Service Enabling with SAP NetWeaver Process Integration 7.1

Applies to:

SAP NetWeaver Process Integration 7.1 EHP1, Web Service Development

For more information, visit the SOA Management homepage.

Summary

This article is intended to familiarize ABAP Web service developers with the procedure for building and configuring a standards-compliant Web service using SAP NetWeaver Process Integration 7.1 (Ehp1)

To illustrate the procedure, we show example objects from the SAP NetWeaver Demo Model.
You can find out more about the SAP NetWeaver Demo Model in the SAP Developer Network here:

https://www.sdn.sap.com/irj/sdn/nw-demomodel

Author: Paul Read

Company: SAP AG

Created on: 12 March 2009

Author Bio

Paul Read works in SAP NetWeaver Product Management, at Walldorf, Germany.

Table of Contents

Service Enabling with SAP NetWeaver Process Integration 7.1

Service enabling is the process of making business functionality available as executable standards-compliant services. Once a service is configured, it can be called (consumed) by an application. Additionally, services can be published to a services registry.

This article covers the complete service enabling process, which spans from designing objects to implementing, publishing, and consuming the resulting Web service. In order to keep this article as brief as possible, we have reduced the steps described to a minimum, and we have restricted ourselves to only the design objects needed to create a **synchronous, stateless** service interface, implemented in ABAP, and configured using **SOA Manager**.

Designing a Service

You first need to design the objects that you need to build a simple Web service.

The following objects are essential to create a working service.

Data type

Data types define the structures and sub-structures to be used in the messages exchanged at runtime.

In this article, we use the following data types: BusinessPartnerByIDQuery and BusinessPartnerByIDResponse

Message type

Message types define the messages to be exchanged at runtime between a service provider and a consumer. For example, a message type can be an error message, a warning, an information message, or a success message.

Message types are used to build a service interface. The same message type can be reused in multiple service interfaces.

For synchronous service interfaces, you will normally need to design at least two message types: a request message type and a response message type. Optionally, you can also design a fault message type. (For an asynchronous operation, you would normally design only a request message type.)

A message type is based on a data type.

In this article, we use the request message type BusinessPartnerByIDQuery and the response message type BusinessPartnerByIDReponse.

Service interface

A service interface groups entities that belong to a service. One or more operations can be assigned to a service interface.

The service provider can supply a single interface for any number of operations. A consumer can select any operation from the same service interface.

For each operation, you can define one or more parameters. An operation can comprise one or more message types.

In this article, we use the following service interface: BusinessPartnerByIDQueryResponseIn

More Information:

Enterprise Services Repository

Providing, Discovering, and Consuming Services

Designing a Data Type

To design the objects that are needed to build a service, you use the Enterprise Services Repository (ES Repository) in a back-end system.

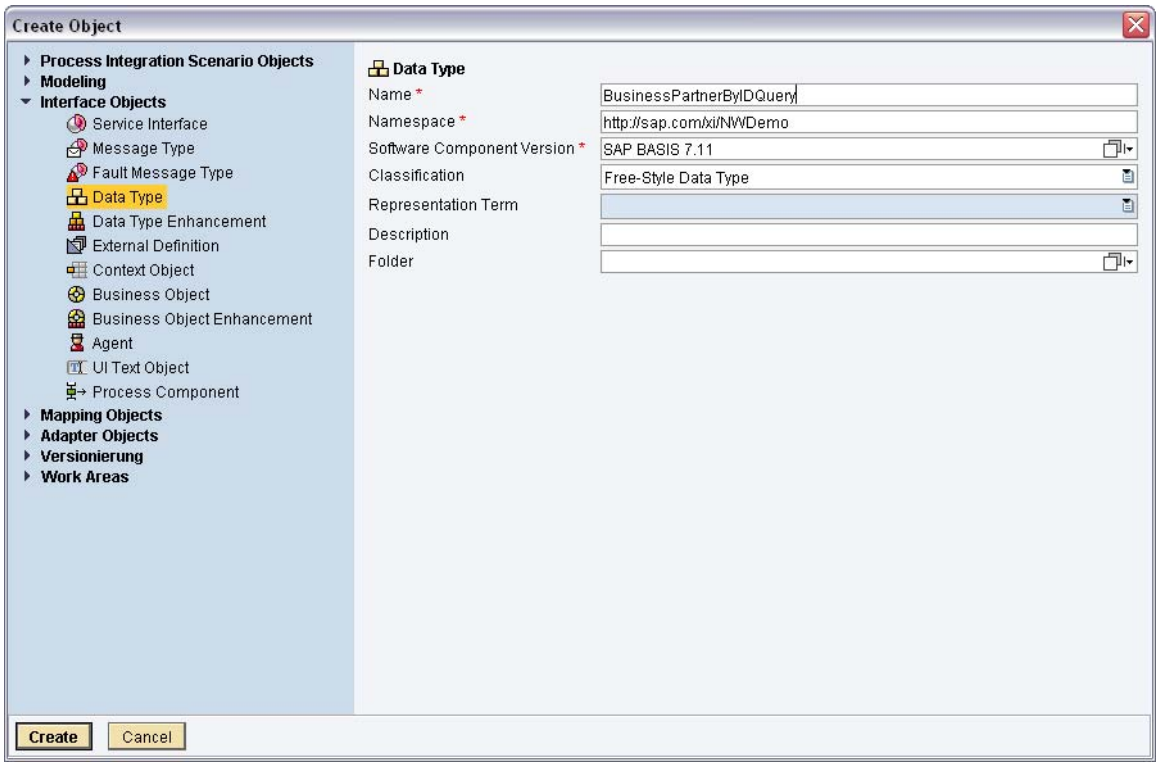
You follow essentially the same procedure for each object type.

1. To access the Enterprise Services Repository, use the Enterprise Services Builder (transaction code `SXMB_IFR`).
2. Choose *Enterprise Services Builder* and log on.
3. In the *Design Objects* tab, select a software component and version.
4. Within a software component, select and expand a namespace.

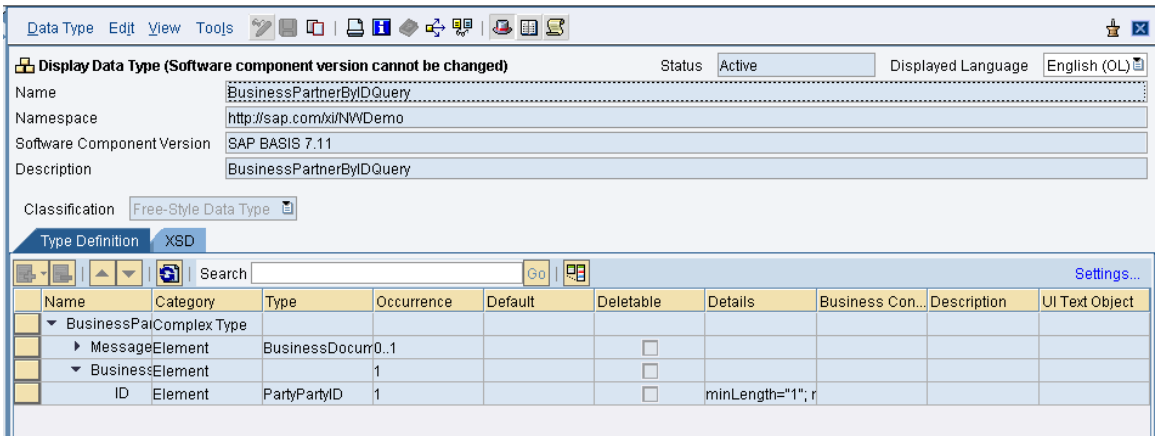
   Now you can design a data type, a message type, and a service interface.
5. Open the context menu, then choose *New*.
6. In the dialog box, open the *Interface Objects* section and select *Data Type*.
7. Specify a name for the data type.

   Here, we use the BusinessPartnerByIDQuery and BusinessPartnerByIDResponse data types.
8. Select a classification.

   Select the Free-Style classification for the BusinessPartnerByIDQuery data type.



9. *Create*.



10. *Activate*.

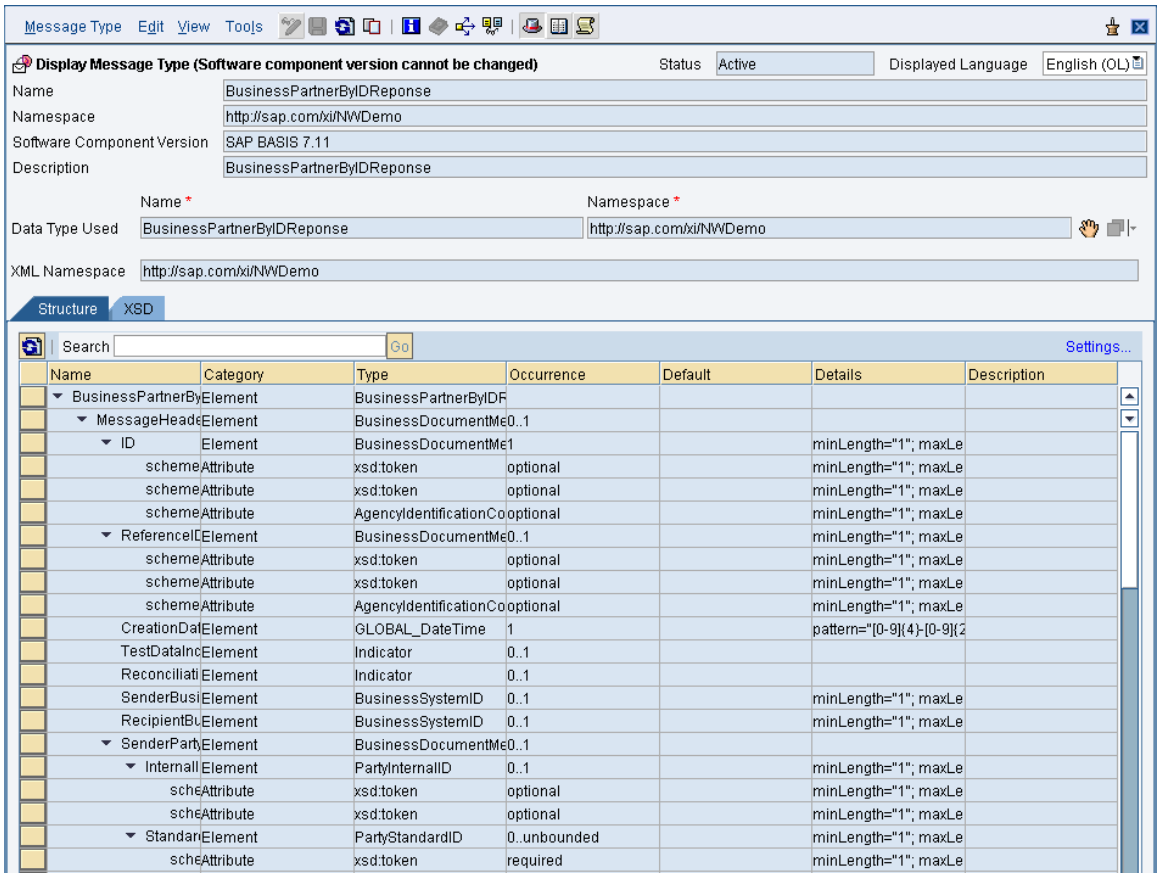
    Once it has been saved in the Enterprise Services Repository, a data type needs to be activated so that it can be used for proxy generation later on.

Designing a Message Type

Message types are designed in the Enterprise Services Repository.

1. In the Enterprise Services Repository, open the context menu, then choose *New*.
2. Open the *Interface Objects* section and select *Message Type*.
3. Specify a name, for the message type.

   In this example, we use the request message type BusinessPartnerByIDQuery and the response message type BusinessPartnerByIDReponse.



4. *Create*.
5. In the editor, associate a message type with a data type.

   In our example, the data type BusinessPartnerByIDQuery is associated with the message type BusinessPartnerByIDQuery. The BusinessPartnerByIDResponse data type is associated with the message type BusinessPartnerByIDQuery.

6. Save the message type.
7. Activate the message type.

   Once saved in the Enterprise Services Repository, the message types need to be activated so that they can be used for proxy generation.

Designing a Service Interface and its Operations

A service interface is normally first designed on the provider side. This service interface is known as *inbound*. A consumer can use the definition of an inbound service interface to define its own *outbound* service interface. The inbound and outbound service interfaces must be identical in order to be used for *point-to-point* communication scenarios.. The Enterprise Services Repository ensures that the service interfaces are identical.

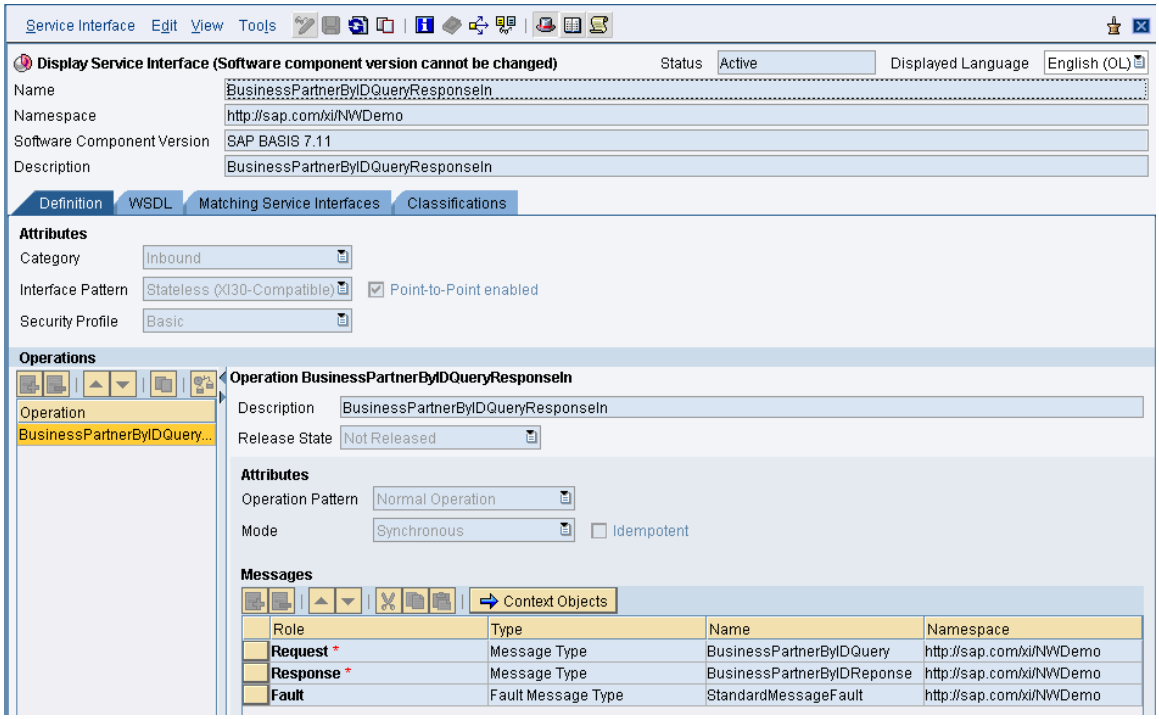
A consumer service interface can also be defined without reference to a provider service interface. This is also supported by the Enterprise Services Repository. Here, however, point-to-point communication is not possible as the structures of the inbound and outbound services interfaces are not identical. *Brokering functionality* is needed, and this is provided by Process Integration.

1. From the Enterprise Services Builder, open the context menu, then choose *New*.
2. From the dialog box, open the *Interface Objects* section and select *Service Interface*.
3. Specify a name for the service interface.
4. *Create*.
5. In the editor, set the options for the service interface.

   For a *provider service interface*, set *inbound*.

   Note

Abstract:	The service interface has no implementation in an application system.
Synchronous:	The consumer is blocked until the service call is returned.
Asynchronous:	The consumer is not blocked until a response is returned.
Stateless:	No state is retained on the provider system between a service's operations calls.
Stateless (XI30–compatible):	Only one operation
Stateful:	The session state can be retained in memory on the service provider side across multiple service operation calls.
TU & C/C (Tentative Update & Confirm / Compensate):	For synchronous communication between applications where changes in the persistent state are required on the provider side. More information: Interface-Pattern and Consuming TU & C/C Web Services.



6. Assign the message types.

   Use the request message type BusinessPartnerByIDQuery and the response message type BusinessPartnerByIDReponse.

   In our example, a standard fault message type is also assigned.

7. Save the service interface.

8. Activate.

   Once saved in the Enterprise Services Repository, the service interface needs to be activated so that it can be used for proxy generation.

   The modeling (design) process in the Enterprise Services Repository is now complete. You have created a model for a service interface, that is, the objects needed to build an executable enterprise service. The service interface will subsequently be used to generate a provider proxy in the ABAP back-end system.

More Information

Developing Service Interfaces

Implementing a Service

When a service has been modeled in the Enterprise Services Repository, you can implement it in a provider (back-end) system.

In the ABAP environment, you implement a service as follows:

1. Generate a provider proxy for the service interface.
2. Add your own source code to the provider proxy.
3. Create a runtime configuration for the service.

These tasks are outlined in the following sections.

Generating a Provider Proxy for the Service Interface

When a proxy is generated, a service definition is generated based on the entities modeled in the Enterprise Services Repository. A service definition contains programming language-specific constructs, such as an interface with methods and parameters.

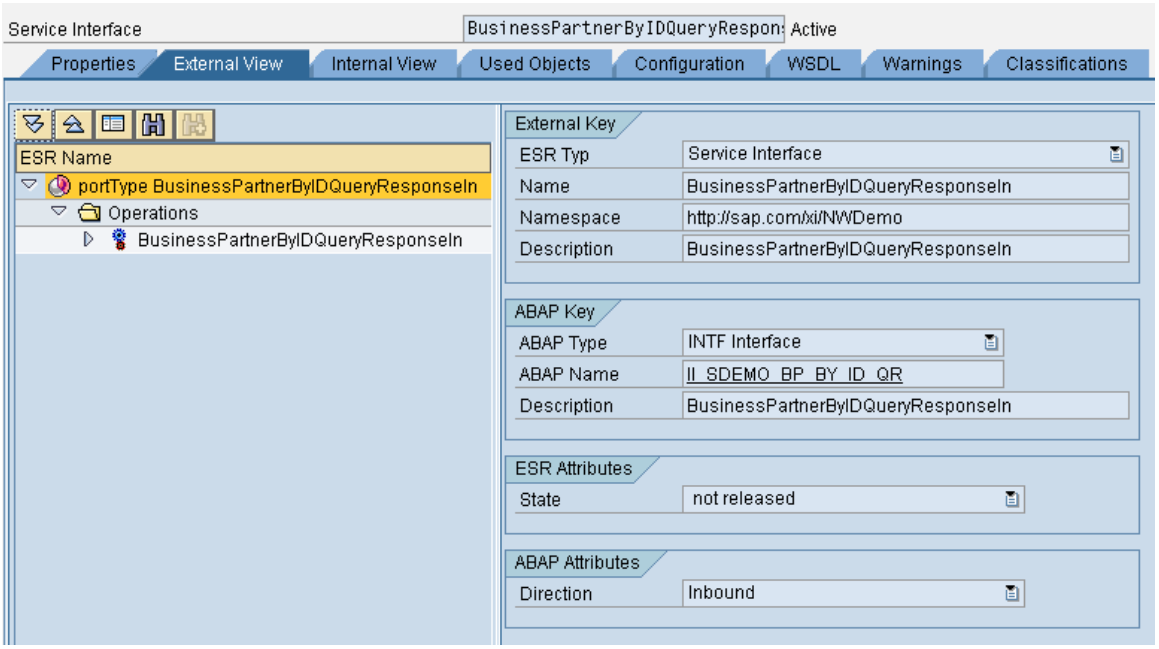
Before a provider (inbound) proxy is generated, you first need to decide which provider (back-end) system to generate it in.

Perform the following steps in the back-end system:

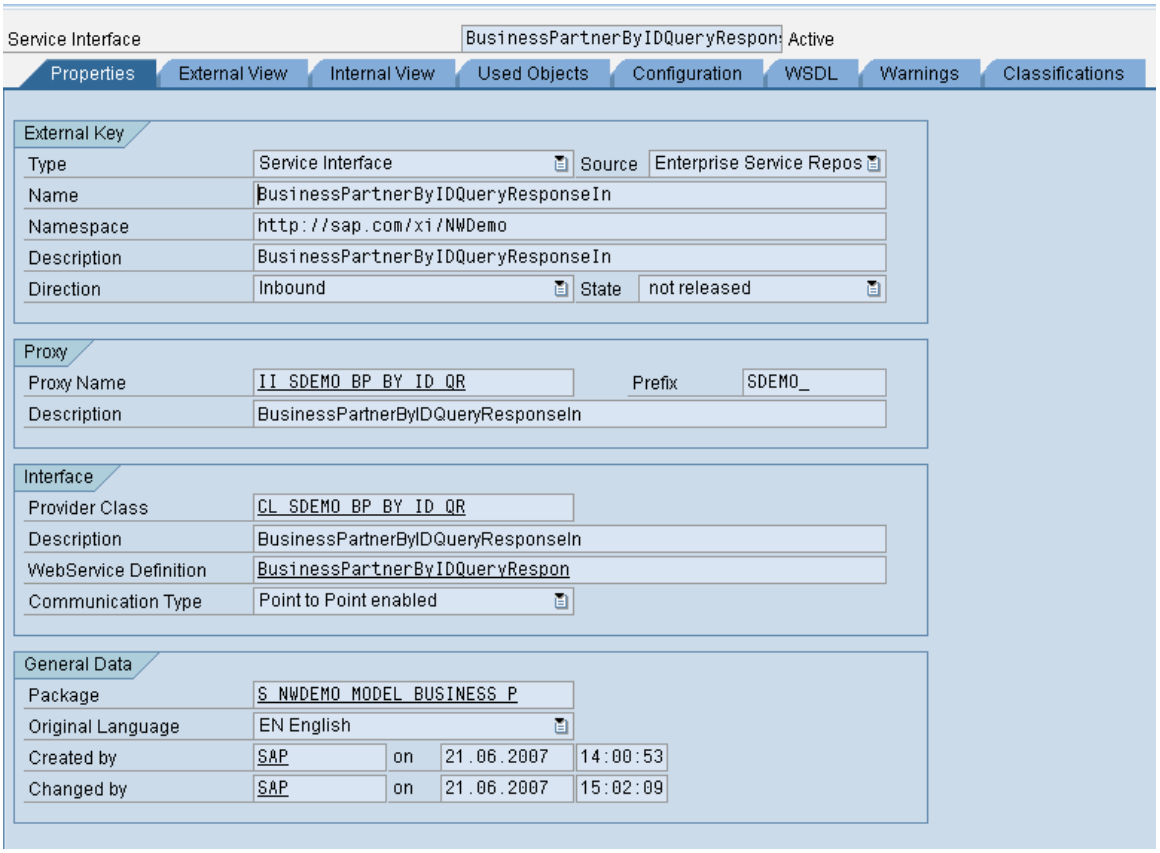
1. Start the *Enterprise Services Browser* (Use transaction code SPR0XY).
2. Expand the nodes of the software component version and the namespace, and select the service interface.
3. Open the context menu and choose *Create proxy*.

   A wizard guides you through the steps to create a proxy. You need to specify a package, a prefix for the repository object names, and a transport request.

   When the proxy has been generated, check the information displayed in the *External view* and *Internal view* tabs. Here, you can see the operations and methods, and the corresponding ABAP message types and data types.



4. Save and activate the provider proxy.



When the provider proxy is activated, all the related data types and message types are activated as well. A service definition is also activated. A configured service definition is needed to be able to use the generated service interface as a Web service.

The next step is to add the service functionality.

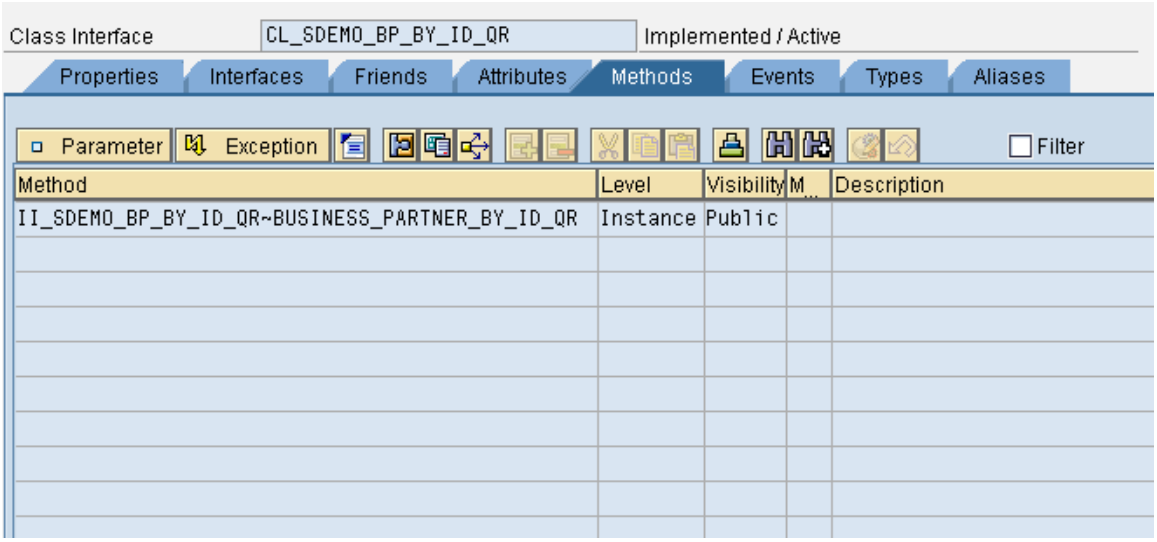
Implementing the Proxy Source Code

The provider proxy consists of an ABAP proxy interface and an implementing class that uses the proxy interface. The implementing class contains the operations designed in the *Enterprise Services Repository* in the form of methods. To implement the functionality of a service operation, the methods need to be filled with application code.

In the *Enterprise Services Browser* (transaction code SPR0XY), perform the following steps:

1. Locate the generated provider proxy and go to the `Properties` tab.
2. Double-click the provider class.

   The methods that correspond to the operation are displayed in the *Class Builder*.



3. Double-click the method that you want to implement.

   In this example, there is only one method: II_SDEMO_BP_BY_ID_QR~BUSINESS_PARTNER_BY_ID_QR.

4. Add the code – the implementation of the proxy method.

Method II_SDEMO_BP_BY_ID_QR~BUSINESS_PARTNER_BY_ID_QR Active

```
METHOD ii_sdemo_bp_by_id_qr~business_partner_by_id_qr.
*** **** INSERT IMPLEMENTATION HERE **** ***
* DATA DECLARATION
**********************************************************************
  DATA: l_partner_id TYPE sdemo_bp-partner_id.

  DATA: lr_partner       TYPE REF TO cl_nwdemo_service_bp.
  DATA: lr_nwdemo_bp_err TYPE REF TO cx_nwdemo_bp.

* PROGRAM
**********************************************************************
  CREATE OBJECT lr_partner.

  l_partner_id =
     input-business_partner_by_idquery-business_partner_sel_by_id-id.
  IF l_partner_id IS INITIAL.
    RAISE EXCEPTION TYPE cx_sdemo_standard_msg_fault
      EXPORTING
        textid = cx_nwdemo_bp=>co_partnerid_initial.
  ENDIF.

  TRY.
      CALL METHOD lr_partner->read_bp
        EXPORTING
          i_partner_id    = l_partner_id
        IMPORTING
          es_proxy_output = output.
    CATCH cx_nwdemo_bp INTO lr_nwdemo_bp_err.
      RAISE EXCEPTION TYPE cx_sdemo_standard_msg_fault
        EXPORTING
          textid = lr_nwdemo_bp_err->textid.
  ENDTRY.
ENDMETHOD.
```

Next, you need to configure the service interface.

Configuring a Service Provider

To be able to work with a service, a runtime configuration is needed. A runtime configuration is created using the service definition that was automatically created during proxy generation.

You configure a service by creating a *service endpoint* for it, or by editing existing endpoints. An endpoint contains a single runtime configuration for an enterprise service. To define different runtime behaviors, you can create multiple endpoints for the same service. This allows you to provide the same enterprise service with a different runtime behavior to different consumers.

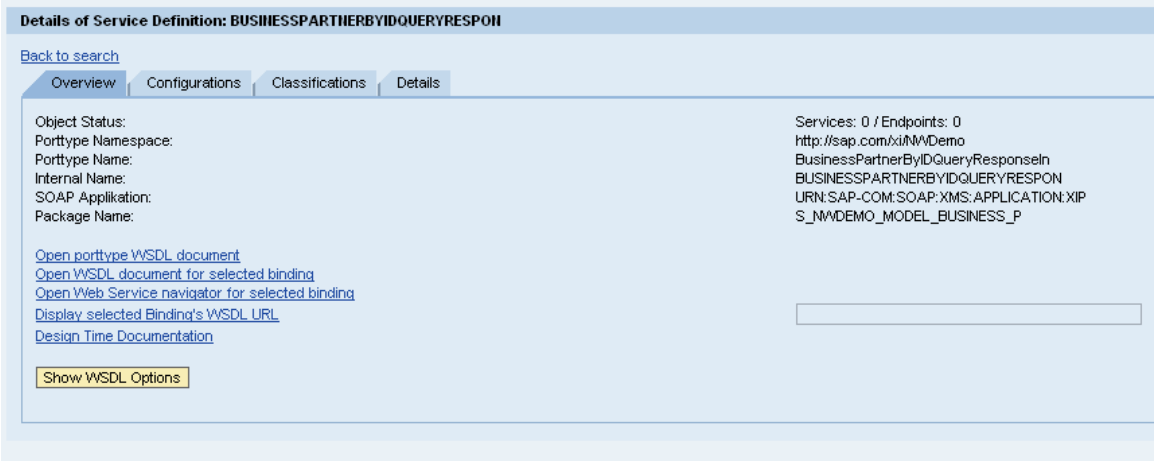
Note

If you do not create an endpoint, it will not be possible to call the enterprise service.

To configure a service provider, perform the following steps:

1. Start the SOA Manager (transaction code SOAMANAGER).
2. From the main screen of the SOA Manager, go to the *Application and Scenario Communication* tab.
3. Select *Single Service Administration*.
4. Specify a service and choose Go.

   In this example, we are using BusinessPartnerByIDQueryResponseIn.
5. Select the service and choose *Apply Selection* to display more information about the service definition.



   The information includes the namespace for the port type, the package name, and links to the WSDL documents for the endpoint(s).
6. Go to the *Configurations* tab.
7. Choose *Create Endpoint*.

   A dialog box is displayed.
8. Specify a unique name for the endpoint.

   Here, we are using: BusinessPartnerByIDQueryResponseIn
9. Choose *Apply Settings*.

   The configuration settings for the endpoint are displayed.
10. Check that the configuration settings meet your requirements.

    To make changes, choose *Edit*.

    For *Authentication Settings*, set a user and password with which the consumer and the provider will authenticate themselves to each other.

    Leave the remaining settings at their default values.
11. Save.

    The service can now be called and the WSDL document for the service is available.

    Make a note of the URL of the WSDL document for the service interface. You will need this URL later when you come to configure a consumer proxy. You can find out the URL from the Overview tab.

Publishing a Service

Services can be published to the UDDI-compliant Services Registry. The Services Registry stores service descriptions and other technical details that are needed to call a service.

In the Services Registry, taxonomies are used to classify services. Service consumers and users can then locate services using the classification criteria.

We assume here that the connection to the Services Registry is already configured.

To publish a service, you need to perform the following tasks:

Publishing a Service to the Services Registry

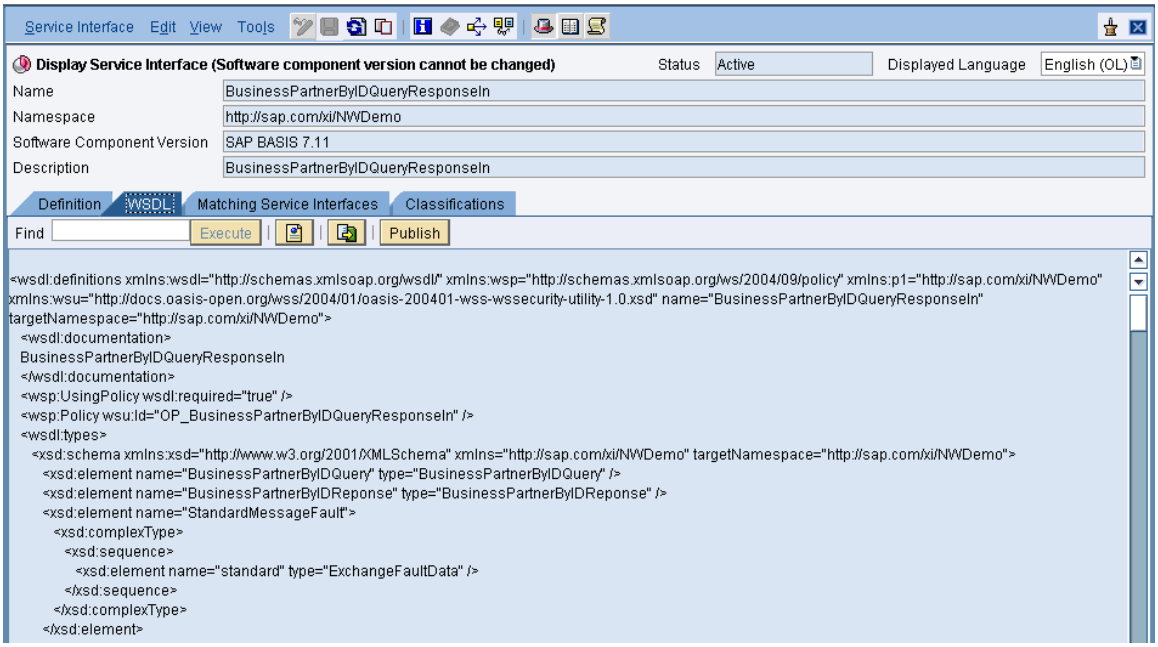
You can publish services directly from the Enterprise Services Repository.

1. To start the Enterprise Services Repository, use transaction code SXMB_IFR.
2. Choose *Enterprise Services Builder* and log on when you are prompted.
3. In the *Enterprise Services Builder*, open the inbound Service Interface that you want to publish.

   We are using the service interface: BusinessPartnerByIDQueryResponseIn
4. Go to the *WSDL* tab.

   The WSDL document for the service interface is generated and displayed.
5. Choose *Publish*.

   The system creates information about the physical system, the UDDI key, and the URL for the WSDL document in the Services Registry.



Checking That the Service is Available

1. Connect to the Services Registry in which you want to publish the service.

   Use transaction code SXMB_IFR and choose Services Registry.

   Your system administrator will be able to provide the information and authorizations you need to access the registry.

2. Search for BusinessPartnerByIDQueryResponseIn to confirm that it is now published and available in the registry.

   You have now created and published an executable service.

Consuming a Service

To be able to create and configure a service consumer, you need to generate and configure a *consumer proxy*.

Consuming a service involves three main tasks:

1. Generate the consumer proxy.
2. Configure the consumer proxy.
3. Implement the consumer application.

These tasks are described in the following sections.

Generating a Consumer Proxy

A consumer proxy is used to implement the enterprise service consumer.

There are two different ways to generate a consumer proxy.

Method 1

You can generate a consumer proxy from the outbound service interface in the Enterprise Services Repository.

1. Start the *Enterprise Services Browser* (transaction code SPR0XY).
2. Locate the outbound service interface BusinessPartnerByIDQueryResponseIn.
3. Open the context menu and choose *Create proxy*.

   A wizard prompts you through the steps to create a proxy. You need to specify a package, a prefix for the repository object names, and a transport request.

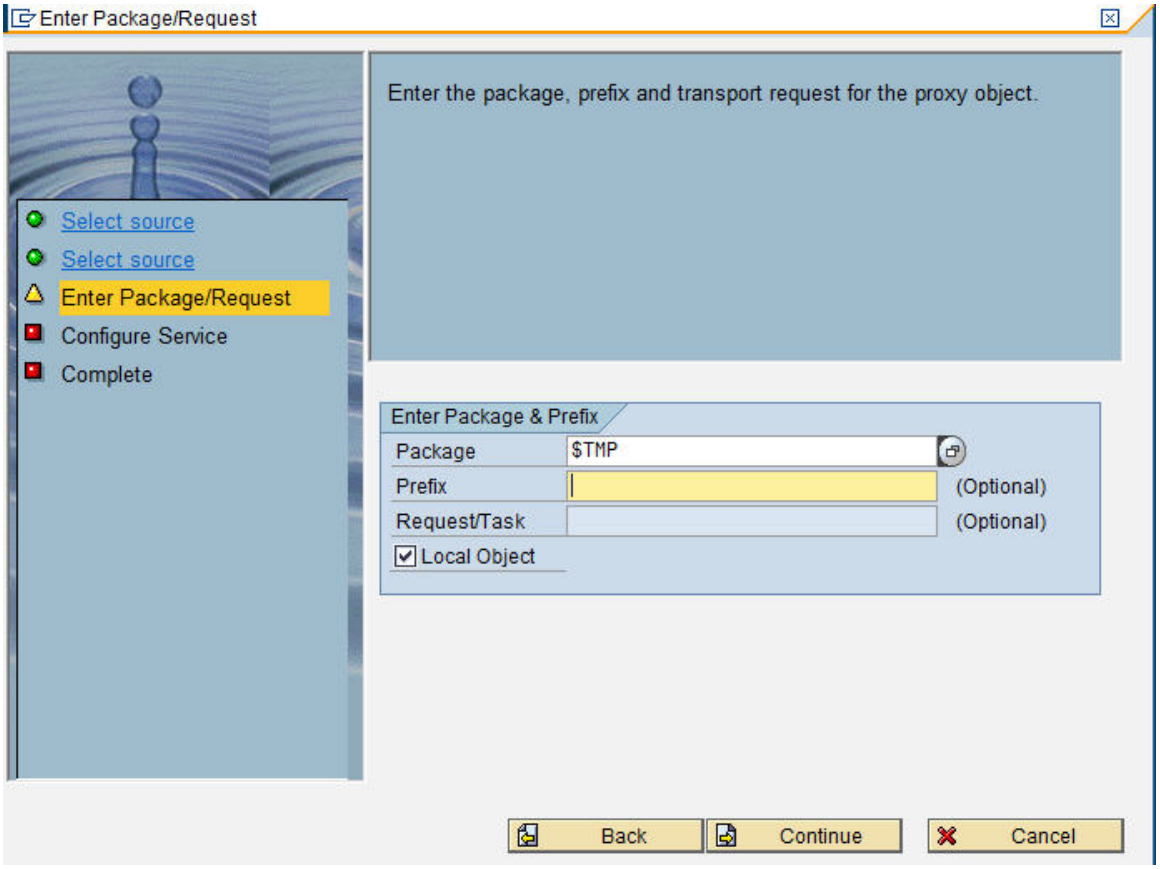
Method 2

Note

You need to know the URL of the WSDL document for the provider proxy. See *Configuring a Service Provider* above to find out how to locate the WSDL in the SOA Manager.

1. Start the Object Navigator (transaction code SE80).
2. In the Repository Browser, right-click over the service provider to open the context menu.
3. Choose *Create* to start the wizard.
4. Select *Service Consumer* and *Continue*.
5. Select *URL/HTTP destination* and *Continue*.
6. Specify the URL of the WSDL document for the provider proxy.

   You may be prompted to supply a host name, a port number, and your user. Select “Local Objects” since you are not transporting objects here.



7. Save and activate the consumer proxy.

Configuring a Consumer Proxy

To configure a service consumer to access a particular service, you need to create and configure one or more *logical ports* for the consumer proxy.

A logical port references a service endpoint, which is available at a unique location on a provider system. For this reason, a logical port is created based on the requirements of the provider.

You can create a logical port based on the WSDL document for a service. The WSDL document describes how to access the endpoint, and includes the URLs for all the endpoints defined for a service.

When the consumer proxy is generated and configured, you need to create an executable program to consume the service.

Creating a Logical Port

1. From the main screen of the SOA Manager (transaction code SOAMANAGER), go to the *Application and Scenario Communication* tab.
2. Select *Single Service Administration*.
3. Locate the *consumer proxy* BusinessPartnerByIDQueryResponseIn.
4. Select the consumer proxy and choose *Apply Selection* to display design-time information about it.
5. Go to the *Configurations* tab.
6. Choose *Create Logical Port*.
7. In the dialog box, specify a name for the new logical port (LP_BusinessPartner).

   To make this logical port the default logical port, select *Logical Port is Default*. If a consumer application is not configured to call a specific logical port, it calls the default logical port.

8. Select the configuration type.

   WSDL-Based Configuration: Specify the WSDL access settings: *Via HTTP access.*

   Here, you specify the path to the WSDL document and a user and password.

9. Choose *Apply Settings*.

   The default settings for the logical port are displayed.

10. Check the configuration.

    Configuring a Consumer Proxy

11. Save.

    The consumer proxy is now generated and configured.

Implementing a Simple Consumer Application

To implement a consumer application in ABAP, you need to create a program in which the consumer can use the client proxy program to execute the service.

1. Use the *ABAP Editor* (transaction code SE38) to create a simple application.
2. In the consumer application start transaction SE80.
3. Open the $TMP context menu.
4. Right-click on the Programs folder, and choose Create -> Program. Specify a program name.
5. Deselect “With TOP INCL”, then save.
6. Select the name of the proxy on the left-hand side and drag it into the empty program.

   Now you can adapt the coding in the program:

7. Instantiate the generated proxy class.

```
DATA: lv_<client proxy name> TYPE REF TO <client proxy name>.
    CREATE OBJECT lv_<client proxy name>
      EXPORTING
          LOGICAL_PORT_NAME = ...
```

   The logical port must be specified if no standard port has been agreed upon.

   Specify the logical port name BusinessPartnerByIDQueryResponseOut. If this logical port is the default, you do not need to specify a logical port.

8. Fill in the query data structure.

```
DATA: lv_<request data> TYPE <request data>.
      lv_<response data> TYPE <response data>.
```

9. To call the service, execute an instance method to call the enterprise service operation

```
CALL METHOD lv_<client proxy name>-><name of method / operation>
    EXPORTING
      INPUT = lv_<request data>
    IMPORTING
      OUTPUT = lv_<response data>
```

   The runtime supports protocols for the purpose of more specialist services.

10. Click on “Test -> Direct Processing” to test your program.

    Consuming Web Services Using Point-to-Point Communication

Related Content

Enterprise Services Repository & Registry

New Capabilities in SAP NetWeaver Process Integration 7.1 - Podcast

SAP NetWeaver Process Integration 7.1 Started Ramp-up - Presentation

Copyright