command. XSLT is the most powerful and advanced technology available for the transformation of XML documents. XML data can be transformed into ABAP data structures and vice versa; however, XSLT is not limited to those types of output. You can also generate HTML documents or plain text files that are made available as loadable assets to other applications. XSLT is widely used and well documented, we don't intend to give all the information in this document – just a brief start point - you can find useful links in the end of this appendix. In EXC we use these transformations for formatting input and output XML messages and for XML to asXML conversion.

12.1 CREATING TRANSFORMATIONS

The easiest way to create an XSLT is using Object Navigator (SE80) as show on images below.

Image: Create Transformation

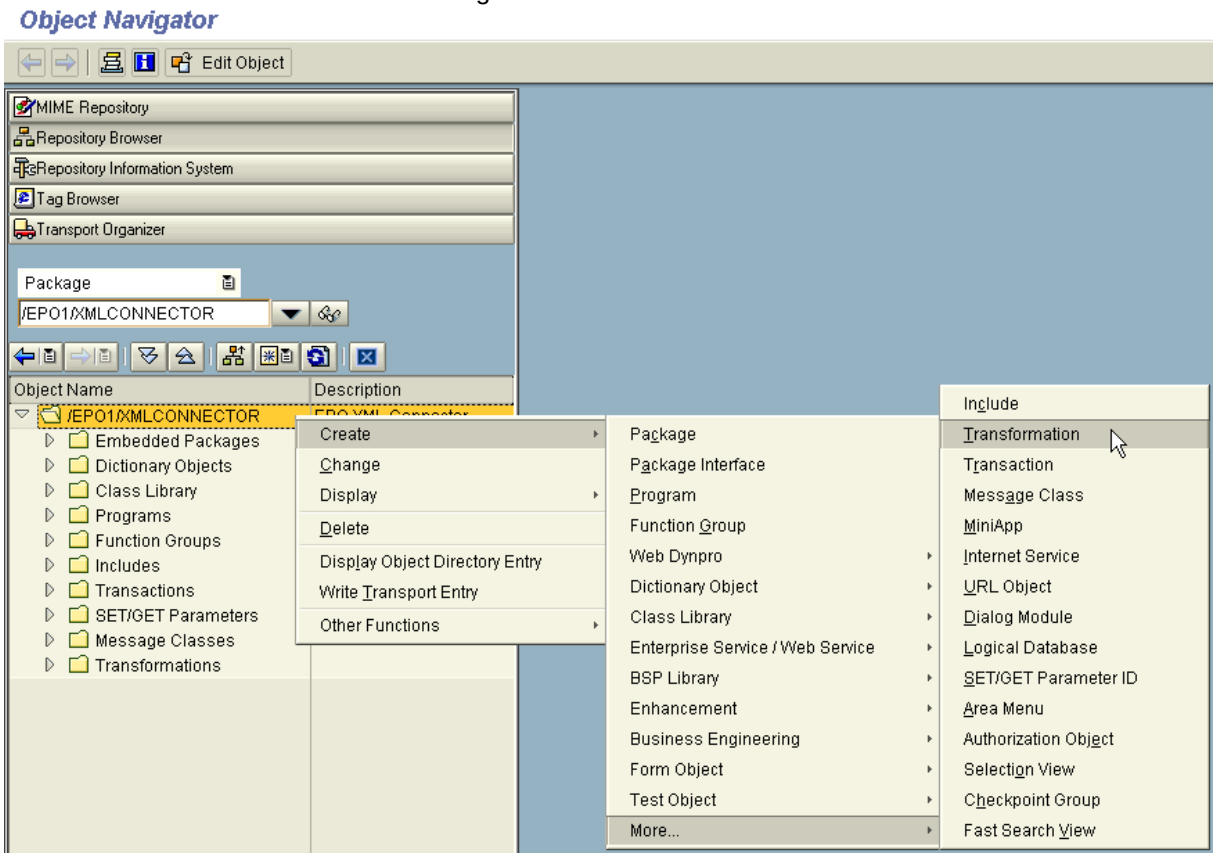
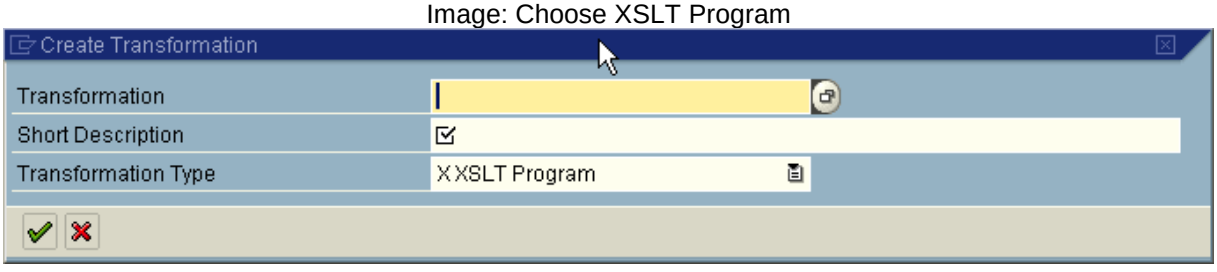


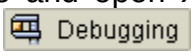
Image: Choose XSLT Program



The XSLT program might look something like this – template SOAP to asXML XSLT:

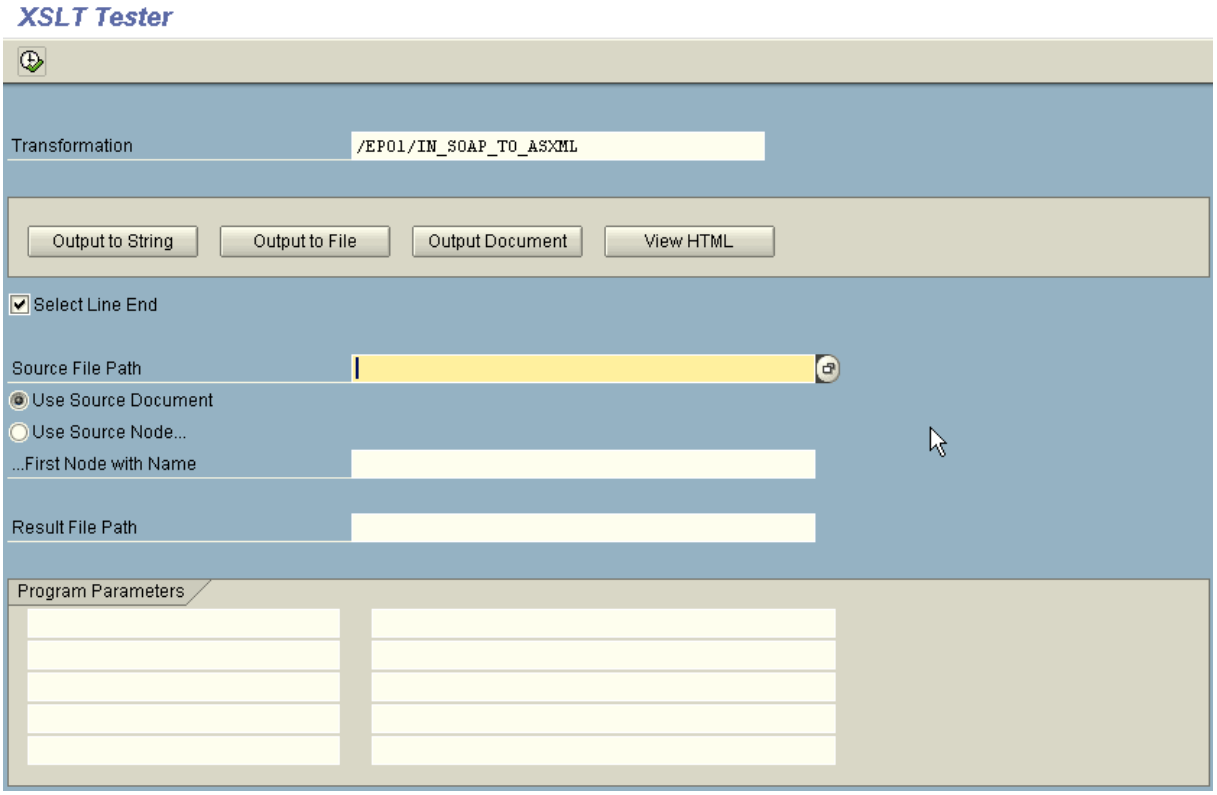
```
<xsl:transform version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:nsm="urn:sap-com:document:sap:soap:functions:mc-style">

   <xsl:template match="nsm:*">
      <xsl:call-template name="asXML"/>
   </xsl:template>

   <xsl:template name="asXML">
      <asx:abap xmlns:asx="http://www.sap.com/abapxml" version="1.0">
         <asx:values>
            <xsl:call-template name="Uppercase"/>
         </asx:values>
      </asx:abap>
   </xsl:template>

   <xsl:template name="Uppercase">
      <xsl:for-each select="*">
         <xsl:element name="{translate(local-name(.),
'qwertyuiopasdfghjklzxcvbnm','QWERTYUIOPASDFGHJKLZXCVBNM')}">
            <xsl:value-of select="text()"/>
            <xsl:call-template name="Uppercase"/>
         </xsl:element>
      </xsl:for-each>
   </xsl:template>
</xsl:transform>
```

12.2 DEBUGGING XSLT IN SAP

When you create and open XSLT in Object Navigator (SE80), you can press  button to test it, 

button to debug it, or you can do it in XSLT Tester (transaction XSLT), which is the program called by Object Navigator anyway.

Image: XSLT Tester transaction



12.3 LINKS TO XSLT DOCUMENTATION

W3C - The Extensible Stylesheet Language Family (XSL)
W3C - XSL Transformations (XSLT)
W3C - XML Path Language (XPath)

SAP Transformation Editor
SAP XSLT Processor Reference
SAP XSLT Debugger

And a lot more out there...

+++ End of document +++