Exposing the XI monitoring functionality as a Web Service

Applies to:

SAP Exchange Infrastructure (SAP NetWeaver Process Integration 7.0)

Summary

The document shows you a way to fetch the XI monitoring data from the various standard SAP tables. It also includes the steps to expose the XI monitoring data to the outside world. By understanding how to do that, one can get an insight to the messages flowing through the XI/PI.

Author: Pooja Pandey

Company: Objectnet Technologies

Created on: 15 January 2008

Author Bio

Pooja Pandey works at The Home Depot as a SAP XI consultant at the Interface Development team. Prior to her role in The Home Depot, she worked for Wipro technologies.

Table of Contents

Introduction 3
WHAT IF…… 3
Structure of the document:...3
Part I: Introducing the important XI tables...3
Part II: Creating a RFC which fetches the basis XI interface monitoring data...4
Part III: Exposing the XI Monitoring RFC as a Web Service...11
Usage...19
Note of Thanks 19
Related Content...19
Disclaimer and Liability Notice...20

Introduction

Business always cares about a status of the messages flowing through XI. SAP Exchange Infrastructure has been known for many of its dazzling features and one of them is the effective and efficient message tracing facility through the transaction - SXMB_MONI. The transaction provides a detailed tracing information for each and every XI interfaces.

WHAT IF……

……, Business wants to see the message status through their handset i-phone or blackberry?

……, You could provide an option to the business to look at the XI message status through the web service?

……, Someone doesn't have to know or remember the SAP transactions to trace the XI message status?

To answer the above "WHAT-IF" questions, I am presenting a document to you, which will give you a direction to dig some of the important SAP tables and exposing it to the real world.

Structure of the Document:

Part I: Introducing the Important XI tables

Part I: Creating a Remote Function module which fetches the basic XI interface monitoring data.

Part II: Exposing the XI monitoring RFC as a web service.

Part I: Introducing the Important XI Tables

Here are the key objects for any XI interface in a runtime environment:

- Inbound message Interface
- Outbound message Interface
- Interface namespace
- Sending Business System
- Receiving Business System
- Interface Mapping
- Message Payload
- Message status

The SAP XI stores the monitoring trace data in the multiple tables. Before I go further, I would like to give some overview on the tables which I have used to fetch the basic XI interface monitoring data.

1. **SXMSPMAST:** (Integration Engine: Message Queue (Master))
   The table is the master table for the monitoring information. You can get the timestamp, interface status, message id, and many more information.
2. **SXMSPEMAS:** (Integration Engine: Enhanced Message Queue (Master))
   The table provides the namespaces, the business systems and message interfaces involved in the interface.
3. **SXMSMSTATT:** (Exchange Infrastructure: Message Status Description)
   This table provides a message status description.
4. **SXMSMSTAT:** (Exchange Infrastructure: Message Status)
   This table is helpful if you want to show the process status icon for the corresponding message state.
5. **SMPPMAP3:** (Mapping Runtime: Mapping)
   The above table gives you the mapping name (if any) used in the interface. It also provides the details on the type of mapping *(ABAP mapping, Java mapping, Generated Mapping, XSLT java mapping, XSLT ABAP mapping)* used for an interface.

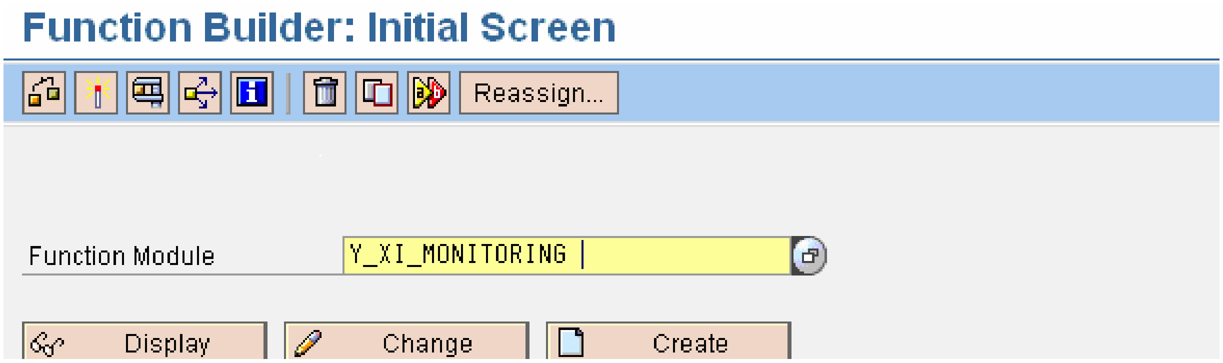
6. **SMPPREL3:** (Mapping Runtime: Mapping Relation)
The table gives an interface mapping details for an interface and its corresponding message interface.

Note: For more information consider the following tables:

- SXMSPERROR: XML Message Broker: Message Queue (Incorrect Entries)
- SXMSCLUP: XMB: Property Cluster
- SXMSPVERS: Integration Engine: Message Version
- SXMSAGGRAW: SAP XI Status Overview: Integration Engine Raw Data
- SXMSALERTLOGGER: XI Alert Logs
- SXMSAEADPMODCHN: XI: Adapter Module Chains
- SMPPSPLIT : XI Mapping: Merge and Split

Part II: Creating a RFC which Fetches the Basis XI Interface Monitoring Data

Step 1: Create a function module (in our case Y_XI_MONITORING) using the transaction SE37.

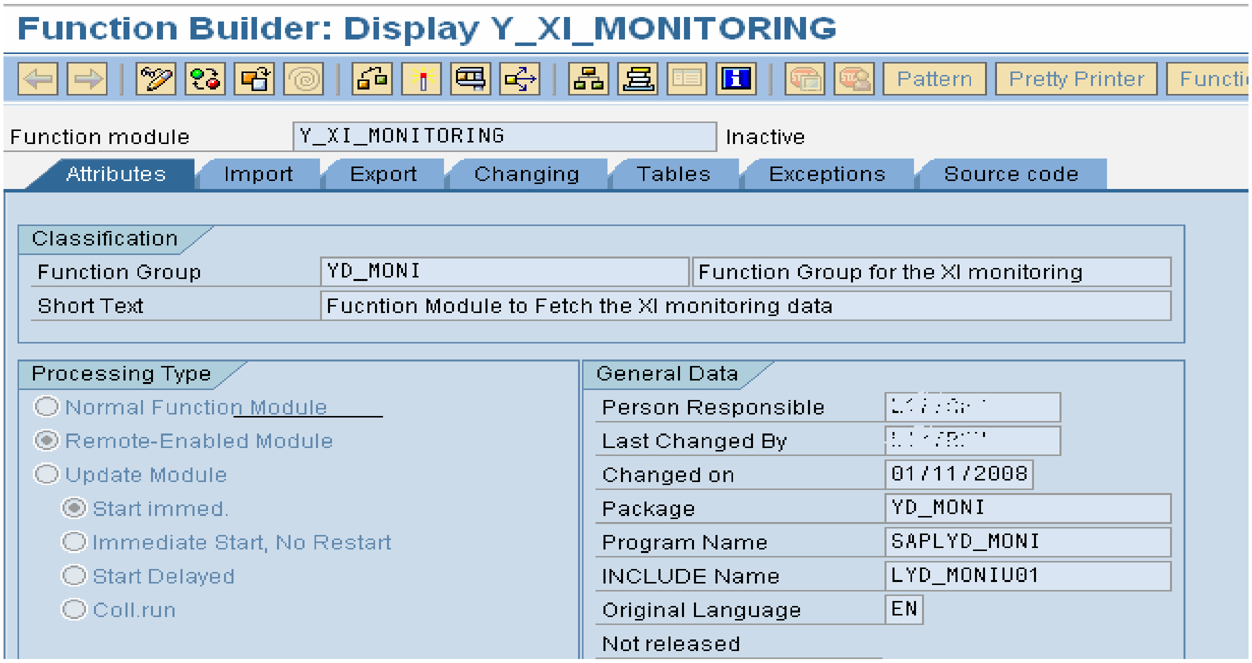


Function Builder: Initial Screen

Function Module Y_XI_MONITORING

Figure 1

Step 2: As we are creating a web service for the XI monitoring message. Go to the Attribute tab and select the Processing type as *"Remote-Enabled Module*" which enables you to expose your function module as a web-service.



Function Builder: Display Y_XI_MONITORING

Function module: Y_XI_MONITORING Inactive

Classification

Function Group	YD_MONI	Function Group for the XI monitoring
Short Text	Fucntion Module to Fetch the XI monitoring data	

Processing Type

○ Normal Function Module
● Remote-Enabled Module
○ Update Module
● Start immed.
○ Immediate Start, No Restart
○ Start Delayed
○ Coll.run

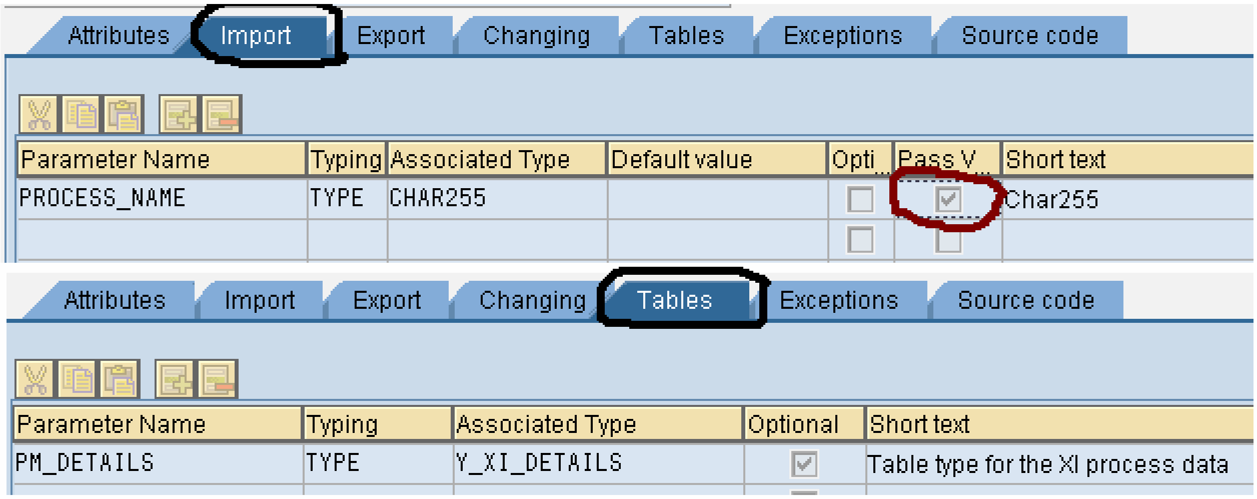
General Data

Person Responsible	[redacted]
Last Changed By	[redacted]
Changed on	01/11/2008
Package	YD_MONI
Program Name	SAPLYD_MONI
INCLUDE Name	LYD_MONIU01
Original Language	EN
Not released	

Figure 2

Step 3: To make the RFC simple, we are providing an XI Message Interface to the RFC and in return, we will be getting the messages in the table (PM_DETAILS) of type Y_XI_DETAILS *(shown in the step 4)* for the given message interface.

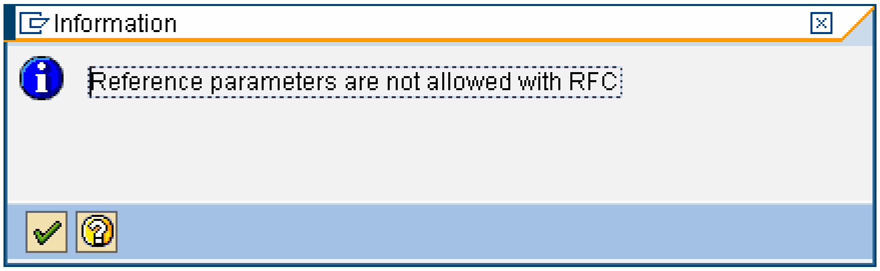
Note: While declaring an import parameter name, you have to make sure that you have checked the “Pass Value” otherwise you will get an error shown in the *figure 4*.



Parameter Name	Typing	Associated Type	Default value	Opti	Pass V	Short text
PROCESS_NAME	TYPE	CHAR255		□	☑	Char255
				□	□	

Parameter Name	Typing	Associated Type	Optional	Short text
PM_DETAILS	TYPE	Y_XI_DETAILS	☑	Table type for the XI process data

Figure : 3



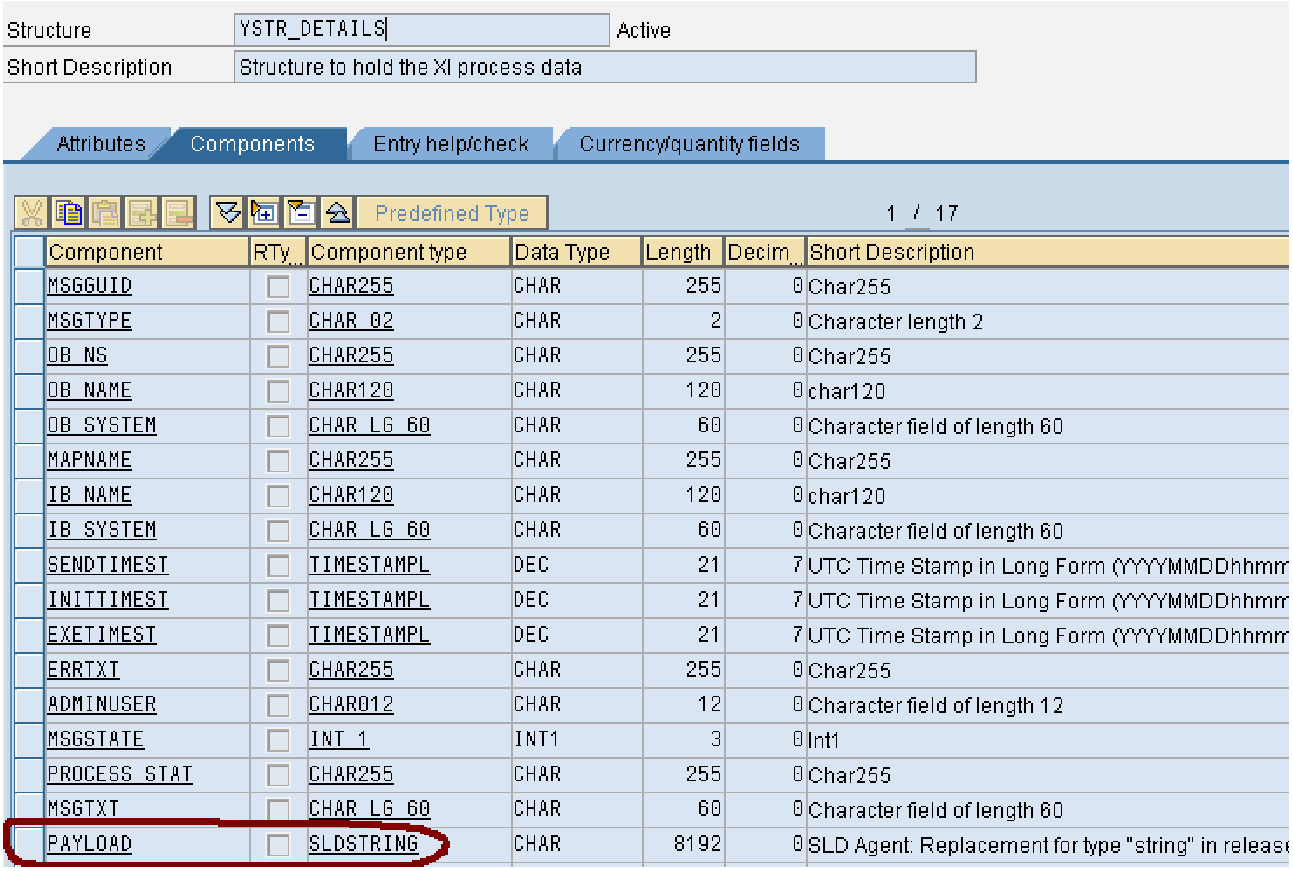
Information

Reference parameters are not allowed with RFC

figure : 4

Step 4: Using SE11, create a structure that holds the XI message monitoring information.

The “PAYLOAD” field needs a special attention; which carries the message payload data. The field needs to define in such a way that it can carry a long string and at the same time is compatible with the RFC standards.



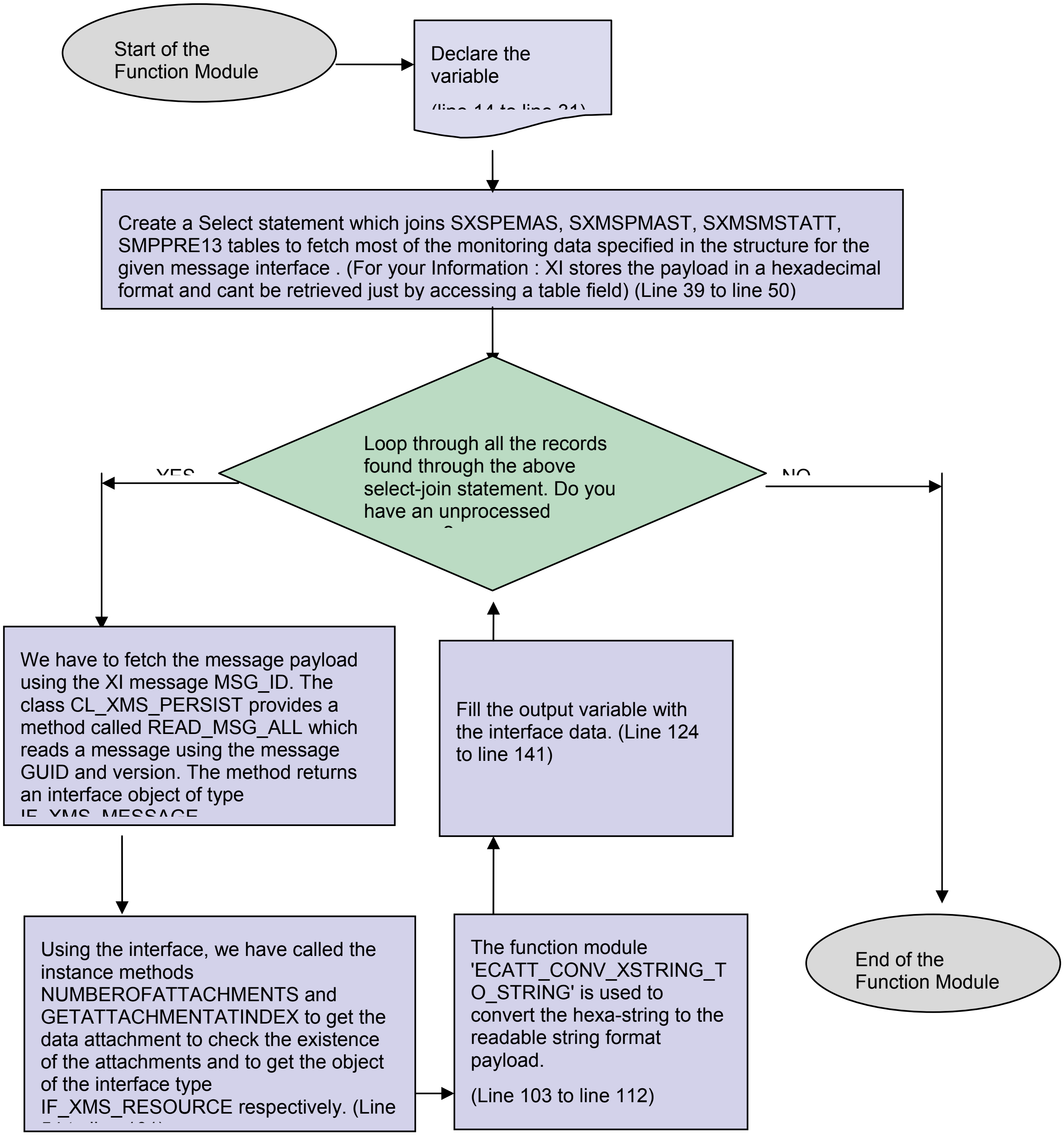
Structure: YSTR_DETAILS Active

Short Description: Structure to hold the XI process data

Component	RTy...	Component type	Data Type	Length	Decim...	Short Description
MSGGUID	☐	CHAR255	CHAR	255	0	Char255
MSGTYPE	☐	CHAR_02	CHAR	2	0	Character length 2
OB_NS	☐	CHAR255	CHAR	255	0	Char255
OB_NAME	☐	CHAR120	CHAR	120	0	char120
OB_SYSTEM	☐	CHAR_LG_60	CHAR	60	0	Character field of length 60
MAPNAME	☐	CHAR255	CHAR	255	0	Char255
IB_NAME	☐	CHAR120	CHAR	120	0	char120
IB_SYSTEM	☐	CHAR_LG_60	CHAR	60	0	Character field of length 60
SENDTIMEST	☐	TIMESTAMPL	DEC	21	7	UTC Time Stamp in Long Form (YYYYMMDDhhmm
INITTIMEST	☐	TIMESTAMPL	DEC	21	7	UTC Time Stamp in Long Form (YYYYMMDDhhmm
EXETIMEST	☐	TIMESTAMPL	DEC	21	7	UTC Time Stamp in Long Form (YYYYMMDDhhmm
ERRTXT	☐	CHAR255	CHAR	255	0	Char255
ADMINUSER	☐	CHAR012	CHAR	12	0	Character field of length 12
MSGSTATE	☐	INT_1	INT1	3	0	Int1
PROCESS_STAT	☐	CHAR255	CHAR	255	0	Char255
MSGTXT	☐	CHAR_LG_60	CHAR	60	0	Character field of length 60
PAYLOAD	☐	SLDSTRING	CHAR	8192	0	SLD Agent: Replacement for type "string" in release

Step 5: Now, without waiting further, we should get into the logic which will fetch the monitoring data for the provided XI message interface from the different tables.

Next page contains a logical flow of the program :


Start of the Function Module
Declare the variable
(line 14 to line 31)
Create a Select statement which joins SXSPEMAS, SXMSPMAST, SXMSMSTATT, SMPPRE13 tables to fetch most of the monitoring data specified in the structure for the given message interface . (For your Information : XI stores the payload in a hexadecimal format and cant be retrieved just by accessing a table field) (Line 39 to line 50)
Loop through all the records found through the above select-join statement. Do you have an unprocessed
YES
NO
We have to fetch the message payload using the XI message MSG_ID. The class CL_XMS_PERSIST provides a method called READ_MSG_ALL which reads a message using the message GUID and version. The method returns an interface object of type IF_XMS_MESSAGE
Fill the output variable with the interface data. (Line 124 to line 141)
Using the interface, we have called the instance methods NUMBEROFATTACHMENTS and GETATTACHMENTATINDEX to get the data attachment to check the existence of the attachments and to get the object of the interface type IF_XMS_RESOURCE respectively. (Line
The function module 'ECATT_CONV_XSTRING_TO_STRING' is used to convert the hexa-string to the readable string format payload.
(Line 103 to line 112)
End of the Function Module

SAMPLE CODE:

```
FUNCTION Y_XI_MONITORING.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(PROCESS_NAME) TYPE  CHAR255 OPTIONAL
*"  EXPORTING
*"     VALUE(HEADER) TYPE  ZHEADER
*"  TABLES
*"      PM_DETAILS TYPE  Y_XI_DETAILS OPTIONAL
*"----------------------------------------------------------------------

******************* Data Declaration section *************************************

    DATA:in_monitor_data  TYPE table of YMONITOR_DATA,
      wa_monitor type YMONITOR_DATA,
      payload type string,
      iv_version TYPE sxmslsqnbr,
      i_count TYPE sxmslsqnbr,
      size TYPE i,
      wa_process_data LIKE LINE OF PM_DETAILS,
      ref_xms_persist TYPE REF TO cl_xms_persist,
      ref_xms_main TYPE REF TO cl_xms_main,
      ex_message TYPE REF TO if_xms_message,
      prop TYPE REF TO if_xms_prop,
      props TYPE REF TO sxms_pro_t,
      resource TYPE REF TO if_xms_resource,
      ex_msgstate TYPE sxmspmstat,
      ex_string TYPE STRING,
      comp_string TYPE string,
      data TYPE ETXML_LINE_STR,
      cx_ref TYPE REF TO cx_root.

 * Selection statement, joins four tables to fetch the interface monitor data ****
 ********** SXSPEMAS, SXMSPMAST, SXMSMSTATT, SMPPRE13 35 **********************


  SELECT      mas~MSGGUID     mas~PID    mast~MSGTYPE    mas~OB_NAME    mas~OB_NS
 mas~IB_NAME    mas~IB_SYSTEM    mas~OB_SYSTEM    mast~SENDTIMEST
 mast~EXETIMEST  mast~INITTIMEST  mast~ADMINUSER    mast~MSGSTATE  statt~MSGTXT
       rel~MAPNAME    mast~vers  INTO CORRESPONDING FIELDS OF TABLE in_monitor_data
 FROM ( ( ( sxmspemas AS mas
 INNER JOIN sxmspmast AS mast ON mast~MSGGUID = mas~MSGGUID )
 INNER JOIN sxmsmstatt AS statt ON mast~MSGSTATE = statt~MSGSTATE
                                              and statt~LANGU = 'E' )
 INNER JOIN smpprel3  AS rel ON rel~FROMACTION = mas~OB_NAME
                                        AND rel~FROMSRVC = mas~OB_SYSTEM  )
 WHERE mas~OB_NAME = PROCESS_NAME.

***************** Loop-1  Starts: (Fetch the Message Payload) ********************

  LOOP AT in_monitor_data INTO wa_monitor.
    CRE7ATE OBJECT ref_xms_persist.
    clear payload.
clear ex_string.
clear comp_string.

IF sy-subrc = 0.
  CLEAR i_count.
  MOVE wa_monitor-vers TO iv_version.
  iv_version = iv_version + '001'.
  DO iv_version TIMES.
   IF ex_string NE comp_string AND comp_string IS NOT INITIAL
                                AND ex_string IS NOT INITIAL.
           CONTINUE.
    ENDIF.
    TRY.
       CALL METHOD ref_xms_persist->read_msg_all
         EXPORTING
           im_msgguid  = wa_monitor-msgguid
           im_pid      = wa_monitor-pid
           im_version  = i_count
         IMPORTING
           ex_message  = ex_message
           ex_msgstate = ex_msgstate.
           CATCH cx_xms_syserr_persist INTO cx_ref.
            EXIT.
       ENDTRY.

      CALL METHOD ex_message->numberofattachments
        RECEIVING
          size = size.

      IF size IS NOT INITIAL.
        CALL METHOD ex_message->getattachmentatindex
          EXPORTING
           index    = '1'
           RECEIVING
            resource = resource.
         TRY.
          CALL METHOD resource->getbinarydata
           RECEIVING
             data = data.
            CATCH cx_xms_exception.
            CATCH cx_xms_system_error.
         ENDTRY.
          IF ex_string IS NOT INITIAL.
           MOVE ex_string TO comp_string.
      ENDIF.

      CALL FUNCTION 'ECATT_CONV_XSTRING_TO_STRING'
        EXPORTING
          im_xstring = data
        IMPORTING
          ex_string  = ex_string.

      IF i_count EQ '000'.
        CONCATENATE ex_string  payload INTO payload.
      ENDIF.

```

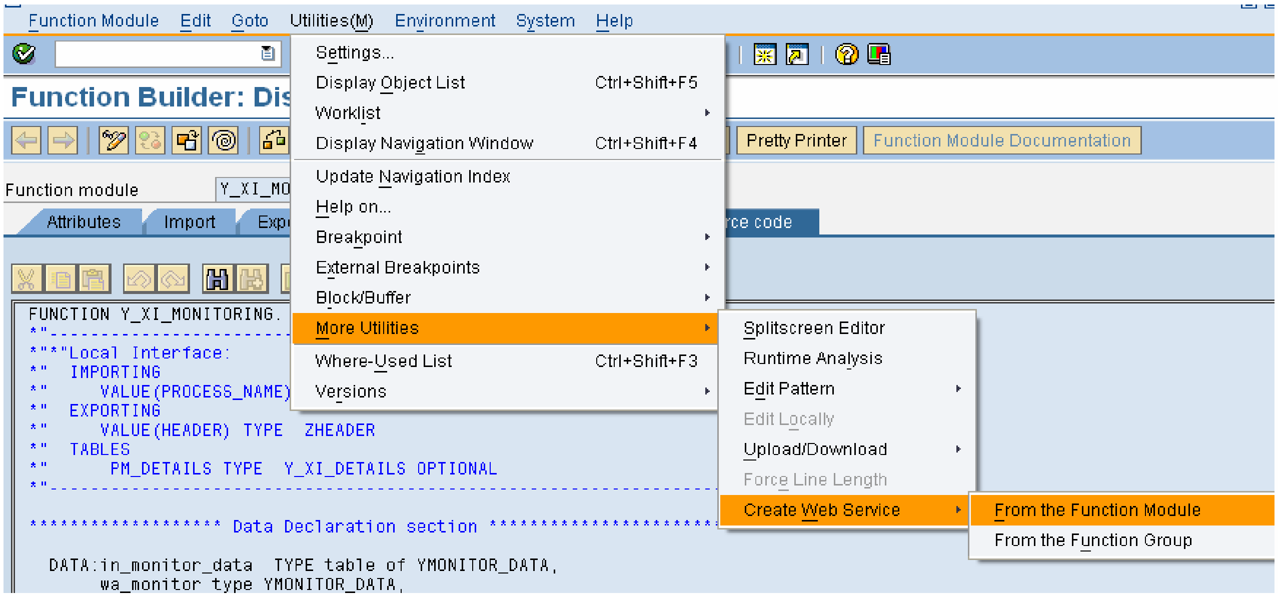
```
         IF ex_string NE comp_string AND comp_string IS NOT INITIAL.
           CONCATENATE ex_string  payload INTO payload.
           CONTINUE.
         ENDIF.
       ENDIF.
       i_count = i_count + 1 .
     ENDDO.
   ENDIF.
  ********************* Filling the detailed data
***********************************
   wa_process_data-MSGGUID    = wa_monitor-msgguid.
   wa_process_data-MSGTYPE    = wa_monitor-MSGTYPE.
   wa_process_data-OB_NAME    = wa_monitor-OB_NAME.
   wa_process_data-OB_NS      = wa_monitor-OB_NS.
   wa_process_data-IB_NAME    = wa_monitor-IB_NAME.
   wa_process_data-MAPNAME    = wa_monitor-MAPNAME.
   wa_process_data-OB_SYSTEM  = wa_monitor-OB_SYSTEM.
   wa_process_data-IB_SYSTEM  = wa_monitor-IB_SYSTEM.
   wa_process_data-SENDTIMEST = wa_monitor-SENDTIMEST.
   wa_process_data-EXETIMEST  = wa_monitor-EXETIMEST .
   wa_process_data-INITTIMEST = wa_monitor-INITTIMEST.
   wa_process_data-ADMINUSER  = wa_monitor-ADMINUSER .
   wa_process_data-MSGSTATE   = wa_monitor-MSGSTATE.
   wa_process_data-MSGTXT     = wa_monitor-MSGTXT.
   wa_process_data-ERRTXT     = wa_monitor-ERRTXT.
   wa_process_data-PAYLOAD    = payload.
   append wa_process_data to PM_DETAILS.
   clear  wa_process_data.
  ********************** Filling ended ******************************************
 endloop.
  *************** Loop-1 Ends : ( Fetch the Message Payload )*******************
ENDFUNCTION.
```

Part III: Exposing the XI Monitoring RFC as a Web Service

So far, we have seen creation of the RFC which fetches the XI monitoring data from the SAP tables.

Now, let's go through the steps for creating a web service for the RFC:

Step 1. Go to the *Utilities (M) → More Utilities → Create Web Service → From the Function Module* which opens the "Wizard: Create Service Definition" screen.

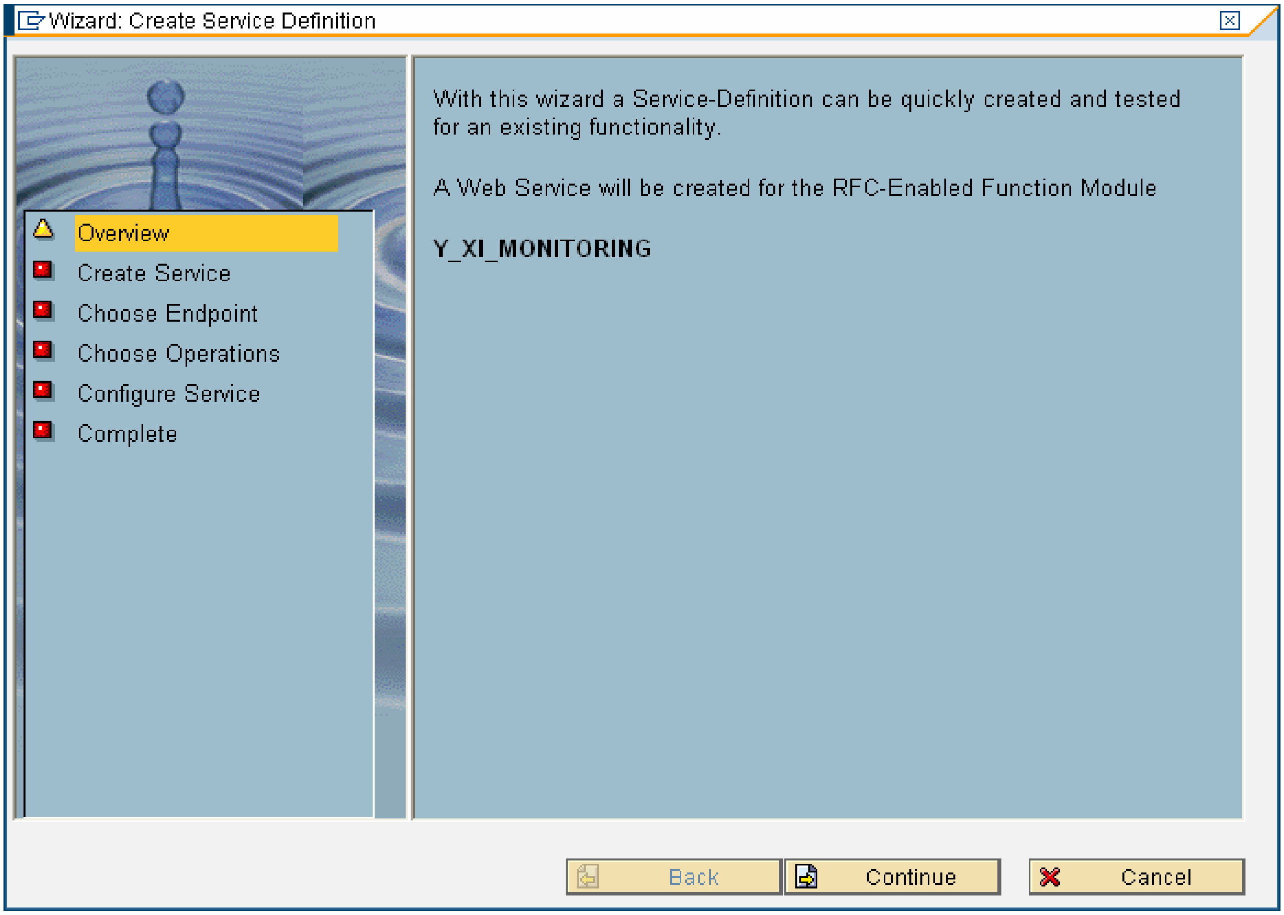


```
FUNCTION Y_XI_MONITORING.
*"-----------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(PROCESS_NAME)
*"  EXPORTING
*"     VALUE(HEADER) TYPE  ZHEADER
*"  TABLES
*"      PM_DETAILS TYPE  Y_XI_DETAILS OPTIONAL
*"----------------------------------------------------------------------

****************** Data Declaration section *********************

  DATA:in_monitor_data  TYPE table of YMONITOR_DATA,
       wa_monitor type YMONITOR_DATA,
```

Step2. Below is the *"Wizard: Create Service Definition"* screen which is the first step of creating the web service. Press "Continue".



Wizard: Create Service Definition

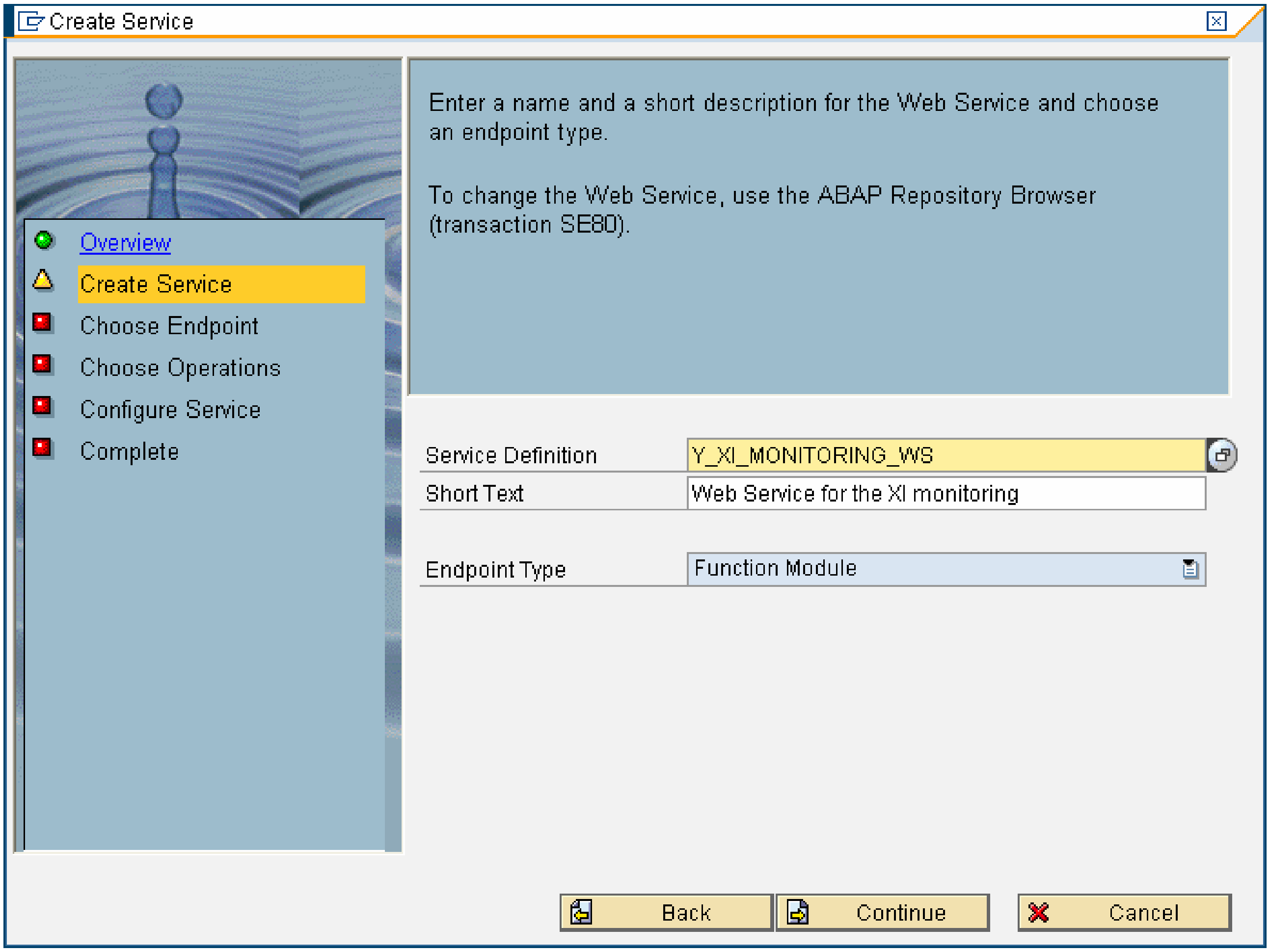
With this wizard a Service-Definition can be quickly created and tested for an existing functionality.

A Web Service will be created for the RFC-Enabled Function Module

Y_XI_MONITORING

Step 3: The below *"Create Service"* screen expects from you a name and a short description for the web service we want to create.

In our case, I have given the name as "Y_XI_MONITORING_WS" for our web service.



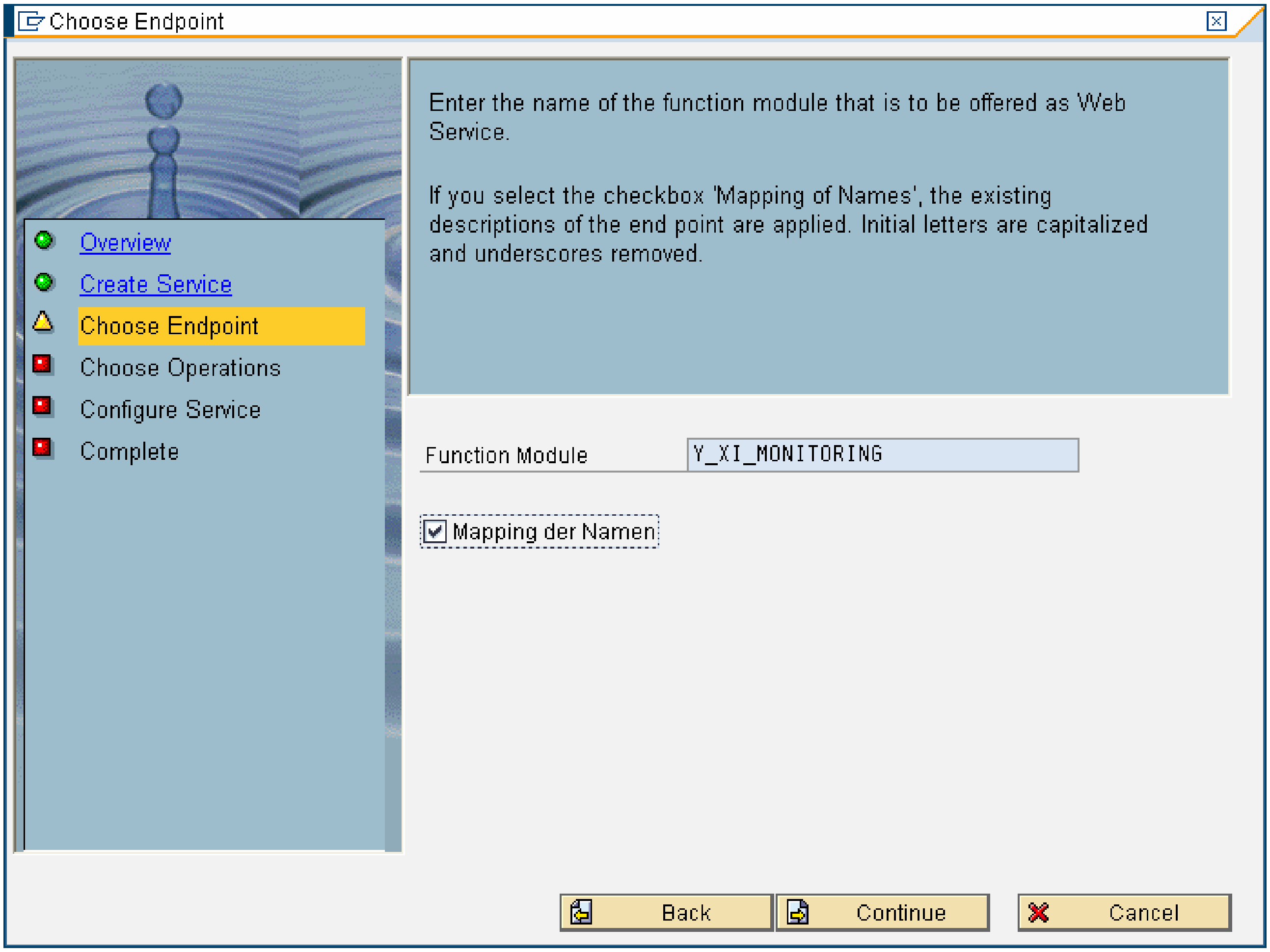
Enter a name and a short description for the Web Service and choose an endpoint type.

To change the Web Service, use the ABAP Repository Browser (transaction SE80).

Service Definition	Y_XI_MONITORING_WS
Short Text	Web Service for the XI monitoring
Endpoint Type	Function Module

Step 4: This step connects your web service name with the RFC which you want to expose as a web service.

In our case, we have provided the RFC (Y_XI_MONITORING) which we have created in the section II.



Choose Endpoint

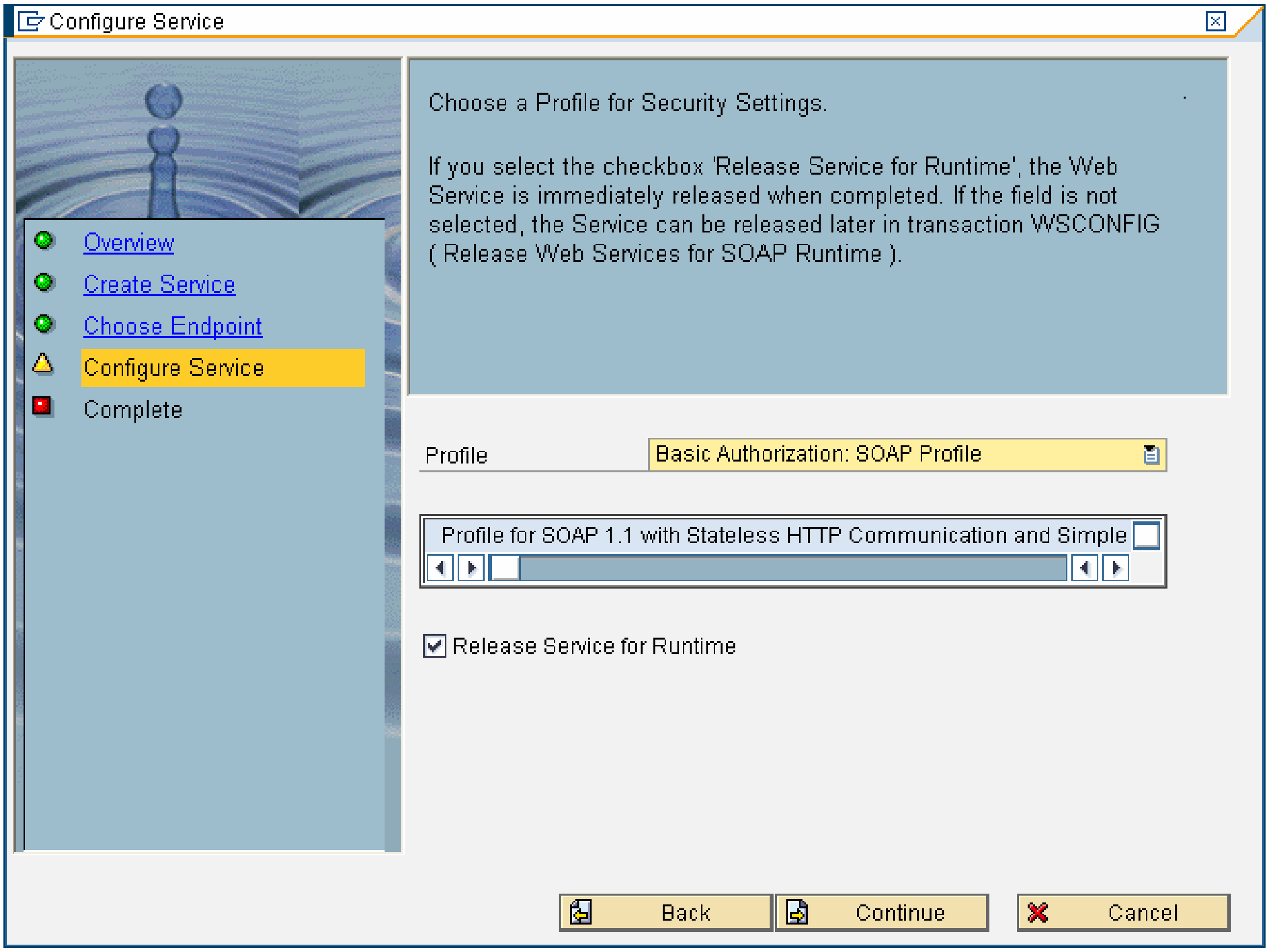
Enter the name of the function module that is to be offered as Web Service.

If you select the checkbox 'Mapping of Names', the existing descriptions of the end point are applied. Initial letters are capitalized and underscores removed.

Function Module Y_XI_MONITORING

☑ Mapping der Namen

Step 5: This step defines the security level you want to put for your web service. In our case, we are selecting the basis authentication; which will allow anyone having access to the XI system to use the service.



Configure Service

Choose a Profile for Security Settings.

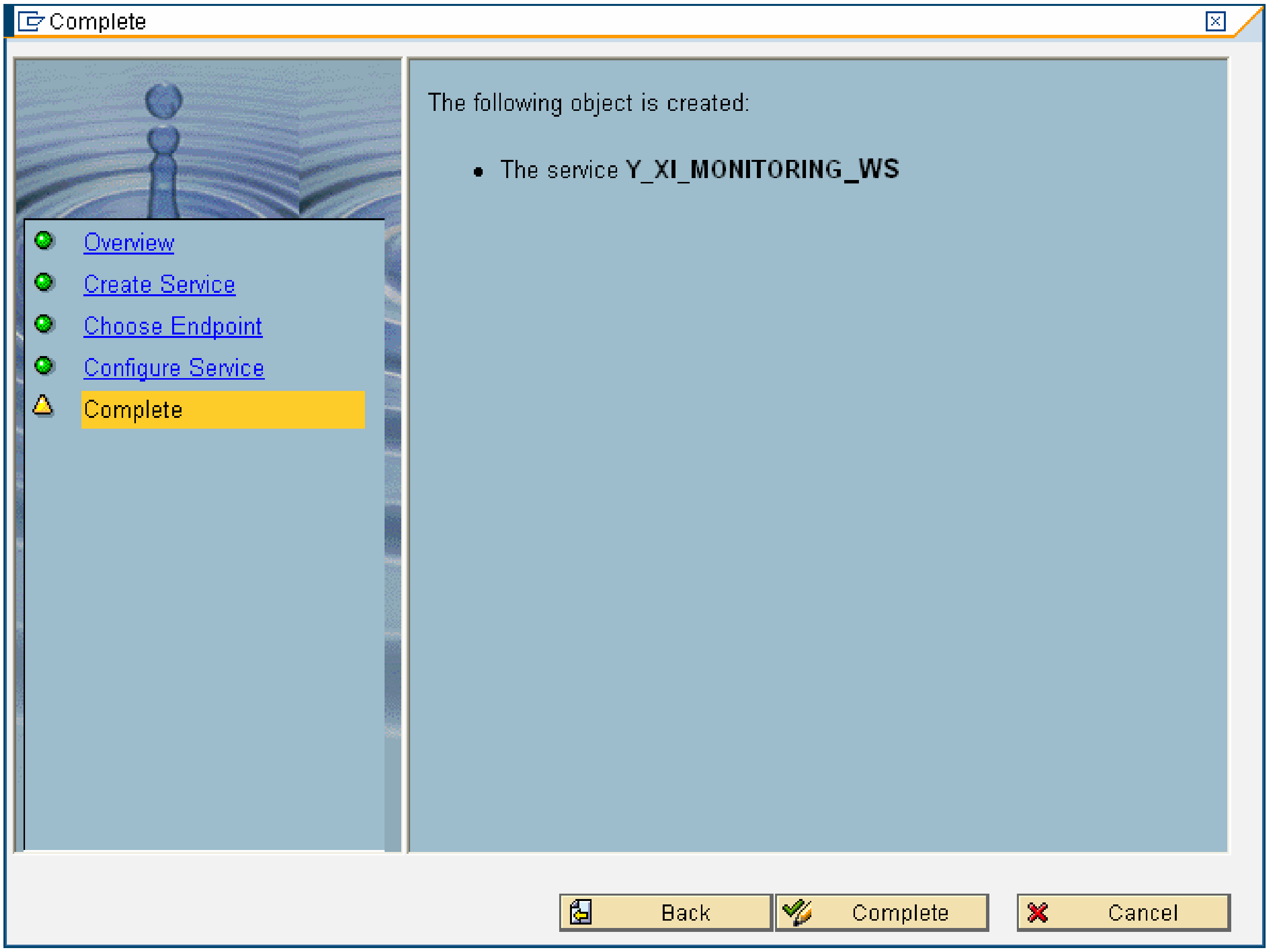
If you select the checkbox 'Release Service for Runtime', the Web Service is immediately released when completed. If the field is not selected, the Service can be released later in transaction WSCONFIG (Release Web Services for SOAP Runtime).

Profile: Basic Authorization: SOAP Profile

Profile for SOAP 1.1 with Stateless HTTP Communication and Simple

☑ Release Service for Runtime

Step 6: This last screen of the Wizard confirms that the information provided for the web service is accurate and consistent. Once you click the *“Complete*” button; it generates a web service.



The following object is created:

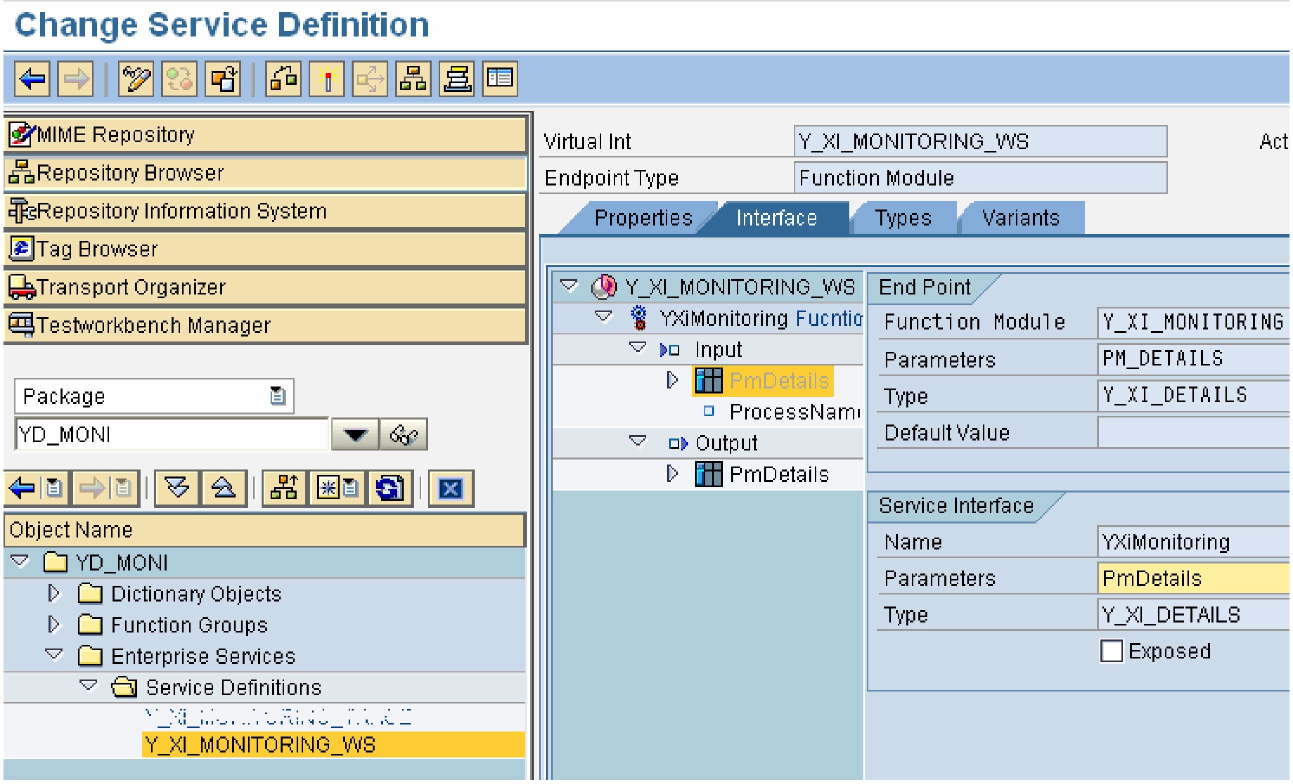
- The service **Y_XI_MONITORING_WS**

Step 7: (Optional) Go to the transaction se80 to check the web service created by the above steps.

Select the package under which the web service was generated. Go to the *Enterprise Services → Service definition* and select the web service created above. In our case it's **Y_XI_MONITORING_WS.**

Go to the *"Interface"* Tab; Here you see "*PmDetails"* under the input node as well as in the output node. But as we know that the table variable (*PmDetails*) is going to carry only the output data, we will chance the setting in such a way that client will only see only the "*ProcessName*" field in the input.

To remove the table (*PmDetails)* from the input parameter list of our web service. Select the input table type as shown below and uncheck the exposed □ (as shown below).



Change Service Definition

Package: YD_MONI

Virtual Int: Y_XI_MONITORING_WS

Endpoint Type: Function Module

End Point

Function Module: Y_XI_MONITORING

Parameters: PM_DETAILS

Type: Y_XI_DETAILS

Default Value:

Service Interface

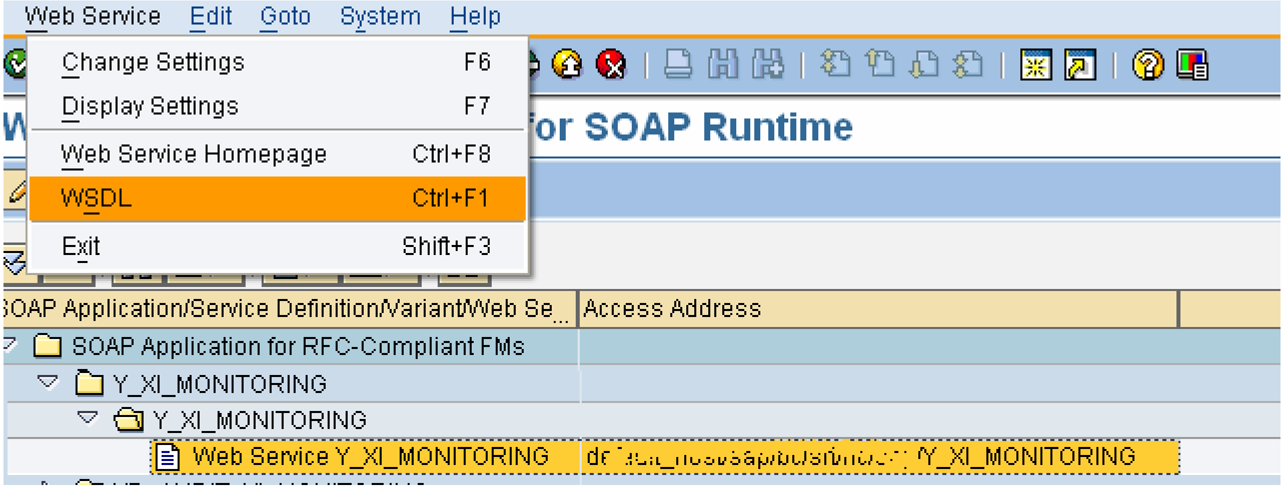
Name: YXiMonitoring

Parameters: PmDetails

Type: Y_XI_DETAILS

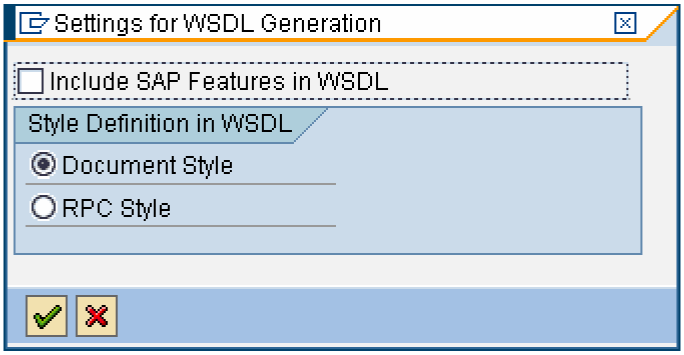
□ Exposed

Step 7: To create a WSDL for the web service, go to the transaction WSADMIN. Select the web service definition created above and then go to Web Service→ WSDL.



SOAP Application/Service Definition/Variant/Web Se...	Access Address
SOAP Application for RFC-Compliant FMs	
Y_XI_MONITORING	
Y_XI_MONITORING	
Web Service Y_XI_MONITORING	[illegible]/Y_XI_MONITORING

Step 8: The below popup screen "*Settings for WSDL Generation*" appears. Select the WSDL Style as "*Document Style*".


Settings for WSDL Generation

☐ Include SAP Features in WSDL

Style Definition in WSDL

● Document Style

○ RPC Style

Step 9: The wsdl will be generated and opens in the default web browser.

Usage

The Above created web service can be consumed by any web client. For example, you can create a web application using Java or Visual Basis and from that application you can call the XI monitoring web service to fetch the testing statues for the XI interfaces. You can also program your iPhones and BlackBerry to call the XI web service to find the interface status.

Have a look into the below document to see an example of consuming the web service in ABAP
https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/6066fbe8-edc4-2910-9584-a9601649747d

Note of Thanks

I would like to thank Anish Abraham for his help in creating the RFC.

Related Content

- http://help.sap.com/saphelp_46c/helpdata/en/22/0424fe488911d189490000e829fbbd/frameset.htm
- http://help.sap.com/printdocu/core/Print46c/en/data/pdf/BCFESDE2/BCFESDE2.pdf
- https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8032

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.