EDM EMM Scenario Part I: Service Enabling - How to Implement eSOA-Compliant Services

Applies to:

MySAP ERP based upon SAP NetWeaver 2004 or higher, XI 3.0.

Summary

This tutorial is part I of a series that describes core concept of composite applications and how to build them with NW04s by way of the EDM demo scenario, as demonstrated previously by the SAP Platform Ecosystem Market Development Engineering team.

The document focuses on service-enabling in a MySAP ERP backend including the service definition in the XI Integration Repository from a technical point of view.

Author: Mark Mauerwerk

Company: Axentiv AG

Created on: 24 August 2006

Author Bio

Mark Mauerwerk is currently working as Solution Architect of SAP Platform Ecosystem primarily focused on developing early stage demos and technical due diligence of SAP's newest technologies. He is an expert in SAP NetWeaver architecture (ABAP and JAVA) and senior consultant at axentiv AG, a German SAP service partner and ISV enabling partner. He looks back on many years of experience in development and consultancy both internally at SAP as well as for customers & partners in the field.

Table of Contents

Introduction

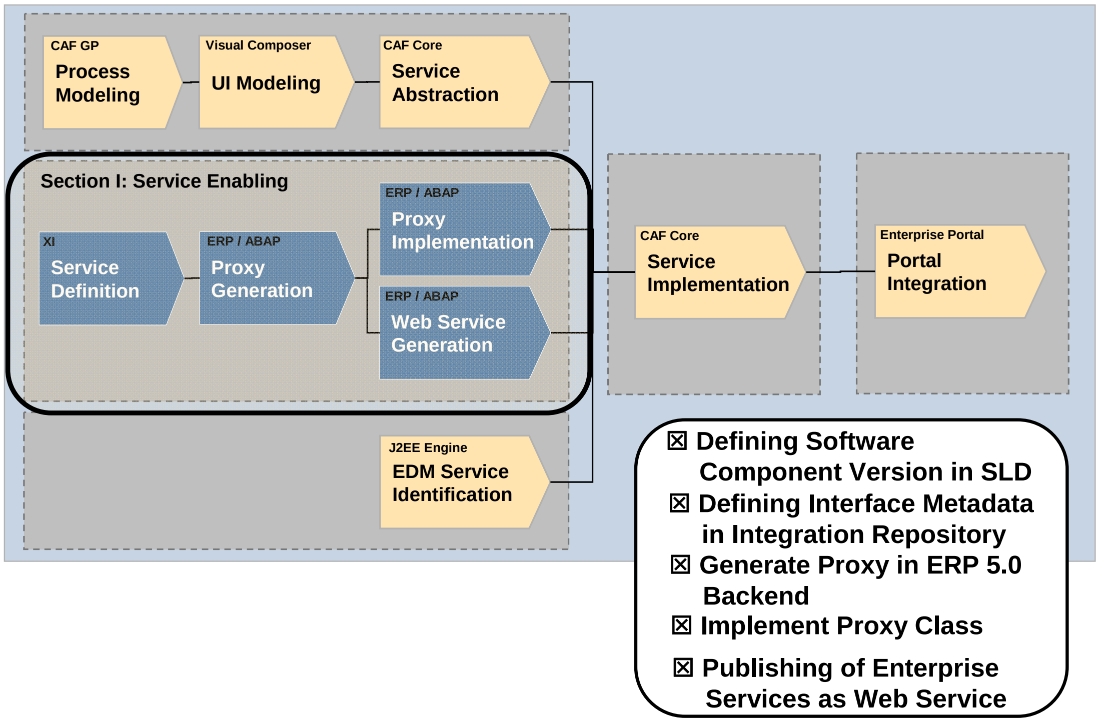
Overview

Service-oriented development allows you to design, build, and implement your own enterprise services using the Enterprise Services Infrastructure.

It is a variant of the Enterprise Service Enabling scenario. This scenario variant focuses on building new functionality by developing service interfaces and its implementation.

The end result will be a callable enterprise service ready for implementation in a client application.

The graphic below provides a summary of the processes for service-oriented development emphasizing the service enabling part:



Scope

In a nutshell, enterprise services are simply Web services that provide enterprise-level business functionality.

Technically, in our case, they are a web services that access existing business functionality in a MySAP ERP backend via ABAP proxies. Implementing those is the focus of this document.

Note: If you currently do not have access to an SAP ERP system, you may omit reading this document for now and simply continue with the following document of this series (Part II Service Enabling). Instead of implementing the respective Composition services through calls to ERP services, you may replace the code using some dummy implementation that you can replace by the correct ERP service calls whenever you get access to an ERP system at a later stage.

However, to understand the borderline of our implementation example here we have to consider the term “enterprise-level business functionality” of our definition of enterprise service.

Enterprise Services may range from very simple lookup services (like finding a material list) to more complex and composite services, but what they have in common is that they are highly integrated in a process or application. Typically enterprise services are high-level components that take more granular Web services and aggregate them into reusable elements with business value.

For example, take the service Create Purchase Order. An elementary Web service Create Purchase Order would simply insert a purchase order in the corresponding database. However, if the stated goal is “create purchase order” in a broader business context the service has to become a more far-reaching enterprise service that handles a process end-to-end, and therefore has to trigger a number of follow-up actions, including:

- Check against production orders
- Trigger a corresponding billing process
- Update of inventory/warehouse information

If you step through this document you will find, that these aspects are unaccounted in our scenario. We assume that you will use either SAP standard interfaces (then this document would be obsolete), or you want to service enable your own business logic or simply follow-up with our example.

In the last case you need to enter some material master data in the ERP backend. An example how this could be done in a simple way is described in Appendix B.

You can browse the enterprise services, which have been already delivered by SAP at the ES Workplace: https://www.sdn.sap.com/irj/sdn/developerareas/esa/esworkplace → *Start Browsing now!* or even test them by registering at https://www.sdn.sap.com/irj/sdn/esareg.

Implementation Steps in the Exchange Infrastructure (XI)

The message orientation of services is a key factor in increasing loose coupling and making enterprise services and the systems built on them flexible and reusable. The message orientation of services refers to the fact that services are able to act as traditional functions that are invoked or can instead send and receive messages.

In the integration builder, you design the relevant interface objects:

- Data types
- Message types
- Service interfaces and their operation modes

At this stage, you are only working with metadata. The objects designed here will subsequently be used to generate and implement services.

This work is done in the XI 3.0 Integration Repository or in future releases in the Enterprise Service Repository (ESR). Objects in one ESR can be transported between different Enterprise Services Repositories. This enables you to transport objects within a system landscape.

Defining a Software Component Version in the System Landscape Directory (SLD)

The service design metadata will need to be associated with a specific software component and version.

To design a new software component and a software component version, you will need to use the SAP System Landscape Directory (SLD) and then import the software component into the Enterprise Services Repository. You can call the SLD from the SAP Exchange Infrastructure (transaction SXMB_IFR). You can then add the new software component to the Integration Repository: Choose Tools → Transfer from System Landscape Directory → Import Software Component Versions.

Create your own product and software component version:

1. In SLD choose Software Catalogue
2. New product: Click *New Product* and type a vendor name, product name and product version and save.
3. The wizard automatically asks you to create a new component version (the new product version and vendor is already preselected by default): Type the component name and a version number and save.

Your software component version has been created.

Designing Interface Metadata in the Integration builder

Starting the Integration Builder

1. Logon to the XI system.
2. Call transaction SXMB_IFR.
3. The tools menu is displayed in a Web browser window (You can also access this URL directly via (http://<host>:<port>/rep).
4. Follow the link Integration Repository and log on.
5. An overview of software components and their versions is displayed.
6. Select a software component and version or import your new component version.
7. Expand the tree for the software component version to display the available namespaces (may be empty in case of a newly creaed component version).
8. Select a namespace.

To add a new namespace to a software component version, open the context menu over the software component version. Choose Open namespaces to display an overview of available namespaces. Choose Edit, add a new namespace to the overview, then save and activate the new namespace.

Expand the namespace, and then expand Interface Objects.

The nodes for data types, message types, and service interfaces are displayed if they have been created. If the namespace has just been created, no nodes will be displayed.

Designing Custom Interface Objects

Design data types:

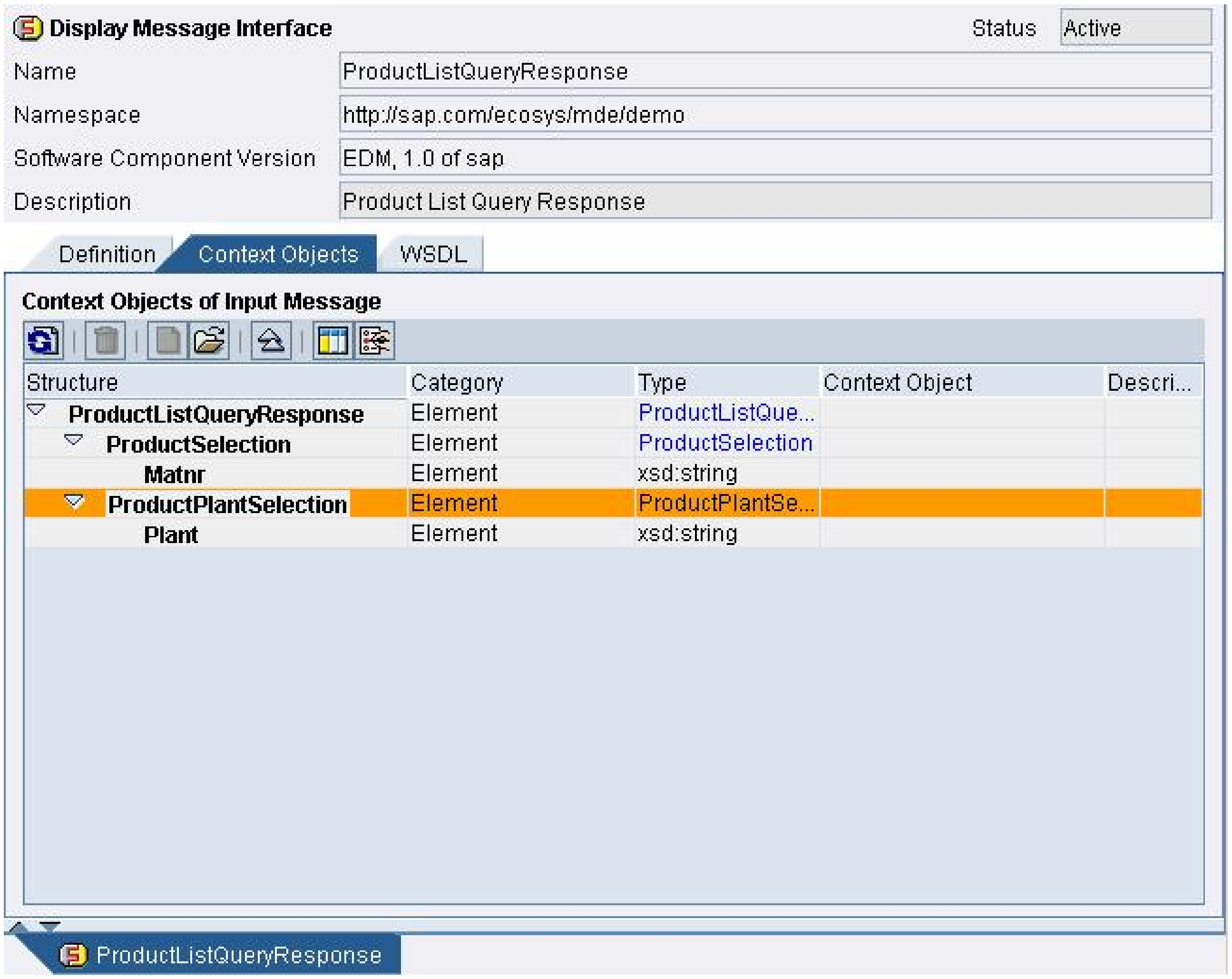
1. Expand the *Data Types* node.
2. From the context menu for the namespace, you can create new data types or copy existing data types.
3. You can add simple (“elementary”) data types, for example “date”, “string” or “integer”. You can combine simple data types to form complex data types. You can define attributes for the data types, for example, occurrence, a value range, optional or mandatory.
4. Activate the data types.

Design Message Types

A message type is a special data type that can be used as a parameter structure in a service interface. You need to associate a message type with a data type.

Design the Message Types:

1. Expand the *Message Types* node.
2. From the context menu, you can create new message types or copy existing message types.
3. Save and activate the message types.
4. The message types need to be activated in order to be used for proxy generation!



Display Message Interface

Status	Active
Name	ProductListQueryResponse
Namespace	http://sap.com/ecosys/mde/demo
Software Component Version	EDM, 1.0 of sap
Description	Product List Query Response

Context Objects of Input Message

Structure	Category	Type	Context Object	Descri...
ProductListQueryResponse	Element	ProductListQue...		
ProductSelection	Element	ProductSelection		
Matnr	Element	xsd:string		
ProductPlantSelection	Element	ProductPlantSe...		
Plant	Element	xsd:string		

Design the Service Interfaces and Their Operations Attributes

The operational behaviour of an interface is defined by its communication parameters. In NW04s you can only define one method per interface meaning one (in asynchronous case) or two (in synchronous case) message types can be exchanged per interface. Additionally the attributes inbound/outbound affects the role of the service.

Determining the direction of an interface:

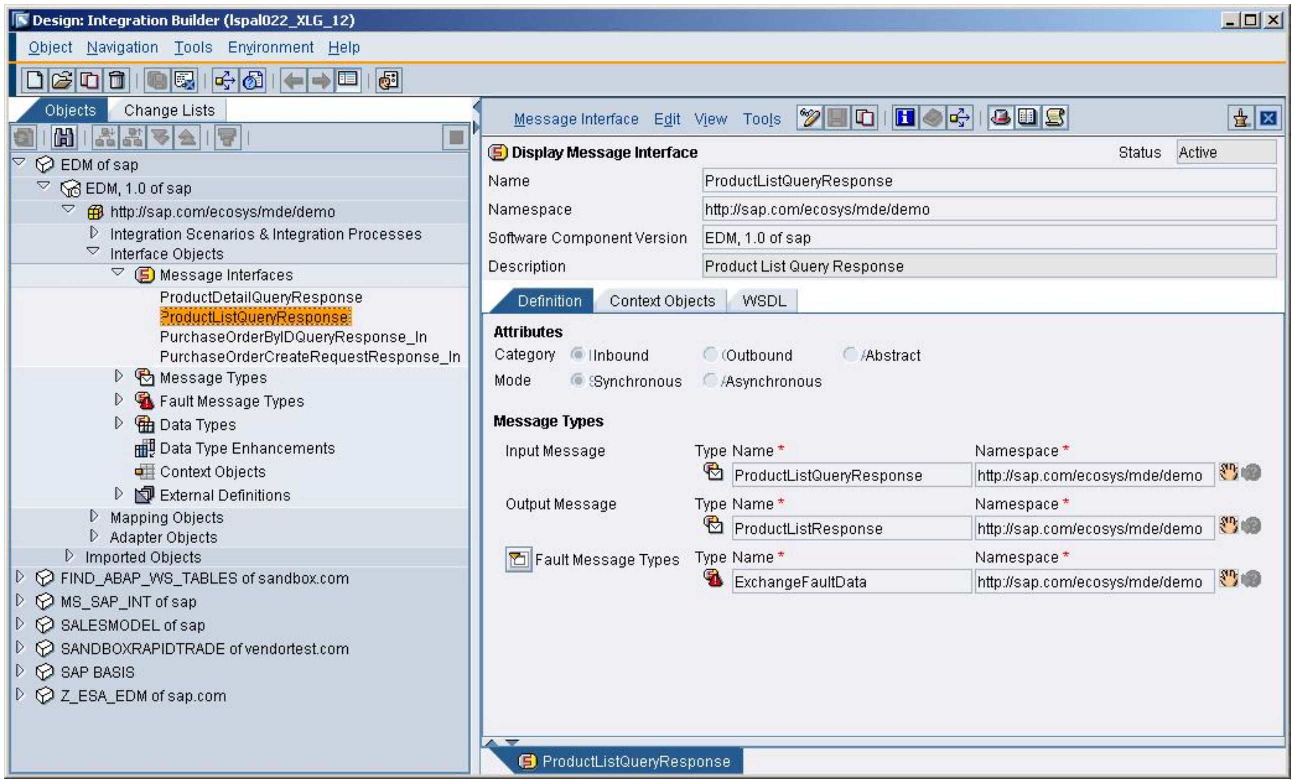
- An **outbound** interface sends a request message and is used for the definition of a client proxy.
- An **inbound** interface receives a request message and is used for the definition of a server proxy.

Defining the mode of communication:

In the case of **synchronous** communication, a response message is expected from the receiver after a request has been sent. Once the request message has been sent, no further messages can be sent until the response to the request has arrived back at the sender system.

In **asynchronous** communication a (immediate) response is not expected. A sending process can send multiple messages to a receiver in a bundle and then continue executing the process

In our example we use server side service provider with synchronous mode of communication. (Vice versa, client proxies can be generated from “outbound” interfaces.)


Design: Integration Builder (lspal022_XLG_12)
Object Navigation Tools Environment Help
Objects
Change Lists
Message Interface Edit View Tools
Definition
Context Objects
WSDL

- EDM of sap
  - EDM, 1.0 of sap
    - http://sap.com/ecosys/mde/demo
      - Integration Scenarios & Integration Processes
      - Interface Objects
        - Message Interfaces
          - ProductDetailQueryResponse
          - ProductListQueryResponse
          - PurchaseOrderByIDQueryResponse_In
          - PurchaseOrderCreateRequestResponse_In
        - Message Types
        - Fault Message Types
        - Data Types
        - Data Type Enhancements
        - Context Objects
        - External Definitions
      - Mapping Objects
      - Adapter Objects
    - Imported Objects
- FIND_ABAP_WS_TABLES of sandbox.com
- MS_SAP_INT of sap
- SALESMODEL of sap
- SANDBOXRAPIDTRADE of vendortest.com
- SAP BASIS
- Z_ESA_EDM of sap.com

Display Message Interface Status: Active

Name	ProductListQueryResponse
Namespace	http://sap.com/ecosys/mde/demo
Software Component Version	EDM, 1.0 of sap
Description	Product List Query Response

Attributes

Category: [✓ Inbound] Outbound Abstract

Mode: [✓ Synchronous] Asynchronous

Message Types

	Type Name *	Namespace *
Input Message	ProductListQueryResponse	http://sap.com/ecosys/mde/demo
Output Message	ProductListResponse	http://sap.com/ecosys/mde/demo
Fault Message Types	ExchangeFaultData	http://sap.com/ecosys/mde/demo

ProductListQueryResponse

The option “Abstract” cannot be used for operations from which you intend to generate a proxy.

Tasks:

1. Expand the Service interfaces node.
2. From the context menu, you can create new service interfaces or copy existing service interfaces.
3. For the interface definition you need to define combinations of communication parameters as described above.
4. Chosse the message type(s) which have to be exchanged on the interface.
5. Save and activate the service interfaces.

Optional: Choose the WSDL tab to display the complete description of the service interface in XML. This description is used to generate and implement a service provider.

The design process in the Enterprise Service Repository is now complete.

Optional Integration Scenarios

As SAP XI covers the functionality of full blown EAI product there are many options to include older R/3 systems as well as 3rd party applications via messaging in custom ESOA scenarios.

For such cases SAP XI also supports the connection of interfaces that cannot be imported to the Integration Repository. There are two variants:

The message schema is contained in a WSDL, XSD, or DTD document and can be processed by proxy generation functions. In this case, you use message interfaces as the counterpart to the interface that cannot be imported.

The message schema cannot be imported or processed by proxy generation. In this case, you use adapters on the sender and receiver side to enable communication.

1.) If your system supports proxy generation and the proxy generation functions can process message schema, proceed as follows:

Import the WSDL, XSD, or DTD document to the Integration Repository as an external definition.

Create a message interface and reference the message(s) of the external definition by using the input help. For external, you can use an abstract message interface since no proxies have to be generated in this case.

Generate a proxy for your message interface. The interface that cannot be imported is already available in the system and can exchange messages with the Integration Server by using the respective adapter.

2.) Communication between Components that do not support Proxies

You can enter interface names in the Integration Builder manually. This enables you to connect systems from which interfaces cannot be imported or whose message schema is not supported by proxy generation. For technical reasons, it is only possible to import RFCs and IDocs for SAP systems Release 4.0 or higher. However, the RFC and IDoc adapters can be implemented with SAP systems Release 3.1I and higher. In this case you must enter the interface names manually. For more information about IDoc and RFC interfaces Release 3.1I and higher, call the Interface Repository at the Internet address ifr.sap.com.

The Adapter Engine supports the connection of external systems that are not necessarily connected by means of an interface (for example, the file adapter). When configuring the inbound adapter, specify instead the ID of the logical sender and receiver by using the respective business system, a namespace, and any interface name. If a mapping is required, you need an interface to be able to specify your mapping program later in the configuration using an interface mapping. For external systems, you can use an abstract message interface since no proxies are to be generated in this case.

In the case of interfaces that have not been imported, if you have entered the repository namespace and the interface name correctly during mapping and routing, the Integration Server recognizes the corresponding communication parties (the namespace does not then necessarily have to be available in the Integration Builder).

Since scenarios referred above are not within the scope of this document, please refer to corresponding documentation for application integration scenarios on help.sap.com.

Importing/Exporting Interface Objects

To copy design objects from one Integration Repository to another Integration Repository the software component version must be the same in the source and target repository. In this way, you provide the design objects of the Integration Repository using export files. SAP Note 836200 gives a brief description of how customers can import process integration content to their Integration Repository.

You have the following options for selecting the objects to be exported:

- All objects of a software component version. When you select this option, the Integration Builder always includes the delete versions in the export.
- All objects of individual namespaces for a software component version.
- Individual objects of a software component version.
- Export of change lists
- Optionally you can also include deleted obejects.

You can only export active objects. If an object to be exported is not active, the Integration Builder exports the last active version.

During the export, a packaged binary file is created in a directory defined on the repository server (or directory server). To import this file to another Integration Repository (or Integration Directory), you must manually copy it to an import directory (see below). You require the appropriate authorizations to be able to access directories on the SAP Web Application Server.

Type	Integration Repository
Export	<systemdir>/xi/repository_server/export
Import	<systemdir>/xi/repository_server/import

<systemdir> is the system directory of your server installation (e.g. /usr/sap/<SID>/sys/global/).

If you have only exported the objects of one namespace, the namespaces of the same software component version are nevertheless visible in the target repository following the import. However, these namespaces are empty because they could not be exported. SAP recommends that you only export complete software component versions so as to avoid confusion.

Import/export design objects of the Integration repository can be performed by using *"Tools → Import/Export Design Objects*" or in case of export functionality by context menu directly from the selected software component version or namespace from the navigation tree.

For transferring design objects within an Integration Repository to other software component versions use *Tools → Release Transfer*... on the design maintenance screen of the Integration Builder and follow the wizard's instructions.

For more details please refer to Transporting XI Objects at help.sap.com.

Enhancement of SAP standard services

You can enhance or modify existing SAP interface objects.

Note: This chapter is only for advanced users already familiar with XI. Those users are recommended to refer to the "Guide for Customer Developments and Modifications in the Integration Repository".

Message interfaces can be defined using message types or external definitions. If message interfaces reference message types, you can enhance them without modification by creating a data type enhancement for the data type that is referenced in the message type. You must create the data type enhancement in the customer software component in the customer namespace. Using the customer namespace prevents naming conflicts between elements or attributes of the enhancement and the names of the data type and other data type enhancements that are also valid in the customer context for this data type (for example, partner enhancements). Elements and attributes of enhancements are qualified with the namespace prefix in XML instances of interfaces.

If you want to modify SAP interface objects at the customer, a „based-on" relationship must be defined between the customer software component and the SAP software component in the SLD. The prerequisite for changing objects in the underlying software components is that the objects *Modifiable* property is set in the customer software component version. To do this, open the customer software component version and select the Objects are Modifiable checkbox under Object attributes. You can then modify an SAP object as follows:

1. In the navigation tree of the customer software component, select the object to be modified in the basis Objects subtree.
2. In the object editor, select change mode. The following message is displayed: Object is defined in a subordinate software component version. Do you want to add the object to the software component and modify it?

3. Choose Modify.mYou can edit the interface objects in the customer component, in other words make modifications to interfaces, (fault) message types, or data types.
4. Activate the change list containing this object.

SAP upgrades do not overwrite the modified version of the object, but the system displays a message that a conflict resolution is necessary.

Implementation Steps in MySAP ERP Backend

After completing the interface definition in the Integration Repository you have to generate or adapt existing ABAP-proxies in the ERP system. Proxy generation converts non-language-specific interface descriptions in WSDL into executable interfaces known as proxies.

Generate ABAP Proxys

ABAP proxy generation enables you to generate proxies to communicate by using the Web service infrastructure.

The counterparts to inbound message interfaces in application systems are server proxies. They are called to start a service that, in the synchronous case, returns a result. The proxy generation functions generate an ABAP object interface (prefix II_) for an inbound message interface.

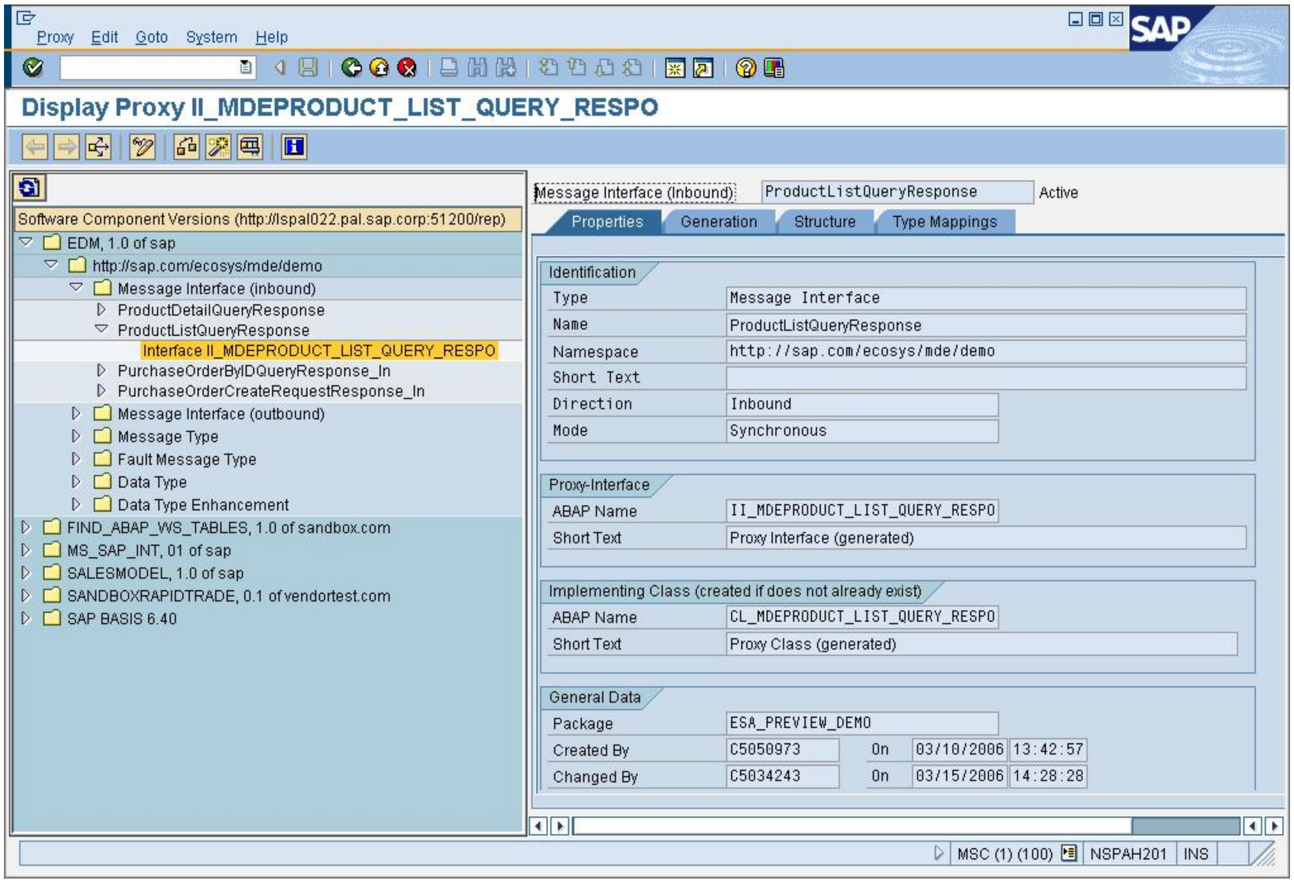
The counterparts to outbound message interfaces in application systems are client proxies. They are called to send a message to an inbound interface. An outbound message interface is mapped to an ABAP object class (prefix CO_).

Proxy generation in ABAP is a task for developers, just like creating programs or structures in the ABAP Dictionary. Therefore, you require developer authorization for your user. Furthermore, we recommend that you create customer objects in the system in the customer namespace (for example, beginning with Z or with /.../). Developers can specify this prefix, together with the package in which the objects are to be created, before proxy generation.

Using the customer namespace is particularly important for the append structures (in case of data type enhancements of existing message types the corresponding proxy objects). Both the append structure name and the field names in the append structure are qualified with the customer prefix. This prevents potential naming conflicts when parallel enhancements are made to the same data type by different partners or customers.

If you want to use objects from other namespaces (for example, from SAP) in your namespace, then the objects must already exist. They are not regenerated or generated during proxy generation.

You can generate the proxy objects by using the transaction SPROXY. To do this, you must first navigate to the customer component and customer namespace in which your interface has been designed. If you call the function (right click) for creating the proxy for your interface and all proxy



Display Proxy II_MDEPRODUCT_LIST_QUERY_RESPO

Software Component Versions (http://lspal022.pal.sap.corp:51200/rep)
- EDM, 1.0 of sap
  - http://sap.com/ecosys/mde/demo
    - Message Interface (inbound)
      - ProductDetailQueryResponse
      - ProductListQueryResponse
        - Interface II_MDEPRODUCT_LIST_QUERY_RESPO
      - PurchaseOrderByIDQueryResponse_In
      - PurchaseOrderCreateRequestResponse_In
    - Message Interface (outbound)
    - Message Type
    - Fault Message Type
    - Data Type
    - Data Type Enhancement
- FIND_ABAP_WS_TABLES, 1.0 of sandbox.com
- MS_SAP_INT, 01 of sap
- SALESMODEL, 1.0 of sap
- SANDBOXRAPIDTRADE, 0.1 of vendortest.com
- SAP BASIS 6.40

Message Interface (Inbound) ProductListQueryResponse Active

Identification

Type	Message Interface
Name	ProductListQueryResponse
Namespace	http://sap.com/ecosys/mde/demo
Short Text	
Direction	Inbound
Mode	Synchronous

Proxy-Interface

ABAP Name	II_MDEPRODUCT_LIST_QUERY_RESPO
Short Text	Proxy Interface (generated)

Implementing Class (created if does not already exist)

ABAP Name	CL_MDEPRODUCT_LIST_QUERY_RESPO
Short Text	Proxy Class (generated)

General Data

Package	ESA_PREVIEW_DEMO			
Created By	C5050973	On	03/10/2006	13:42:57
Changed By	C5034243	On	03/15/2006	14:28:28

MSC (1) (100) NSPAH201 INS

objects will be generated.

Once a proxy has been generated (see picture above) you have to regenerate the proxy after each change of the corresponding interface objects in the integration repository.

Genrated proxy objects:

Interface Object in the Integration Repository	Proxy Object in the System
Message interface (inbound or outbound)	ABAP object interface. You must implement this interface using an ABAP object class to make this service available.
Message Type	ABAP Dictionary structure
Fault message type	ABAP object exception class
Data type	ABAP Dictionary structure(s)/ ABAP Dictionary data element
Data type enhancement	ABAP Dictionary append structure

After proxy generation the following tabs are displayed:

Tab	Meaning
Properties	Generation attributes such as package, last changed by, and so on. For inbound proxies, specify the name of the implementing class here.
Name conflicts	This tab is only displayed immediately after the proxy is generated. It allows you to correct names that were truncated during generation, or that needed to be changed because a collision occurred.
Generation	A list of all the objects generated for an object
Structure	This tab is similar to the Generation tab, except that here the objects are sorted according to their use in a tree structure. Example: Class CO_X ->Method MYMETHOD ->Importing OUTPUT
Documentation	The system displays the documentation from the Integration Repository for the outbound object.
Type mappings	Even if a proxy was generated successfully, there are cases when generation was only possible due to implicit acceptance (for example, restrictions to the value range are checked by the programmer). If such situations arose during generation, they are listed in an application log.

When proxy objects are generated, the number of ABAP Dictionary objects, classes, and interfaces created can lead to a considerable volume of translation. This translation is pointless, however, since these proxy objects do not appear in user interfaces. You should therefore ensure that proxy objects are separated at package level. Create a separate package for the proxy objects and flag it as not relevant for translation.

Implementing the Proxy Class

When the proxy objects have been generated the application must simply implement the class for the server proxy. Proxy generation automatically creates a class with an appropriate signature and empty method. The name of this class is located on the tab page Properties. The application can, however, also enter any class with a suitable signature.

The implementing class uses a method with one of the following naming convention, depending on the type of communication:

In the case of **synchronous** interfaces, the method is called EXECUTE_SYNCHRONOUS

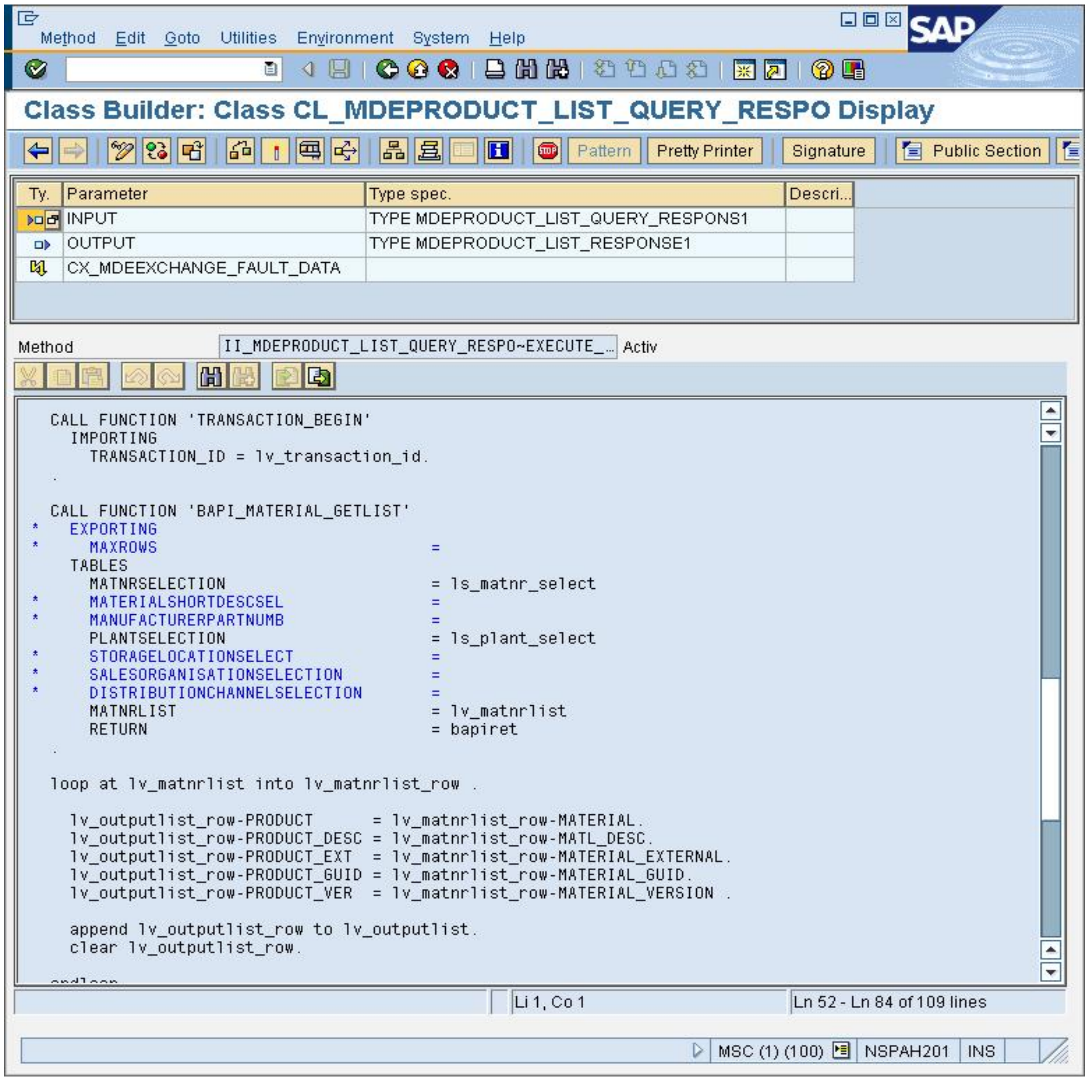
In the case of **asynchronous** interfaces, the method is called EXECUTE_ASYNCHRONOUS

You can either use forward navigation from the properties tab in transaction SPROXY or use SE80 dialogues to edit the class in the class builder.

Implementing the interface call:

1. declare the required data types

```
* transaction controlling
  DATA:
       lt_return          type bapirettab,
       lt_return_errors   type bapirettab,
       lv_transaction_id  type ARFCTID.
**general params and BAPI data
  DATA:
* BAPI input data
       lv_matnrlist_row   type bapimatlst,
       lv_matnr_select    type bapimatram ,
       ls_matnr_select    type table of bapimatram,
       lv_plant_select    type bapimatraw,
```



Class Builder: Class CL_MDEPRODUCT_LIST_QUERY_RESPO Display

Ty.	Parameter	Type spec.	Descri...
	INPUT	TYPE MDEPRODUCT_LIST_QUERY_RESPONS1	
	OUTPUT	TYPE MDEPRODUCT_LIST_RESPONSE1	
	CX_MDEEXCHANGE_FAULT_DATA		

Method II_MDEPRODUCT_LIST_QUERY_RESPO~EXECUTE_... Activ

```
  CALL FUNCTION 'TRANSACTION_BEGIN'
    IMPORTING
      TRANSACTION_ID = lv_transaction_id.
  .

  CALL FUNCTION 'BAPI_MATERIAL_GETLIST'
*   EXPORTING
*     MAXROWS                            =
    TABLES
      MATNRSELECTION                     = ls_matnr_select
*     MATERIALSHORTDESCSEL               =
*     MANUFACTURERPARTNUMB               =
      PLANTSELECTION                     = ls_plant_select
*     STORAGELOCATIONSELECT              =
*     SALESORGANISATIONSELECTION         =
*     DISTRIBUTIONCHANNELSELECTION       =
      MATNRLIST                          = lv_matnrlist
      RETURN                             = bapiret
  .

  loop at lv_matnrlist into lv_matnrlist_row .

    lv_outputlist_row-PRODUCT      = lv_matnrlist_row-MATERIAL.
    lv_outputlist_row-PRODUCT_DESC = lv_matnrlist_row-MATL_DESC.
    lv_outputlist_row-PRODUCT_EXT  = lv_matnrlist_row-MATERIAL_EXTERNAL.
    lv_outputlist_row-PRODUCT_GUID = lv_matnrlist_row-MATERIAL_GUID.
    lv_outputlist_row-PRODUCT_VER  = lv_matnrlist_row-MATERIAL_VERSION .

    append lv_outputlist_row to lv_outputlist.
    clear lv_outputlist_row.
```

Li 1, Co 1 | Ln 52 - Ln 84 of 109 lines

MSC (1) (100) | NSPAH201 | INS

```
       ls_plant_select    type table of bapimatraw,
 * BAPI output data
       lv_matnrlist        type table of bapimatlst,
       bapiret             type bapirettab,
 * XI data structures
       lv_outputlist       type table of mdeproduct_list,
       lv_outputlist_row   type mdeproduct_list
```

2. Preparing the BAPI call you have build up the query type for the material number

```
search input-PRODUCT_LIST_QUERY_RESPONSE-PRODUCT_SELECTION-matnr
    for '.*.'.
  if sy-subrc = 0.
    lv_matnr_select-option = 'CP'.
  else.
    lv_matnr_select-option = 'EQ'.
  endif.
  lv_matnr_select-matnr_low =
    input-PRODUCT_LIST_QUERY_RESPONSE-PRODUCT_SELECTION-matnr.
  append lv_matnr_select to ls_matnr_select.
```

proceed accordingly for plant with `lv_plant_select-option = 'EQ'`

3. call the BAPI to read the material data

```
CALL FUNCTION 'BAPI_MATERIAL_GETLIST'
*   EXPORTING
*     MAXROWS                          =
    TABLES
      MATNRSELECTION                   = ls_matnr_select
*     MATERIALSHORTDESCSEL             =
*     MANUFACTURERPARTNUMB             =
      PLANTSELECTION                   = ls_plant_select
*     STORAGELOCATIONSELECT            =
*     SALESORGANISATIONSELECTION       =
*     DISTRIBUTIONCHANNELSELECTION     =
      MATNRLIST                        = lv_matnrlist
      RETURN                           = bapiret
  .
```

4. the returned table wrapped to the desired interface structure

```
  loop at lv_matnrlist into lv_matnrlist_row .
    lv_outputlist_row-PRODUCT      = lv_matnrlist_row-MATERIAL.
    lv_outputlist_row-PRODUCT_DESC = lv_matnrlist_row-MATL_DESC.
```

```
        lv_outputlist_row-PRODUCT_EXT  = lv_matnrlist_row-
    MATERIAL_EXTERNAL.
        lv_outputlist_row-PRODUCT_GUID = lv_matnrlist_row-MATERIAL_GUID.
        lv_outputlist_row-PRODUCT_VER  = lv_matnrlist_row-
    MATERIAL_VERSION .
        append lv_outputlist_row to lv_outputlist.
        clear lv_outputlist_row.
      endloop .
```

5. and assigned to the methods output parameter.

```
      output-PRODUCT_LIST_RESPONSE-PRODUCT_LIST = lv_outputlist.
```

6. finally you have to check for errors:

**** Check errors

```
      CALL FUNCTION 'EPV_FILTER_MESSAGES'
        EXPORTING
          information = 'X'
          success     = 'X'
          warning     = 'X'
          return_in   = lt_return
        IMPORTING
          return_out  = lt_return_errors
      .
      if lines( lt_return_errors ) > 0.
        call method cl_proxy_fault=>raise
          EXPORTING
            exception_class_name = 'CX_MDEEXCHANGE_FAULT_DATA'
            bapireturn_tab       = lt_return.
      endif.
      CALL FUNCTION 'TRANSACTION_END'
        EXPORTING
          transaction_id = lv_transaction_id
```

You will find the implementation code for all proxies in Appendix A. However, before proceeding it is recommended to enter the scenarios material master data and test the BAPI calls accordingly (see Appendix B).

Publishing of Interfaces as Web Service

The Web Services toolset is integrated in the development environment. A wizard and other tools enable you to publish the implemented proxy application as web service.

The Web Service Creation Wizard makes it possible to define a Web service in a few steps.

After proxy activation in transaction SPROXY you can directly start the web service wizard by choosing the web service wizard push button (F5) or choose Menu *Goto* → *Web service Wizard*.

Perform the steps indicated in the wizard:

Action	Meaning
Create Virtual Interface	The virtual interface maps the names for the operations and parameters of the endpoint for the Web service. Enter a name for the new virtual interface. Assign a short text and choose an endpoint type. Choose the checkbox field Name Mapping. The existing name of the endpoint is copied across. The first letter of a name is written in upper case, all others are lower case. Underscore characters are removed. To change the virtual interface, use the Object Navigator (SE80).
Choose the endpoint	This is the name of your generated ABAP interface and is offered by default.
Create Web Service definition	The features that can be assigned here to the Web service relate to questions of security of data transfer and the type of communication. Choose a predefined feature set from the profiles available. Profile Basic Auth SOAP: Communication type: Stateless Caller authentication: User and password Profile Secure SOAP: Communication type: Stateless Authentication: Client certificate Transferred data is encrypted using the Secure Socket Layer protocol.
Release Web Service	The service definition is created. Finally, you perform the release for the SOAP runtime. If you select Complete, the following objects are created in the Object Navigator: The virtual interface test The Web service test The address of the Web service is default_host/sap/bc/srt/xip/sap/<name of endpoint>

To find the address of the Web service, use the transaction WSADMIN, which supports you to administer web services.

Testing and Debugging

If you plan to implement custom enterprise services on the ERP system, we assume at this point that you are familiar with the ABAP development process itself. So we consider web service level and proxy level as entry point for testing and debugging.

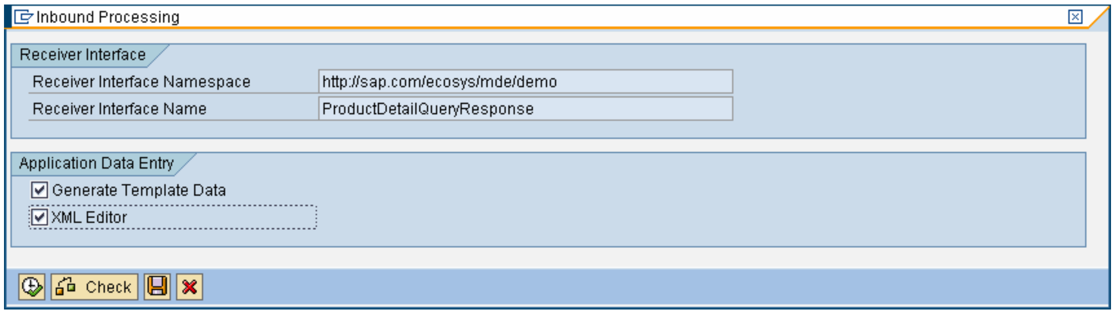
Testing the Web Service

You can call a Web service homepage for any Web service displayed in the transaction WSADMIN. The Web service homepage provides the following services to its user:

- Documentation on the Web service
- The option to display and download the WSDL document
- The ability to testing the Web service

Testing the Proxy

You can test your proxy implementation by choosing the *Testing* push button (F8).


Inbound Processing

Receiver Interface

Receiver Interface Namespace	http://sap.com/ecosys/mde/demo
Receiver Interface Name	ProductDetailQueryResponse

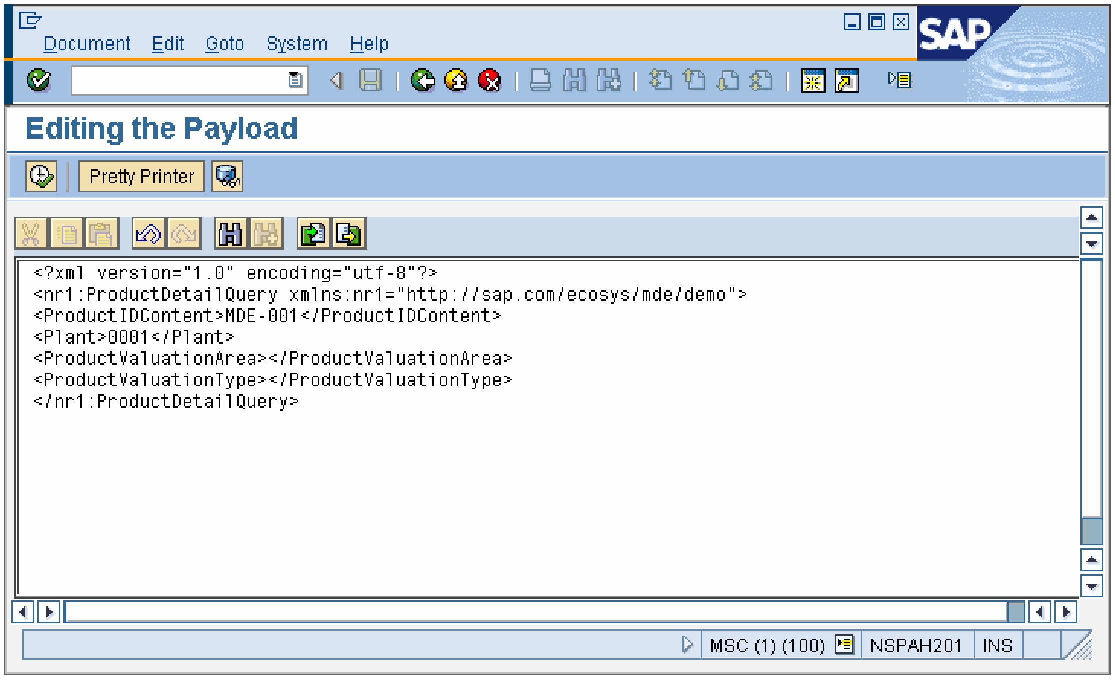
Application Data Entry

☑ Generate Template Data

☑ XML Editor

Check for *XML Editor* to enable editing your test data.

Edit the payload in the xml editor.

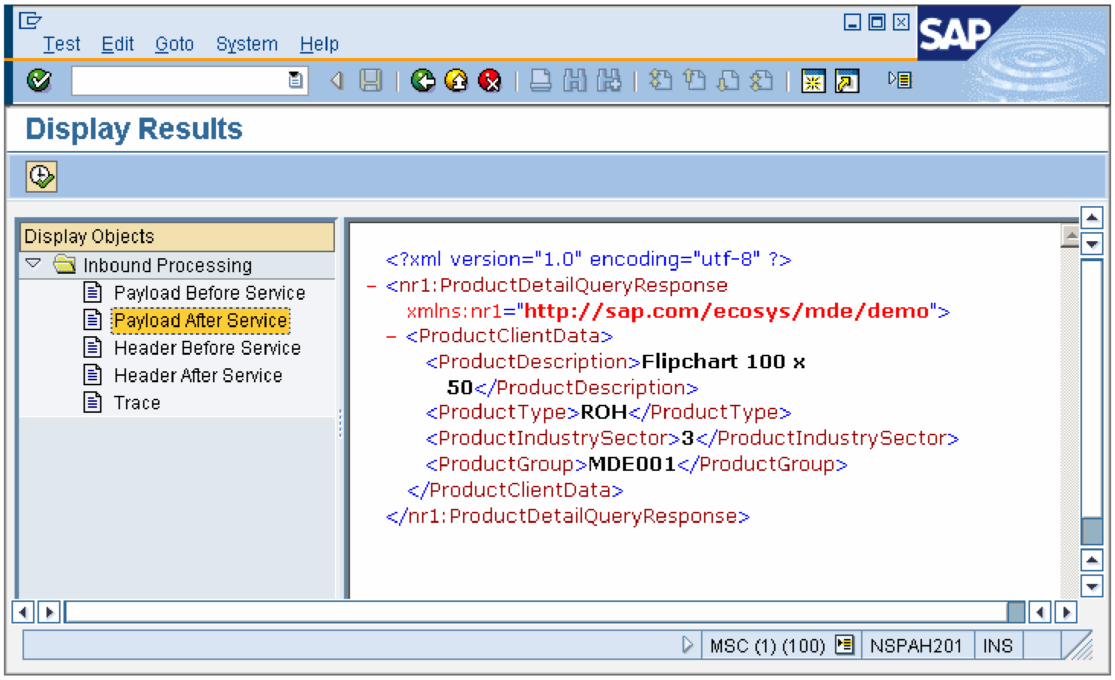


Editing the Payload

```
<?xml version="1.0" encoding="utf-8"?>
<nr1:ProductDetailQuery xmlns:nr1="http://sap.com/ecosys/mde/demo">
<ProductIDContent>MDE-001</ProductIDContent>
<Plant>0001</Plant>
<ProductValuationArea></ProductValuationArea>
<ProductValuationType></ProductValuationType>
</nr1:ProductDetailQuery>
```

Choose *Execute*(F8).

The response message will be displayed – choose *Payload After Service* to see the return values.



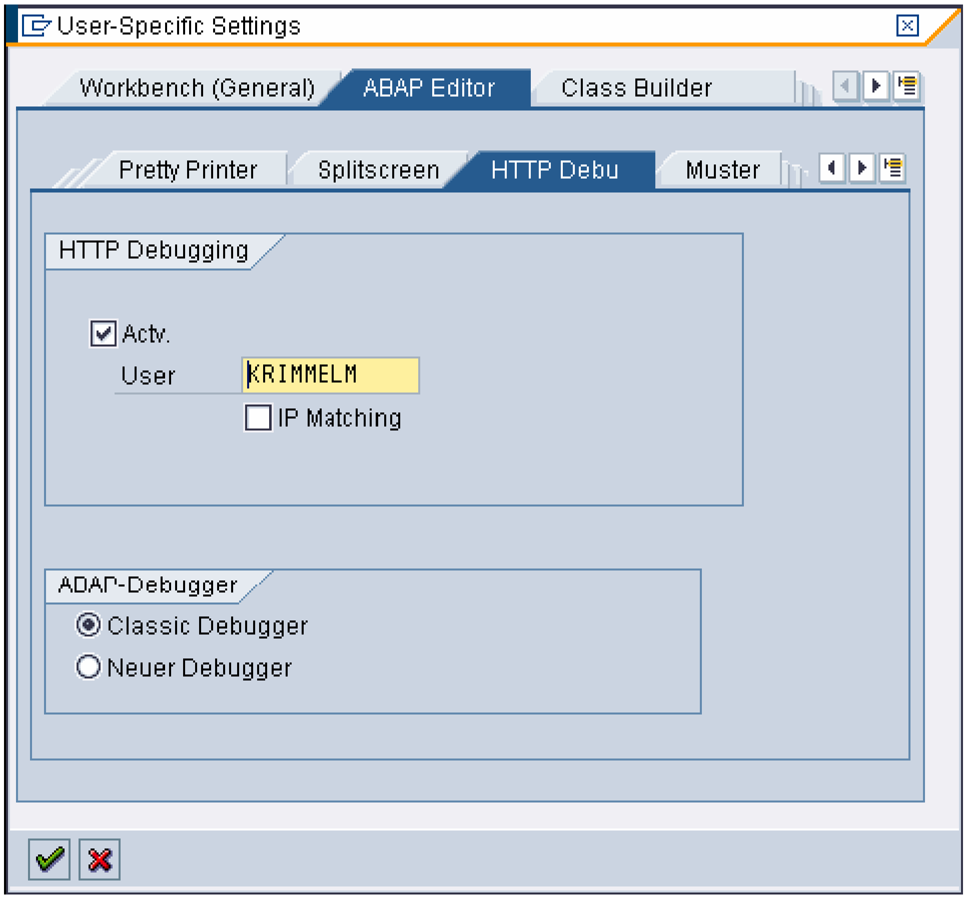
```
<?xml version="1.0" encoding="utf-8" ?>
- <nr1:ProductDetailQueryResponse
   xmlns:nr1="http://sap.com/ecosys/mde/demo">
 - <ProductClientData>
     <ProductDescription>Flipchart 100 x
       50</ProductDescription>
     <ProductType>ROH</ProductType>
     <ProductIndustrySector>3</ProductIndustrySector>
     <ProductGroup>MDE001</ProductGroup>
   </ProductClientData>
 </nr1:ProductDetailQueryResponse>
```

If you need to debug, you can simply place a breakpoint in your code.

Debug Your Interface

If you need to debug your interface by calling it via http you have to activate the debugging as following:

1. Log on with the user that is used to send the message to the application server where the message is sent to.
2. Open the proxy class in SE24:
3. Choose Utilities → Settings in the menu and activate the HTTP debugging for the user you want to use for debugging in ABAP Editor → HTTP Debugging:



HTTP Debugging

[✓] Actv.

User KRIMMELM

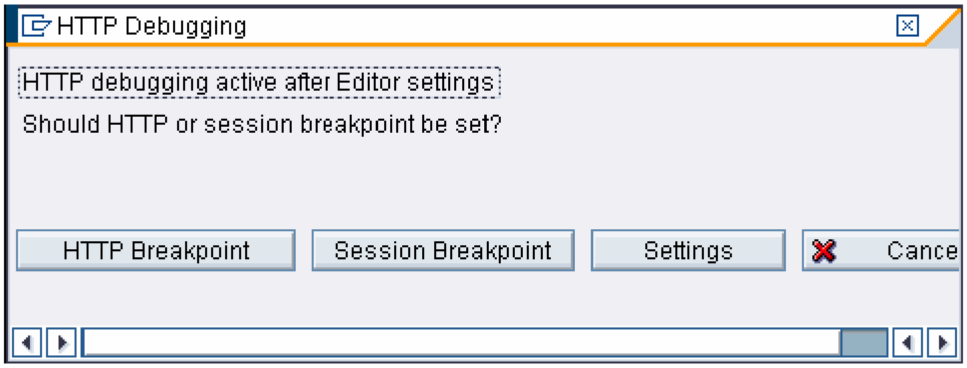
[ ] IP Matching

ABAP-Debugger

(●) Classic Debugger

( ) Neuer Debugger

4. Set the breakpoint via menu Utilities → Breakpoints → Set/Delete → Choose 'HTTP breakpoint' in the popup:


HTTP Debugging

HTTP debugging active after Editor settings

Should HTTP or session breakpoint be set?

When you send the message the debugger will stop at the breakpoint you have set.

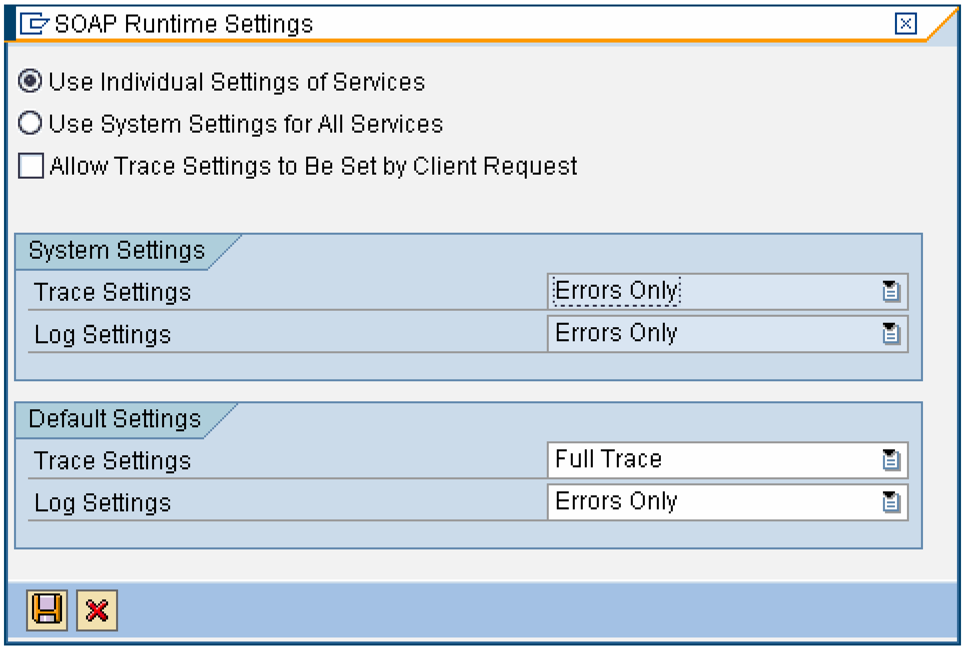
Another possibility to activate the HTTP debugging is to activate the debugging for a specific user in transaction SICF. Select the path and choose via the menu: Edit → Debugging → Activate debugging. In the popup you can set the user for debugging (needs to be the sending user).

When you send the message via web service homepage the debugger will stop at your breakpoint.

Tracking Message Flow on Backend Site

If you tested the proxy itself and the Web service on the backend successfully as well, you may still encounter problems in the message flow between your JAVA and ABAP stack. For this you can exactly trace your message flow on backend site.

Call transaction WSADMIN, switsch trace on *Goto* → *SOAP Runtime Settings* and select the settings as follows:


SOAP Runtime Settings

(●) Use Individual Settings of Services
( ) Use System Settings for All Services
[ ] Allow Trace Settings to Be Set by Client Request

System Settings

Trace Settings	Errors Only
Log Settings	Errors Only

Default Settings

Trace Settings	Full Trace
Log Settings	Errors Only

Call transaction SM59 and display trace via menu *RFC* → *Display Trace.* If you choose full trace all system messages including the SOAP messages of the request and response will be displayed. The snippet below displays a SOAP response message and some INFO messages of the ABAP SOAP runtime.

If errors in the SOAP runtime occur, these are also indicated in the system log (*SM21*).

Don't forget to switch of full trace, if you finished your analysis.

Related Content

For most actions and technologies mentioned in this document there are already existing particular documentations and How-to guides.

How-To-Guides for all referred SAP NetWeaver components are available at the SAP Service Marketplace: SAP Service Marketplace http://service.sap.com/netweaver **-> Media Library -> How–To Guides -> <SAP NetWeaver component>**

or using quick link: **/nw-howtoguides**

All installation guides of required SAP installations are available at the SAP Service Marketplace using quick link: **/instguidesnw04**

SAP Service Marketplace: http://service.sap.com/netweaver **-> Media Library -> Literature**

Collaborative community for developers and integrators: http://www.sdn.sap.com

Enterprise Services Design Guide: http://www.sdn.sap.com **→ Enterprise SOA → Developer Resources**

SAP Exchange Infrastructure: Guide for Customer Developments and Modifications in the Integration Repository: http://service.sap.com/xi **→ Media Library → Documentation**

Appendix

Appendix A Implementation Code

MDEPRODUCT_DETAIL_QUERY_RESPONSE

```
method II_MDEPRODUCT_DETAIL_QUERY_RES~EXECUTE_SYNCHRONOUS.
*** **** INSERT IMPLEMENTATION HERE **** ***
data: material type MATNR,
      valuationarea type BWKEY,
      valuationtype type BWTAR_D,
      plant type WERKS_D.

data: mat_general_data type BAPIMATDOA,
      mat_valuation_data type BAPIMATDOBEW,
      ls_return type BAPIRETURN.
material = input-PRODUCT_DETAIL_QUERY-PRODUCT_IDCONTENT.
plant = input-PRODUCT_DETAIL_QUERY-PLANT.
CALL FUNCTION 'TRANSACTION_BEGIN'
* IMPORTING
*   TRANSACTION_ID       =
.
CALL FUNCTION 'BAPI_MATERIAL_GET_DETAIL'
  EXPORTING
   MATERIAL                    = material
   PLANT                       = plant
*   VALUATIONAREA               = valuationarea
*   VALUATIONTYPE               = valuationtype
 IMPORTING
   MATERIAL_GENERAL_DATA       = mat_general_data
   RETURN                      = ls_return
*  MATERIALPLANTDATA           =
*  MATERIALVALUATIONDATA       = mat_valuation_data
          .

If mat_valuation_data-PRICE_CTRL = 'S'.
output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_VALUATION_DATA-
PRODUCT_PRICE = mat_valuation_data-STD_PRICE.
Else.
output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_VALUATION_DATA-
PRODUCT_PRICE = mat_valuation_data-MOVING_PR.
```

```
EndIf.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_VALUATION_DATA-
PRODUCT_PRICE_UNIT = mat_valuation_data-PRICE_UNIT.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_VALUATION_DATA-
PRODUCT_CURRENCY = mat_valuation_data-CURRENCY.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_VALUATION_DATA-
PRODUCT_CURRENCY_ISO = mat_valuation_data-CURRENCY_ISO.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_DESCRIPTION = mat_general_data-MATL_DESC.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_TYPE = mat_general_data-MATL_TYPE.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_INDUSTRY_SECTOR = mat_general_data-IND_SECTOR.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_GROUP = mat_general_data-MATL_GROUP.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_SIZE_DIMENSIONS = mat_general_data-SIZE_DIM.

output-PRODUCT_DETAIL_QUERY_RESPONSE-PRODUCT_CLIENT_DATA-
PRODUCT_NET_WEIGHT = mat_general_data-NET_WEIGHT.

*}   REPLACE

endmethod.
```

MDEPRODUCT_LIST_QUERY_RESPONSE

```
method II_MDEPRODUCT_LIST_QUERY_RESPO~EXECUTE_SYNCHRONOUS.

* transaction controlling

  DATA:

      lt_return          type bapirettab,

      lt_return_errors   type bapirettab,

      lv_transaction_id  type ARFCTID.

**general params and BAPI data

  DATA:

* BAPI input data

      lv_matnrlist_row   type bapimatlst,

      lv_matnr_select    type bapimatram ,

      ls_matnr_select    type table of bapimatram,

      lv_plant_select    type bapimatraw,

      ls_plant_select    type table of bapimatraw,

* BAPI output data

      lv_matnrlist       type table of bapimatlst,

      bapiret            type bapirettab,

* XI data structures

      lv_outputlist      type table of mdeproduct_list,
      lv_outputlist_row  type mdeproduct_list
  .
  lv_matnr_select-sign = 'I' .
* Allow for material number search pattern 'pat*'
  search input-PRODUCT_LIST_QUERY_RESPONSE-PRODUCT_SELECTION-matnr
    for '.*.'.
  if sy-subrc = 0.
    lv_matnr_select-option = 'CP'.
  else.
    lv_matnr_select-option = 'EQ'.
  endif.
  lv_matnr_select-matnr_low =
    input-PRODUCT_LIST_QUERY_RESPONSE-PRODUCT_SELECTION-matnr.
  append lv_matnr_select to ls_matnr_select.
  lv_plant_select-sign = 'I'.
  lv_plant_select-option = 'EQ'.
  lv_plant_select-plant_low =
    input-PRODUCT_LIST_QUERY_RESPONSE-PRODUCT_PLANT_SELECTION-plant.
  append lv_plant_select to ls_plant_select.
  CALL FUNCTION 'TRANSACTION_BEGIN'
    IMPORTING
      TRANSACTION_ID = lv_transaction_id.
  .
  CALL FUNCTION 'BAPI_MATERIAL_GETLIST'
*   EXPORTING
*     MAXROWS                           =
    TABLES
      MATNRSELECTION                    = ls_matnr_select
*     MATERIALSHORTDESCSEL              =
*     MANUFACTURERPARTNUMB              =
      PLANTSELECTION                    = ls_plant_select
*     STORAGELOCATIONSELECT             =
*     SALESORGANISATIONSELECTION        =
*     DISTRIBUTIONCHANNELSELECTION      =
      MATNRLIST                         = lv_matnrlist
      RETURN                            = bapiret
  .
  loop at lv_matnrlist into lv_matnrlist_row .
```

```
        lv_outputlist_row-PRODUCT      = lv_matnrlist_row-MATERIAL.
        lv_outputlist_row-PRODUCT_DESC = lv_matnrlist_row-MATL_DESC.
        lv_outputlist_row-PRODUCT_EXT  = lv_matnrlist_row-
MATERIAL_EXTERNAL.
        lv_outputlist_row-PRODUCT_GUID = lv_matnrlist_row-MATERIAL_GUID.
        lv_outputlist_row-PRODUCT_VER  = lv_matnrlist_row-
MATERIAL_VERSION .
        append lv_outputlist_row to lv_outputlist.
        clear lv_outputlist_row.
      endloop .
      output-PRODUCT_LIST_RESPONSE-PRODUCT_LIST = lv_outputlist.
**** Check errors
      CALL FUNCTION 'EPV_FILTER_MESSAGES'
        EXPORTING
          information = 'X'
          success     = 'X'
          warning     = 'X'
          return_in   = lt_return
        IMPORTING
          return_out  = lt_return_errors
      .
      if lines( lt_return_errors ) > 0.
        call method cl_proxy_fault=>raise
          EXPORTING
            exception_class_name = 'CX_MDEEXCHANGE_FAULT_DATA'
            bapireturn_tab       = lt_return.
      endif.
      CALL FUNCTION 'TRANSACTION_END'
        EXPORTING
          transaction_id = lv_transaction_id
      .
    endmethod.
```

MDEPURCHASE_ORDER_BY_IDQUERY

```
method II_MDEPURCHASE_ORDER_BY_IDQUER~EXECUTE_SYNCHRONOUS.
 DATA:
    lt_return TYPE TABLE OF bapireturn,
    ls_return TYPE bapireturn,
    ht_return1 TYPE bapirettab,
    ht_return2 TYPE bapirettab,
    ls_hreturn TYPE LINE OF bapirettab.
  DATA:
    lv_purchaseorder TYPE ebeln,
    lv_vendor        TYPE elifn,
    lt_header        TYPE TABLE OF bapiekkol,
    ls_header        TYPE bapiekkol,
    lt_item          TYPE TABLE OF bapiekpo,
    ls_item          TYPE bapiekpo,
    lt_schedule      TYPE TABLE OF bapieket,
    ls_schedule      TYPE bapieket.
  DATA:
    ls_out_header    TYPE MDEPURCHASE_ORDER_HEADER,
    ls_out_item      TYPE MDEPURCHASE_ORDER_ITEM
    lt_out_item      TYPE TABLE OF MDEPURCHASE_ORDER_ITEM,
    ls_out_schedule  TYPE MDEPURCHASE_ORDER_ITEM_SCHEDUL,
    lt_out_schedule  TYPE TABLE OF MDEPURCHASE_ORDER_ITEM_SCHEDUL.
  DATA:
    ls_purchase_order_vendor_addr TYPE
epv_purchase_order_vendor_addr.
  MOVE input-purchase_order_item_list_by_h-purchase_order_id TO
lv_purchaseorder.
  MOVE input-purchase_order_item_list_by_h-vendor_id         TO
lv_vendor.
  CALL FUNCTION 'TRANSACTION_BEGIN'.
  CALL FUNCTION 'BAPI_PO_GETDETAIL'
    EXPORTING
      purchaseorder                   = lv_purchaseorder
      items                           = 'X'
*       ACCOUNT_ASSIGNMENT              = ' '
      schedules                       = 'X'
*       HISTORY                         = ' '
*       ITEM_TEXTS                      = ' '
*       HEADER_TEXTS                    = ' '
*       SERVICES                        = ' '
*       CONFIRMATIONS                   = ' '
*       SERVICE_TEXTS                   = ' '
*       EXTENSIONS                      = ' '
    IMPORTING
```

```
        po_header                          = ls_header
*       po_address                         =
      TABLES
*       PO_HEADER_TEXTS                    =
        po_items                           = lt_item
*       PO_ITEM_ACCOUNT_ASSIGNMENT         =
        po_item_schedules                  = lt_schedule
*       PO_ITEM_CONFIRMATIONS              =
*       PO_ITEM_TEXTS                      =
*       PO_ITEM_HISTORY                    =
*       PO_ITEM_HISTORY_TOTALS             =
*       PO_ITEM_LIMITS                     =
*       PO_ITEM_CONTRACT_LIMITS            =
*       PO_ITEM_SERVICES                   =
*       PO_ITEM_SRV_ACCASS_VALUES          =
        return                             = lt_return.
*       PO_SERVICES_TEXTS                  =
*       EXTENSIONOUT                       =.
    LOOP AT lt_return INTO ls_return.
      MOVE-CORRESPONDING ls_return TO ls_hreturn.
      APPEND ls_hreturn TO ht_return1.
    ENDLOOP.
    CALL FUNCTION 'EPV_FILTER_MESSAGES'
      EXPORTING
        information = 'X'
        success     = 'X'
        warning     = 'X'
        return_in   = ht_return1
      IMPORTING
        return_out  = ht_return1.
    IF LINES( ht_return1 ) < 0.
      CALL METHOD cl_proxy_fault=>raise
        EXPORTING
          exception_class_name = 'CX_FOUNDATION_LAYER_FAULT'
          bapireturn_tab       = ht_return1.
    ELSE.
      ls_out_header-purchase_order_id = ls_header-po_number.
      ls_out_header-vendor_id         = ls_header-vendor.
      ls_out_header-vendor_reference  = ls_header-ref_1.
      LOOP AT lt_item INTO ls_item.
        REFRESH lt_out_schedule.
        LOOP AT lt_schedule INTO ls_schedule WHERE po_item EQ ls_item-
  po_item.
          ls_out_schedule-id                 = ls_schedule-serial_no.
          ls_out_schedule-delivery_date      = ls_schedule-deliv_date.
          ls_out_schedule-quantity-value     = ls_schedule-quantity.
          ls_out_schedule-quantity-unit_code = ls_item-unit.
          APPEND ls_out_schedule TO lt_out_schedule.
        ENDLOOP.
        ls_out_item-product_id                     = ls_item-material.
        ls_out_item-nominator_qua_to_prc_unit      = ls_item-conv_num1.
        ls_out_item-denominator_qua_to_prc_unit    = ls_item-conv_den1.
        ls_out_item-purchase_order_item_id         = ls_item-po_item.
        ls_out_item-quantity-value                 = ls_item-quantity.
        ls_out_item-quantity-unit_code             = ls_item-unit.
        ls_out_item-price-amount-currency_code     = ls_header-
  currency.
        ls_out_item-price-amount-value             = ls_item-net_price.
        ls_out_item-price-base_quantity-value      = ls_item-
  price_unit.
        ls_out_item-price-base_quantity-unit_code = ls_item-
  orderpr_un.
        ls_out_item-price-base_quantity-unit_code = ls_item-
  orderpr_un.
        ls_out_item-schedule_line                  = lt_out_schedule.
        APPEND ls_out_item TO lt_out_item.
      ENDLOOP.


      output-purchase_order_idresponse-purchase_order_header =
  ls_out_header.
      output-purchase_order_idresponse-purchase_order_item   =
  lt_out_item.
    ENDIF.
    endmethod.
```

MDE2PURCHASE_ORDER_CREATE

```
method II_MDE2PURCHASE_ORDER_CREATE_R~EXECUTE_SYNCHRONOUS.
* transaction controlling
  data:
    lt_return             TYPE bapirettab,
    lt_return_errors      TYPE bapirettab,
    lv_transaction_id     TYPE ARFCTID.
**general params and BAPI data
  DATA:
* Values for general parameters
      lv_plant            TYPE werks_d value '0001',
      lv_pur_org          TYPE ekorg   value '0001',
      lv_pur_group        TYPE bkgrp   value '005',
      lv_current_item_no  TYPE bapimepoitem-po_item,
      lv_sched_line       TYPE etenr   value 1,
      lv_del_datcat_ext   TYPE lpein   value 'D',
* BAPI input data
      ls_header           TYPE bapimepoheader,
      ls_headerx          TYPE bapimepoheaderx,
      ls_item             TYPE bapimepoitem,
      lt_item             TYPE TABLE OF bapimepoitem,
      ls_itemx            TYPE bapimepoitemx,
      lt_itemx            TYPE TABLE OF bapimepoitemx,
      ls_schedule         TYPE BAPIMEPOSCHEDULE,
      lt_schedule         TYPE TABLE OF BAPIMEPOSCHEDULE,
      ls_schedulex        TYPE BAPIMEPOSCHEDULX,
      lt_schedulex        TYPE TABLE OF BAPIMEPOSCHEDULX,
* BAPI output data
      lv_order_number TYPE bapimepoheader-po_number,
      bapiret TYPE bapirettab,
* XI data structures
      ls_xi_item TYPE MDE2PURCHASE_ORDER_ITEM_CREATE.
  CALL FUNCTION 'TRANSACTION_BEGIN'
    IMPORTING
      TRANSACTION_ID = lv_transaction_id
  .
*** begin BAPI po create section
  ls_header-purch_org = lv_pur_org.
  ls_header-pur_group = lv_pur_group.
** Fill header values received from caller
  ls_header-vendor = input-purchase_order_create_request-vendor_id.
  ls_header-currency = input-purchase_order_create_request-currency.
```

```
** Flags for provided header files
  ls_headerx-currency = 'X'.
  ls_headerx-vendor = 'X'.
  ls_headerx-purch_org = 'X'.
  ls_headerx-pur_group = 'X'.
* ls_headerx-doc_type = 'X'.
** Items
  lv_current_item_no = 10.
  LOOP AT input-purchase_order_create_request-items INTO ls_xi_item.
** Item fields
    CLEAR ls_item.
    ls_item-plant = lv_plant.
    ls_item-po_item = lv_current_item_no.
    ls_item-material = ls_xi_item-product_id.
    ls_item-quantity = ls_xi_item-quantity-value.
    ls_item-po_unit = ls_xi_item-quantity-unit_code.
    ls_item-net_price = ls_xi_item-price.
    INSERT ls_item INTO TABLE lt_item.
** Flags for provided item fields
    CLEAR ls_itemx.
    ls_itemx-po_item = lv_current_item_no.
    ls_itemx-material = 'X'.
    ls_itemx-plant = 'X'.
    ls_itemx-po_itemx ='X'.
    ls_itemx-quantity = 'X'.
    ls_itemx-net_price = 'X'.
    INSERT ls_itemx INTO TABLE lt_itemx.
** Schedule fields per item
    CLEAR ls_schedule.
    ls_schedule-po_item = lv_current_item_no.
    ls_schedule-sched_line = lv_sched_line.
    ls_schedule-del_datcat_ext = lv_del_datcat_ext.
    ls_schedule-delivery_date = ls_xi_item-delivery_date.
    INSERT ls_schedule INTO TABLE lt_schedule.
** Flags for provided schedule fields
    CLEAR ls_schedulex.
    ls_schedulex-po_item = lv_current_item_no.
    ls_schedulex-po_itemx ='X'.
```

```
      ls_schedulex-sched_line ='X'.
      ls_schedulex-del_datcat_ext = 'X'.
      ls_schedulex-delivery_date = 'X'.
      INSERT ls_schedulex INTO TABLE lt_schedulex.
      ADD 10 TO lv_current_item_no.
    ENDLOOP.
    CALL FUNCTION 'BAPI_PO_CREATE1'
        EXPORTING
          poheader                   = ls_header
          poheaderx                  = ls_headerx
       IMPORTING
         exppurchaseorder            = lv_order_number
       TABLES
         return                      = bapiret
         poitem                      = lt_item
         poitemx                     = lt_itemx
         poschedule                  = lt_schedule
         poschedulex                 = lt_schedulex
  .
  *** end BAPI po create section
  output-purchase_order_create_response = lv_order_number.
  **** Check errors
    CALL FUNCTION 'EPV_FILTER_MESSAGES'
      EXPORTING
        information = 'X'
        success     = 'X'
        warning     = 'X'
        return_in   = lt_return
      IMPORTING
        return_out  = lt_return_errors.
    if lines( lt_return_errors ) > 0.
      call method cl_proxy_fault=>raise
        EXPORTING
          exception_class_name = 'CX_MDEEXCHANGE_FAULT_DAT'
          bapireturn_tab       = lt_return.
    endif.
    CALL FUNCTION 'TRANSACTION_END'
      EXPORTING
```

```
        transaction_id = lv_transaction_id.
    endmethod.
```

Appendix B : Backend Configuration and material master data

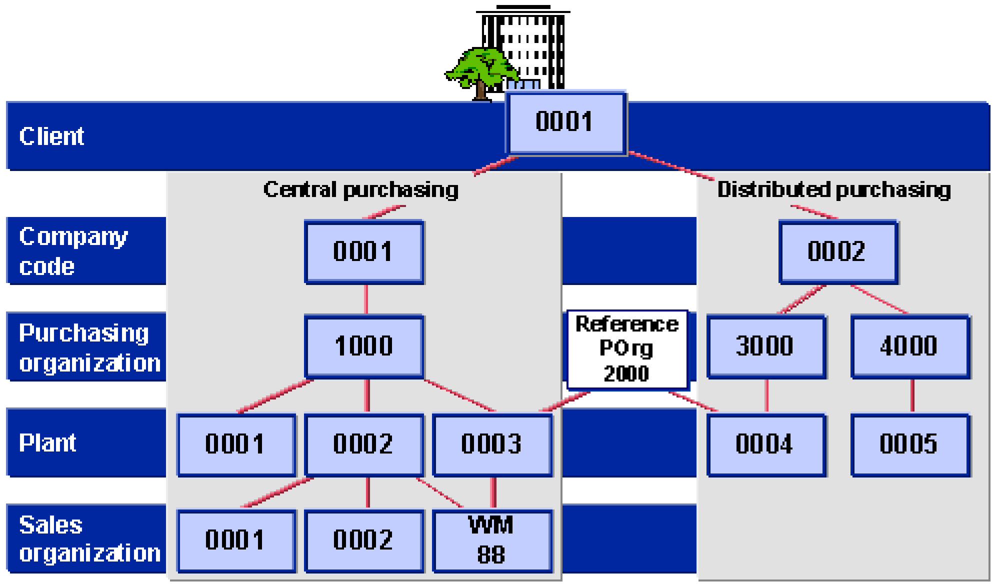
The eSOA Demo Scenario is cutting through all layers of the SAP Netweaver stack including an mySAP ERP backend. Since SAP will deliver substantial ready-to-run enterprise services, which are recommended to be used out-of-the-box and to reduce complexity in this multi layer scenario, we keep the backend implementation part small.

However the eSOA Demo scenario introduces enabling custom eSOA compliant services in the backend, too. If you intend to implement the scenario as it is a few master data have to be edited in a mySAP ERP IDES backend.

This document describes necessary steps to edit the required master data in your ERP system.

Understanding the structure of the master data

The material master has a hierarchical structure resembling the organizational structure of the Company. Some material data is valid at all organizational levels, while other data is valid only at certain levels. This causes some dependencies even for our simple scenario. An example of corporate structure with purchasing organization is given in the following graphic representation:



In every SAP system the client is the top-level of the organizational level. Several company codes can be assigned to one client. In turn, several plants can be assigned to a company code, and several storage locations assigned to a plant.

Master Data

<u>Organisational Structure used for scenario</u>

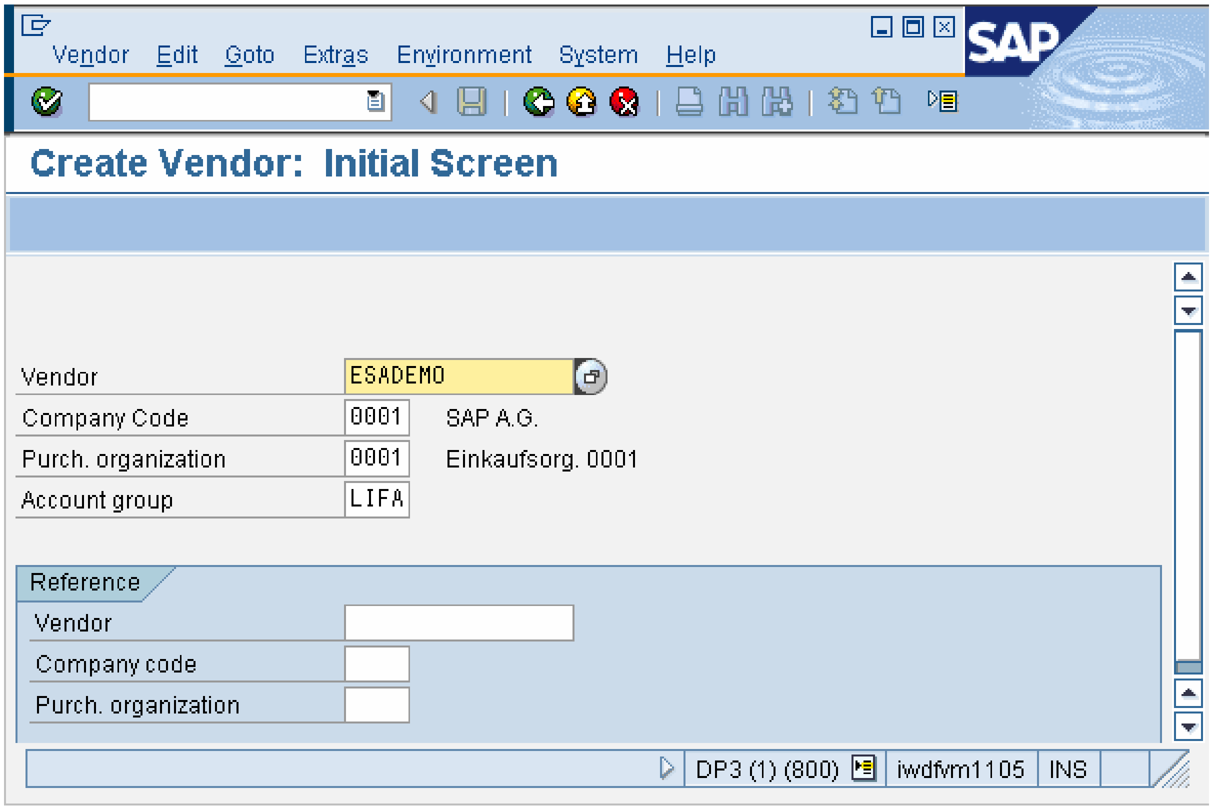
Data	Value
Client	`<your login client>` - not needed to be

	edited explicitly
Company code	`0001`
Purchase Organization	`0001`
Plant	`0001`
Purchase Group	`027`

You may adapt this values according to your own settings.

Creating a vendor

1. Call transaction XK01.
2. Enter vendor name (e.g. ESADEMO) company code ‘0001’ and purchase organization ‘0001’ and Account group ‘LIFA’.



Create Vendor: Initial Screen

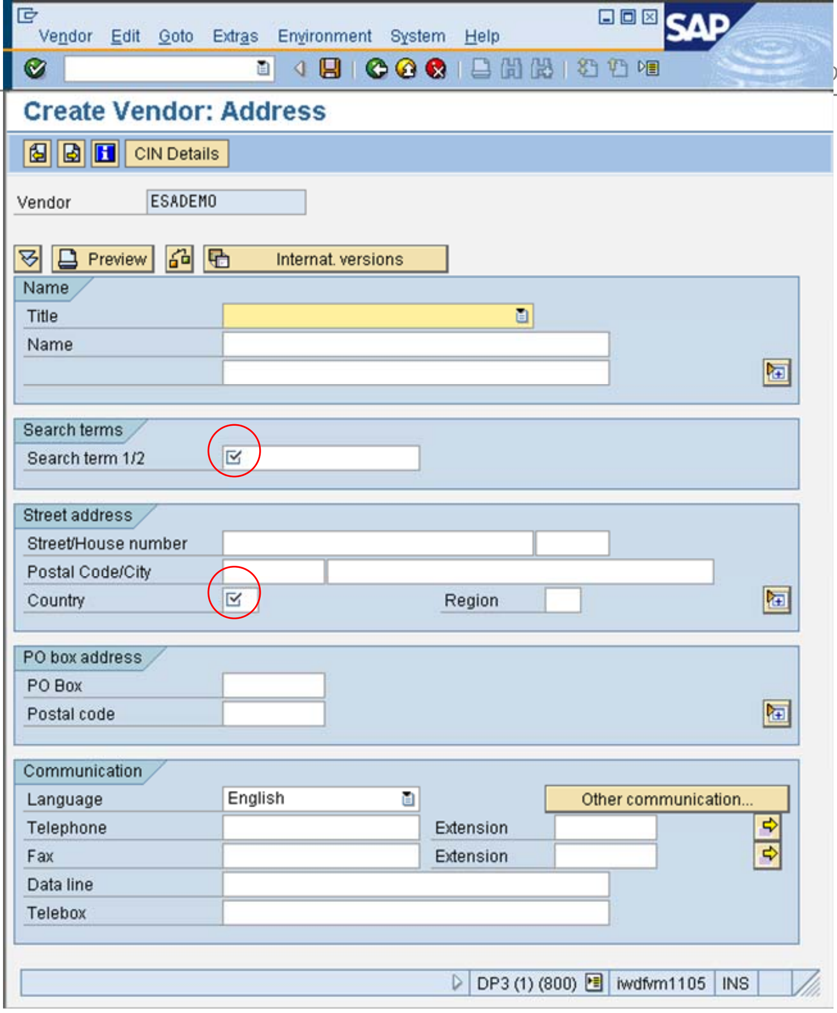
Vendor	ESADEMO	
Company Code	0001	SAP A.G.
Purch. organization	0001	Einkaufsorg. 0001
Account group	LIFA	

Reference

Vendor	
Company code	
Purch. organization	

DP3 (1) (800) iwdfvm1105 INS

3. Enter the vendor address data. At least you have to edit all required fields (marked with checkbox).

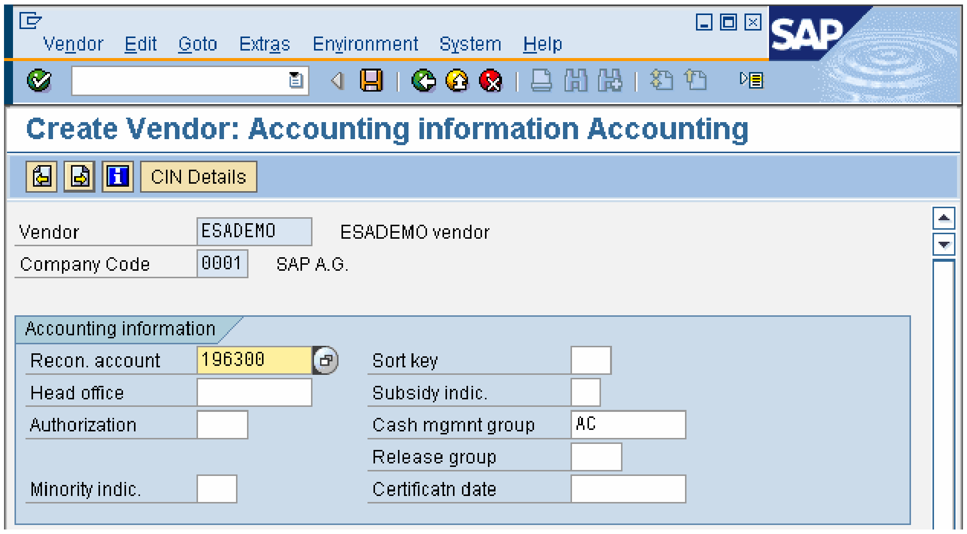


Vendor: ESADEMO

Title	
Name	
Search term 1/2	
Street/House number	
Postal Code/City	
Country	
Region	
PO Box	
Postal code	
Language	English
Telephone	
Extension	
Fax	
Extension	
Data line	
Telebox	

DP3 (1) (800) iwdfvm1105 INS

4. Press (F8) 3 times and proceed with Accounting information view. At least you have to edit Reconciliation account and the cash management group.


Vendor Edit Goto Extras Environment System Help
Create Vendor: Accounting information Accounting
CIN Details
Accounting information

Vendor	ESADEMO ESADEMO vendor
Company Code	0001 SAP A.G.
Recon. account	196300
Head office	
Authorization	
Minority indic.	
Sort key	
Subsidy indic.	
Cash mgmnt group	AC
Release group	
Certificatn date	

5. Save the vendor data.

Create the materials

To provide purchasing option for additional course materials we have to create the materials master data with transaction MM01.

Table 1: Materials used in scenario

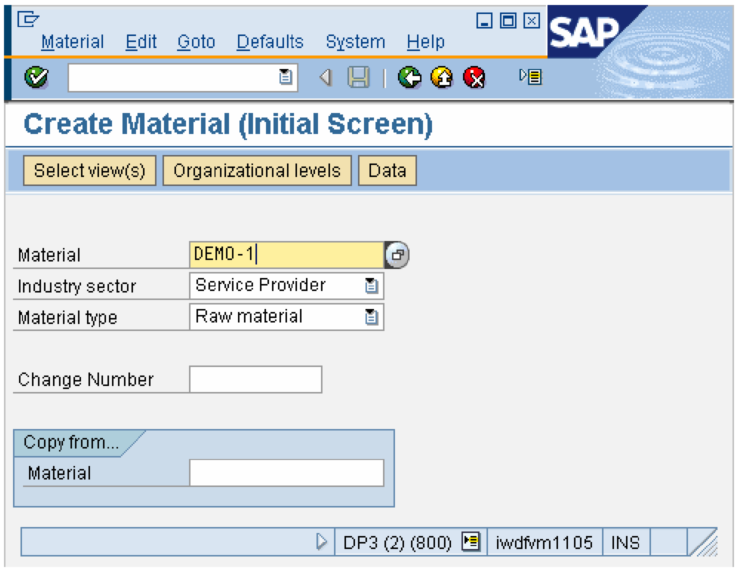
Material-No	Description	Price
DEMO-1	FLIPCHART 100 X 50	40,-
DEMO-2	HANDOUT: ESOA , SAP PRESS	60,-
DEMO-3	HANDOUT: SAP NETWEAVER FOR DUMMIES	45,-
DEMO-4	MODERATION MATERIAL SET	70,-
DEMO-5	PIN BOARD 150 X 150	80,-
DEMO-6	WHITEBOARD 200 X 100	120,-

Table 2: Master data that must be edited for each material

View	Field	Value according scenario
Basic data 1	Material description	See table above
	Base Unit of Measure	PC
	Material group	006
Purchasing	Purchasing group	027
Accounting 1	Base Unit of measure	PC
	Valuation Class	3000
	Standard Price	According to material

Steps:

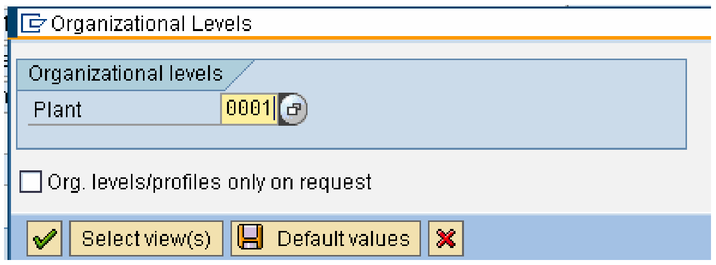
1. Call transaction MM01 and enter Material number, industry type and material type as follows.


Material Edit Goto Defaults System Help
Select view(s)
Organizational levels
Data
Copy from...
DP3 (2) (800) iwdfvm1105 INS

Create Material (Initial Screen)

Material: DEMO-1
Industry sector: Service Provider
Material type: Raw material
Change Number:
Copy from...
Material:

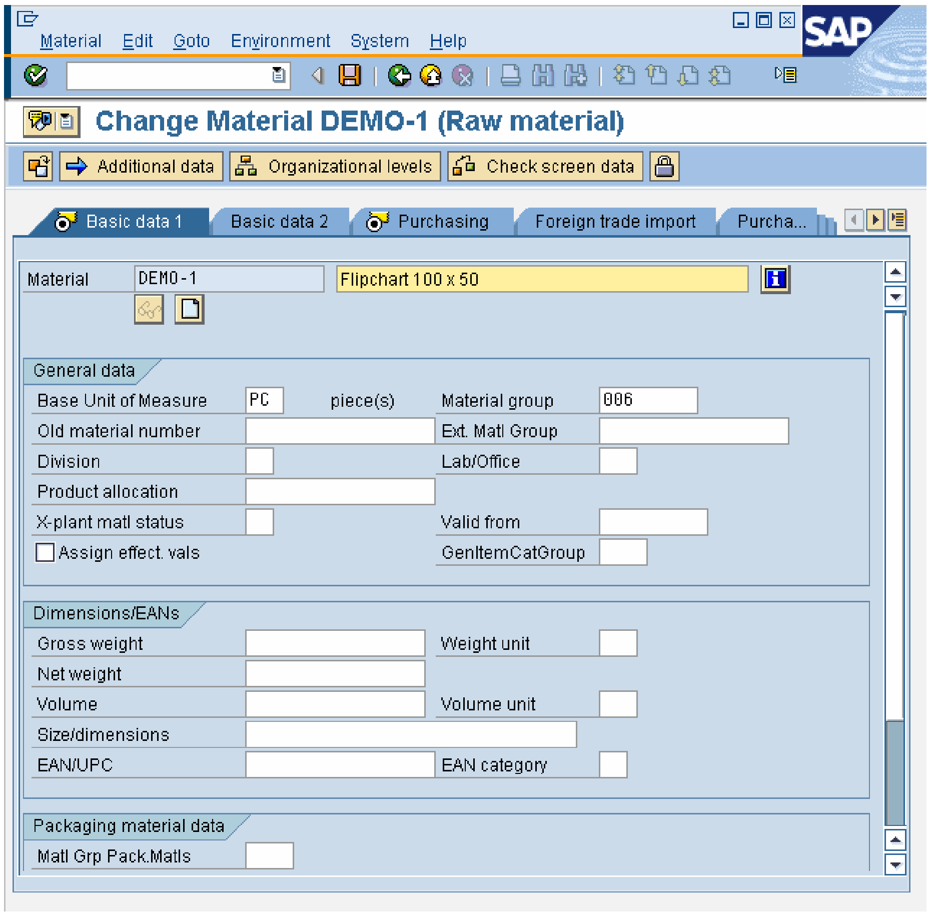
2. Klick ‘Organizational Levels’ and mark ‘purchasing’, ‘basic data1’ and ‘accouning1’ to enable the corresponding views for your materials.



Plant 0001

☐ Org. levels/profiles only on request

3. Use plant ‘0001’ in the following pop-up.
4. Edit values according table 2. The tabs of required views are marked as depicted. You can also select the views directly from symbol on the right hand of the tabulator bar.



Change Material DEMO-1 (Raw material)

Material	DEMO-1	Flipchart 100 x 50	
Base Unit of Measure	PC piece(s)	Material group	006
Old material number		Ext. Matl Group	
Division		Lab/Office	
Product allocation			
X-plant matl status		Valid from	
☐ Assign effect. vals		GenItemCatGroup	
Gross weight		Weight unit	
Net weight			
Volume		Volume unit	
Size/dimensions			
EAN/UPC		EAN category	
Matl Grp Pack.Matls			

You can check sales order creation by using transaction ME21N.

Required values for BAPI call

Structure type	Structure name	Field	Description
header	BAPIMEPOHEADER	purch_org	Purchase organization
		purch_group	Key for a buyer or a group of buyers, who is/are responsible for certain purchasing activities.
		doc_type	Purchasing document type
		vendor	The vendor
		currency	Currency
item table	BAPIMEPOITEM	po_item	Item number
		plant	Plant of material location
		material	Material number
		quantity	Integral quantity number
		po_unit	Unit for quantity
		net_price	Price of the material

You should now test the BAPI calls in SE37 accordingly.