Implementing a Course Request and Approval Process

SAP Composite Application Framework

Document Version 1.00 – September 2005

THE BEST-RUN BUSINESSES RUN SAP

Documentation on SAP Service Marketplace
You can find this documentation at
`service.sap.com/instguidesNW04`

THE BEST-RUN BUSINESSES RUN SAP

Typographic Conventions

Type Style	Represents
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Meaning
⚠	Caution
💬	Example
💡	Note
🧭	Recommendation
syn	Syntax

Contents

1 Introduction

1.1 Purpose

This documentation describes a process, in which you request a course, get an approval for your request, and finally book the course and receive a confirmation. User data is retrieved from a backend system, and the information about the booking is stored into a database.

This example demonstrates how you can integrate a heterogenous environment into a user-centric composite application using the SAP Composite Application Framework tools. The process flow is modeled in Guided Procedures, and the access to the backend system is implemented with services provided by CAF Core.

1.2 Implementation

The following figure shows the components that build the course booking process.

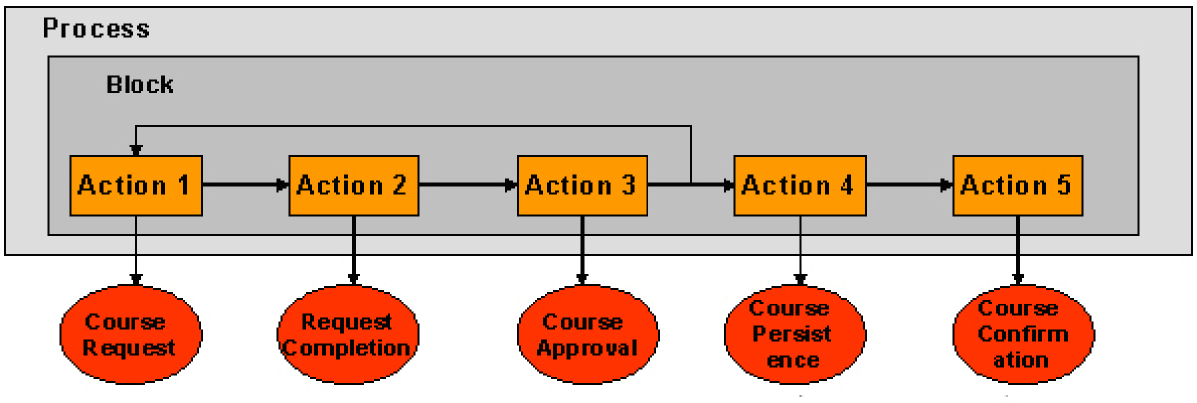


Figure 1

The individual process steps are implemented as callable objects in Guided Procedures, and are encapsulated in actions. The actions are then integrated into a block, which defines the action execution order. Finally, the block defines the process flow.

In the first step of the the process an employee requests a course. He or she provides a user ID, course title, data, and price in an input form. The data is submitted to an SAP system and details about the requester are retrieved. Next, the requestor’s manager reviews the request and either approves or rejects it. In case of rejection, the requestor receives notification, and can modify the entries and re-request approval. In case of an approval, all data is saved in a database for further processing and the requester receives an approval notification.

The implementation of the process involves the following steps:

1. Creating an entity service Course for storing the data for the approved course.
2. Creating an application service CourseAppService that uses the entity service and can be integrated into a Guided Procedures callable object.
3. Creating the callable objects:
   - A Web Dynpro input form for the course request
   - An RFC callable object for access to the SAP system
   - A Web Dynpro approval form
   - A callable object of Composite Application Service type for persisting the course request data
   - A data display form for the request confirmation

4. Encapsulating the callable objects into actions.
5. Creating a sequential block.
6. Creating the process template.

Steps 1 and 2 are implemented using the Composite Application Services (CAS) perspective in the SAP NetWeaver Developer Studio.

Steps 3 through 6 are completed with the Guided Procedures design time. Guided Procedures is available in the Enterprise Portal. To access its tools, you must hold the appropriate authorizations.

For more information, see the CAF documentation on the SAP help portal at **`http://help.sap.com/nw04s`**. The documentation is available under SAP NetWeaver Library → *SAP NetWeaver by Key Capability* → *Composite Application Framework by Key Capability.*

2 Service Layer: Modelling the Services

2.1 Create Entity Service *Course*

2.1.1 Create a New Project:

1. Start the SAP NetWeaver Developer Studio.
2. To create a new project, choose New → *Project…*.

   The *New Project* wizard pops up. Select *Development Component* on the left-hand side, and *Development Component Project* on the right-hand side of the window. Choose *Next*.
3. Expand the *Local Development* node and select *My Components*. Choose *Next*.
4. Enter **`xteched`** in lower-case letters for project name, and a project caption. Choose *Composite Application Services* for the development component type. Choose *Next*.
5. To complete the procedure, choose *Finish*. The system notifies you whether the project creation has been successful.

2.1.2 Create the Entity Service

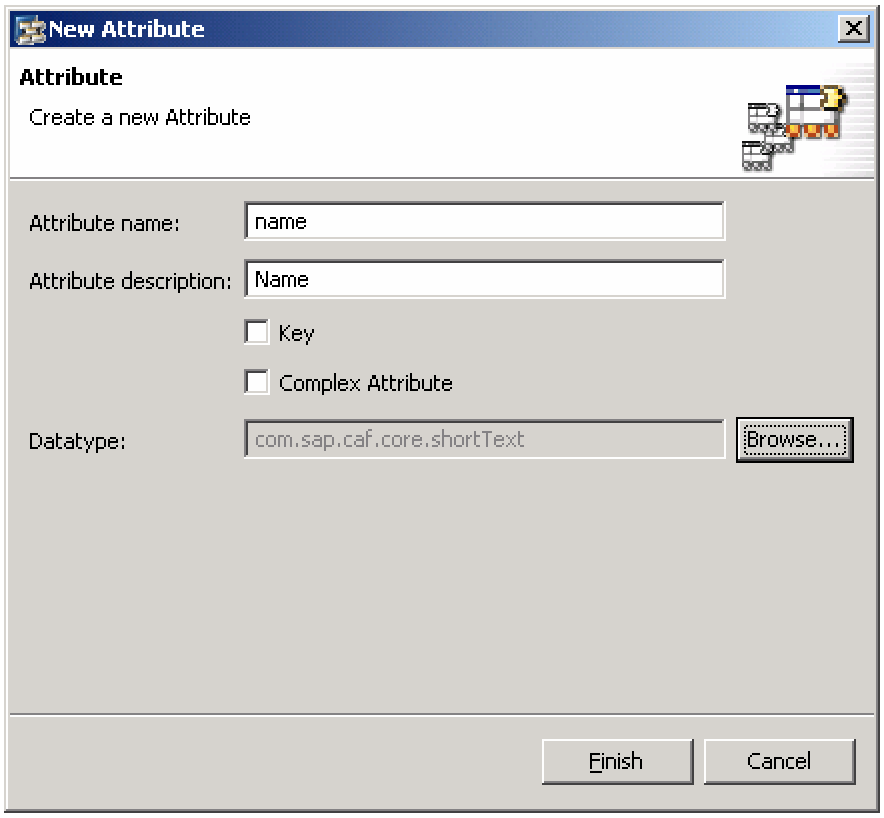
1. Open the CAS perspective by choosing Window → *Open Perspective* → *Composite Application Services.*
2. Open the *Service Explorer* view.
3. Expand the `xteched` node and select *Entity Services*. Open the context menu using the secondary mouse button, and choose *New*.
4. Enter **`Course`** as the entity service name and choose *Finish*.

   In the *Service Explorer* view, a new node appears under *Entity Services*. The service editor opens and you can edit the service configuration.

2.1.3 Create Attributes for the Entity Service

1. In the service editor, select the *Attributes* tabstrip of the Course entity.
2. In the *Attributes* area, open the context menu for the *Course* node, and choose *Create Attribute*.
3. In the dialog window that appears, enter **`name`** (lower case **`n`**) for the attribute's name and **`Name`** (upper case **`N`**) for the attribute's description. The description will be used as a label for an input field in a UI pattern, if we create such for the entity.
4. Choose *Browse* to assign a datatype to the field. In the *Simple Type Selection* window expand the `com.sap.caf.core` node, and select the `shortText` datatype.

   Choose *OK* and then *Finish*.



Attribute name: name
Attribute description: Name
☐ Key
☐ Complex Attribute
Datatype: com.sap.caf.core.shortText

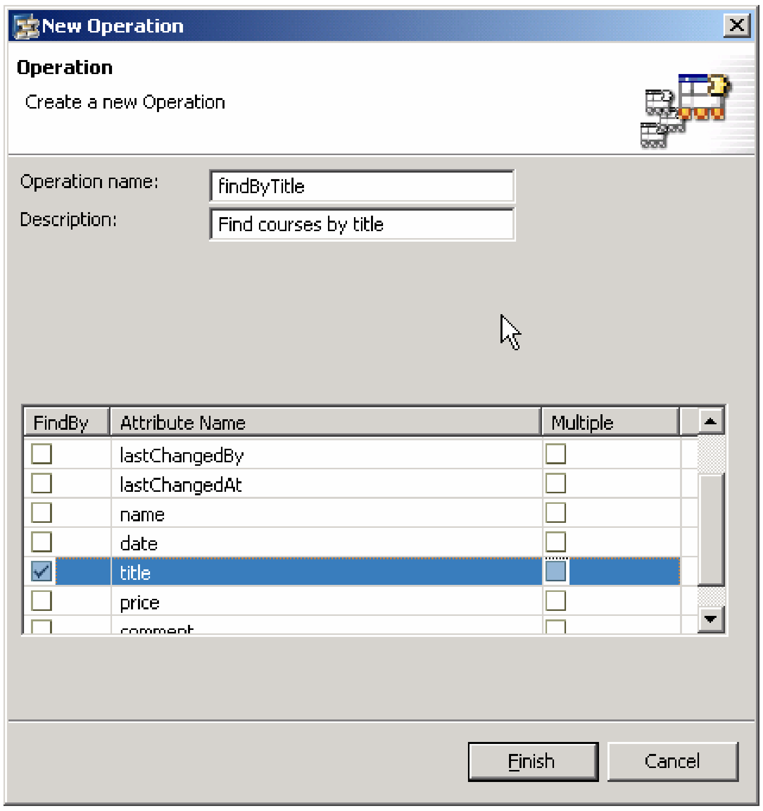
Figure 2

5. Repeat steps 2 to 4 to create the **`date`**, **`title`**, **`price`** (all of type `shortText`), and **`comment`** (of type `longText`).

2.1.4 Create Additional Operations for the Entity Service

1. Select the Operations tabstrip and choose *Add…*. You can add finder methods to the entity.
2. In the dialog window that appears, enter **`findByTitle`** for *Operation name*.

   Add the description **`Find courses by title.`**
3. Select the checkbox for the title attribute in the *findBy* column in the attributes list (see figure 3).
4. Choose *Finish*.


New Operation

Operation

Create a new Operation

Operation name: findByTitle

Description: Find courses by title

FindBy	Attribute Name	Multiple
☐	lastChangedBy	☐
☐	lastChangedAt	☐
☐	name	☐
☐	date	☐
✓	title	☐
☐	price	☐
☐	comment	☐

Figure 3

2.1.5 Additional Settings

You can also check the following entity service settings:

- The *Persistency* tabstrip displays the database table(s) that are to be created as a result of the modeling activities.
  The *Local Persistency* checkbox is selected to indicate that all data will be saved and handled by CAF.
  You must not change anything on this tabstrip.
- The *Datasource* tabstrip is important in case that you disable the *Local Persistency* option in the *Persistency* tabstrip.
  If this is the case, you can assign the entity's lifecycle methods to external calls (for example, to a BAPI or to a Web service).
- For example simplicity, permission checks are not covered at runtime. Therefore, go to the *Permissions* tabstrip and disable the option *Permission checks enabled* (see figure 8).

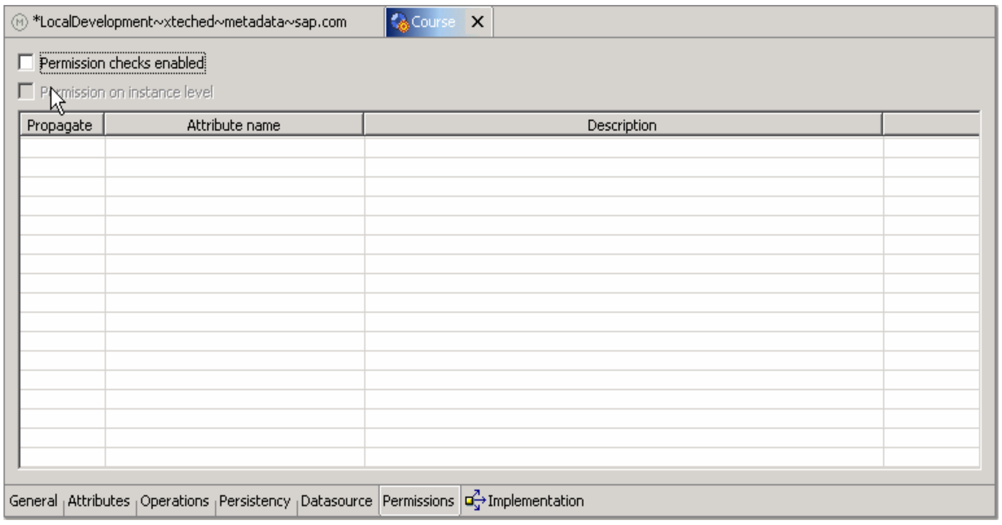


Figure 4

- Finally, check the Implementation tabstrip.

  It displays the coding generated as a result of the modeling activities. You are not allowed to change the code here. If additional logic is required, you must implemented in an application service.

2.1.6 Save and Test the Entity

1. To create and save all metadata for all projects, choose (*Save All Metadata*).
2. Generate the Java code for the complete project.

   In the *Service Explorer* view, open the context menu of the `xteched` project, and choose *Generate All Project Code* (see figure 9).

   The system notifies you whether the operation was successful.

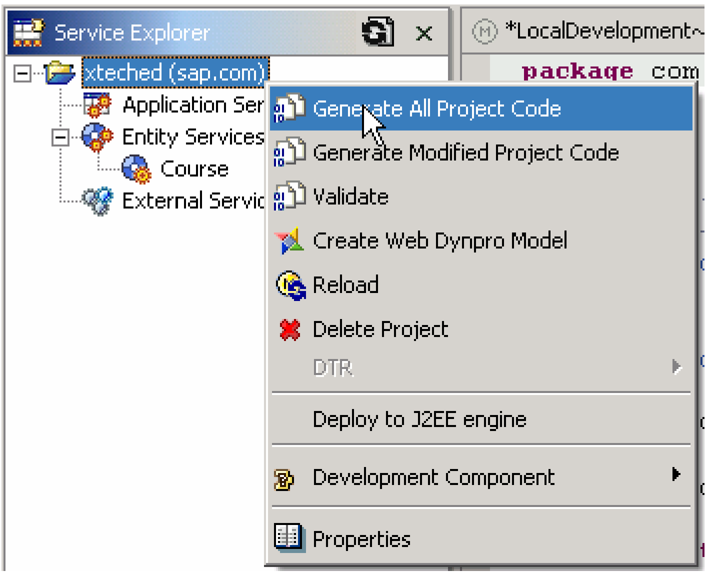


Figure 5

3. To build the project for deployment, open the context menu of the xteched project and choose *Development Component → Build…*.

Make sure that all projects are selected in the *Build Development Components* dialog. Choose *OK*.

4. To deploy the project on the Java server, open the context menu of the xteched project and choose *Deploy to J2EE engine* (figure 6).

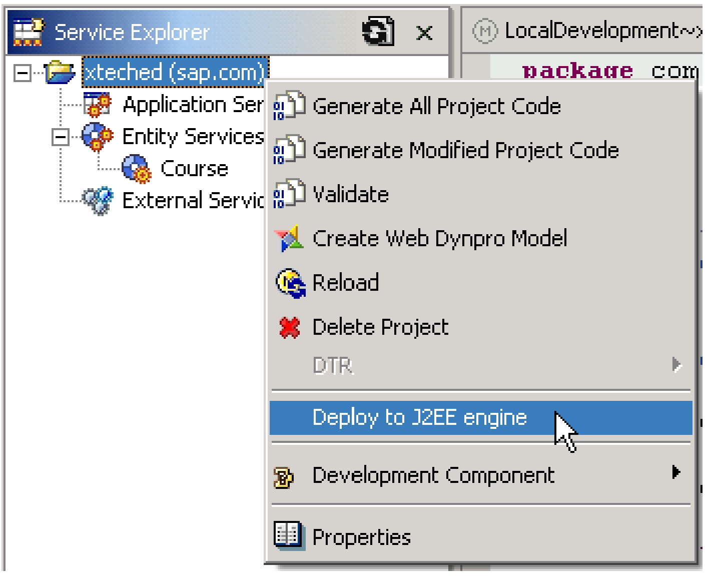
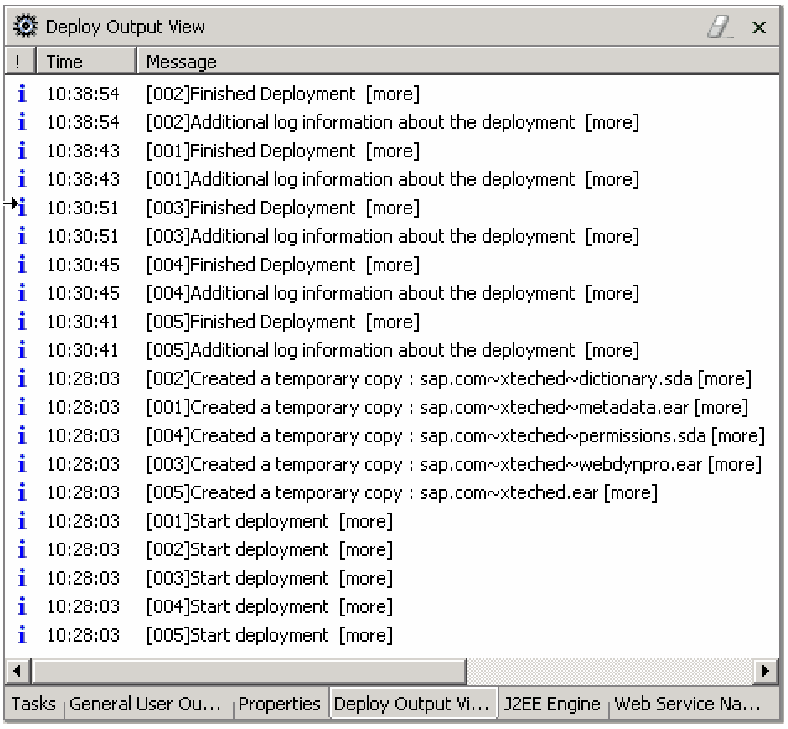


Figure 6

You might be requested to log on to the SDM server. In this case, make sure you supply the credentials of Java server Administrator user.

5. Wait until all projects are deployed. You can check the deployment progress in the Deploy Output View (figure 7).



```
!  Time      Message
i  10:38:54  [002]Finished Deployment  [more]
i  10:38:54  [002]Additional log information about the deployment  [more]
i  10:38:43  [001]Finished Deployment  [more]
i  10:38:43  [001]Additional log information about the deployment  [more]
i  10:30:51  [003]Finished Deployment  [more]
i  10:30:51  [003]Additional log information about the deployment  [more]
i  10:30:45  [004]Finished Deployment  [more]
i  10:30:45  [004]Additional log information about the deployment  [more]
i  10:30:41  [005]Finished Deployment  [more]
i  10:30:41  [005]Additional log information about the deployment  [more]
i  10:28:03  [002]Created a temporary copy : sap.com~xteched~dictionary.sda [more]
i  10:28:03  [001]Created a temporary copy : sap.com~xteched~metadata.ear [more]
i  10:28:03  [004]Created a temporary copy : sap.com~xteched~permissions.sda [more]
i  10:28:03  [003]Created a temporary copy : sap.com~xteched~webdynpro.ear [more]
i  10:28:03  [005]Created a temporary copy : sap.com~xteched.ear [more]
i  10:28:03  [001]Start deployment  [more]
i  10:28:03  [002]Start deployment  [more]
i  10:28:03  [003]Start deployment  [more]
i  10:28:03  [004]Start deployment  [more]
i  10:28:03  [005]Start deployment  [more]
```

Figure 7

6. To test your entity service, choose *Entity Services* → *Course,* and open its context menu. Choose Test (figure 8).

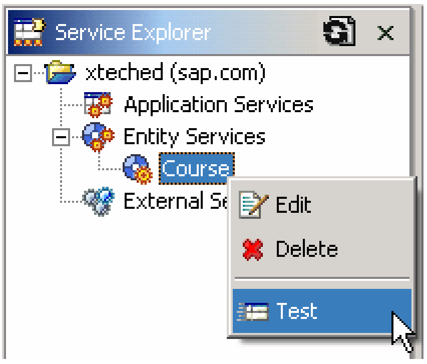
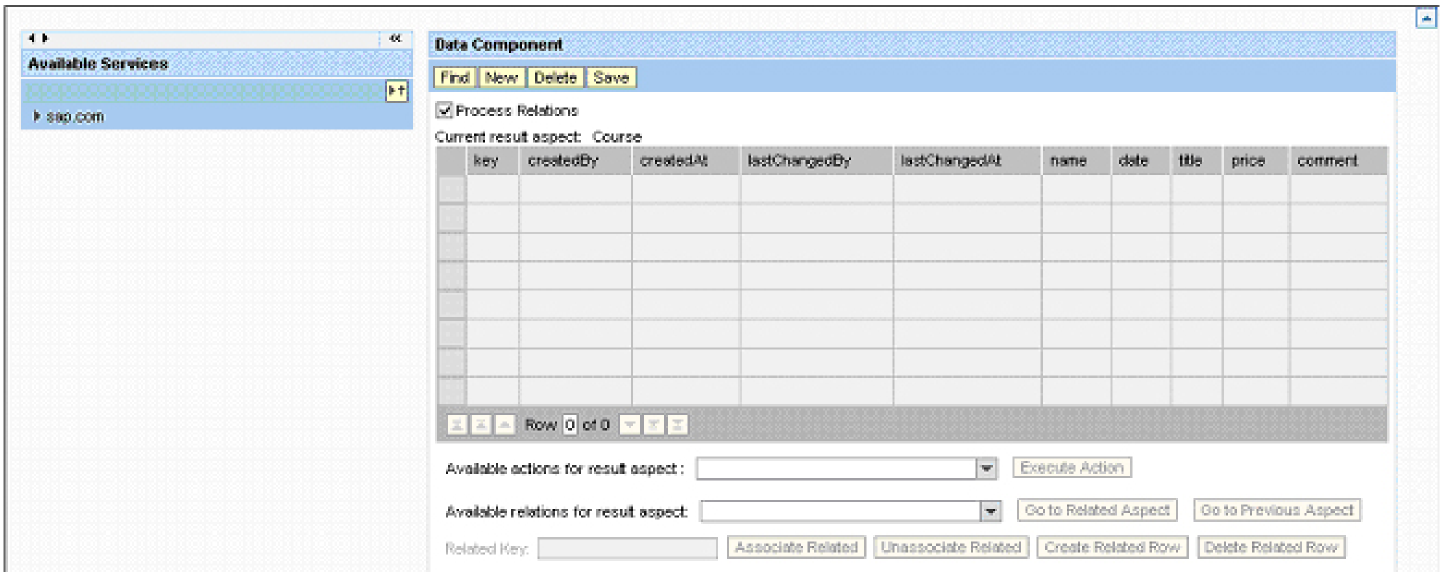


Figure 8

You might be requested to log on to the Java server. Enter the credentials of a user that holds administration permissions.

7. The Service Browser opens in a new window (figure 9).



Current result aspect: Course

	key	createdBy	createdAt	lastChangedBy	lastChangedAt	name	date	title	price	comment

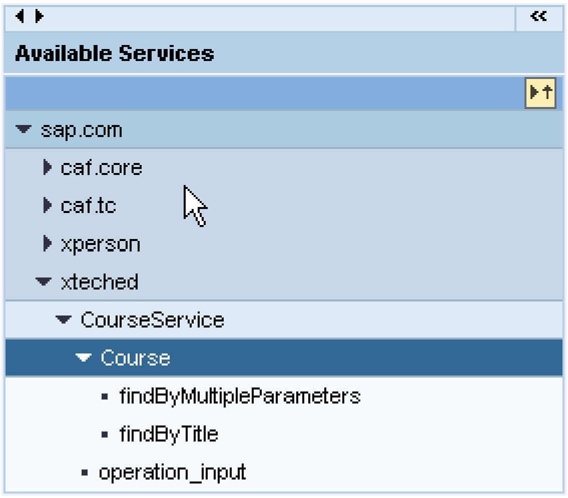
Available actions for result aspect :

Available relations for result aspect:

Related Key:

Figure 9

8. Under *Available Services*, choose *sap.com* → *xteched* → *CourseService* → *Course* (figure 10).



Available Services

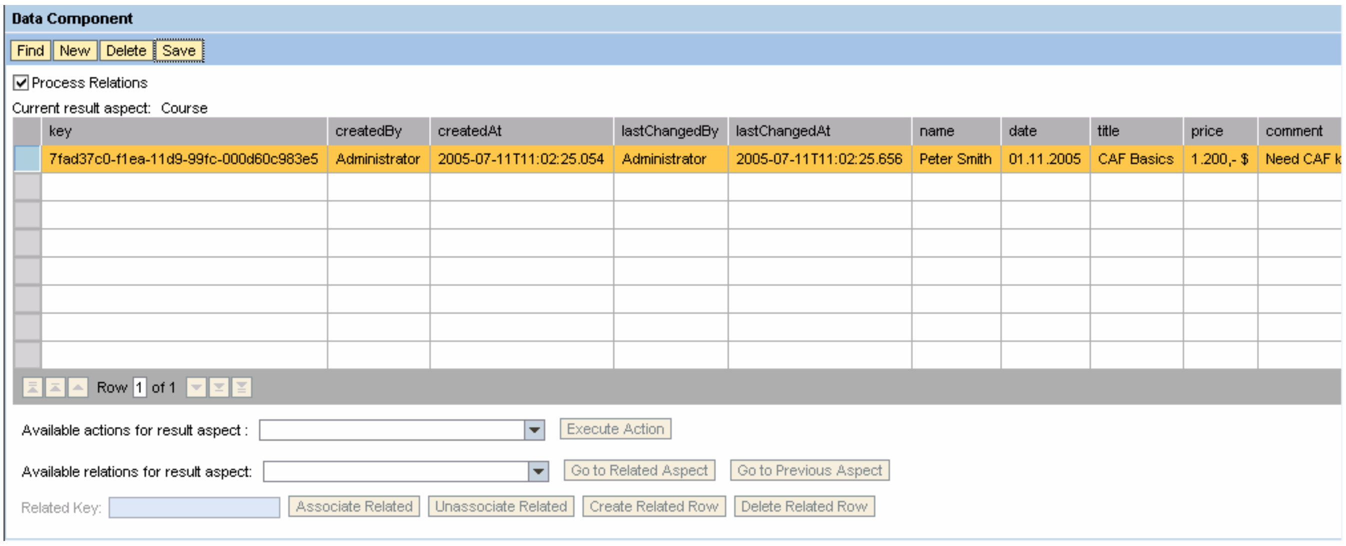
- sap.com
  - caf.core
  - caf.tc
  - xperson
  - xteched
    - CourseService
      - Course
        - findByMultipleParameters
        - findByTitle
      - operation_input

Figure 10

9. To create test input data, under *Data Component*, choose *New*.

   Enter some data in the first row of the table in the columns *name*, *date*, *title*, *price*, and *comment* – for example, **`Peter Smith`**, **`01.11.2005`**, `CAF Basics`, **`1.200,- $`**, **`Need CAF know-how`**. Choose *Save*.

   The entity is saved in the database by the CAF persistence layer, and automatically generates the entries for all other table columns (figure 11).



Data Component

Find | New | Delete | Save

☑ Process Relations

Current result aspect: Course

	key	createdBy	createdAt	lastChangedBy	lastChangedAt	name	date	title	price	comment
	7fad37c0-f1ea-11d9-99fc-000d60c983e5	Administrator	2005-07-11T11:02:25.054	Administrator	2005-07-11T11:02:25.656	Peter Smith	01.11.2005	CAF Basics	1.200,- $	Need CAF k

Row 1 of 1

Available actions for result aspect : Execute Action

Available relations for result aspect: Go to Related Aspect | Go to Previous Aspect

Related Key: Associate Related | Unassociate Related | Create Related Row | Delete Related Row

Figure 11

10. Finally, test the search functionality in the service.

    To do that, in the *Available Services* area of the *Service Browser* select the *findByTitle* operation under *Course*. The title parameter for the findBy operation is now available in the Data Component area. Enter an asterisk (*) and choose *Execute Query*. As a result, the entry created in step 9 appears in the result list.

2.2 Create Application Service "CourseAppService"

2.2.1 Create the Application Service

1. In the Service Explorer view of the CAS perspective in the SAP NetWeaver Developer Studio choose *xteched* → *Application Services.* Open the context menu using the secondary mouse button, and choose *New*.
2. Enter **`CourseAppService`** for the application service's name, and choose *Finish*.
3. In the *Service Explorer* view, a new node appears under *Application Services*. The service editor opens and you can edit different aspects of the application service configuration, as explained in the following sections.

2.2.2 Define Service Dependencies

1. Open the *Dependencies* tabstrip.
2. You can see a list of the available services in the left-hand part of the screen under Service Catalog.

   To define a dependency to the *Course* entity service, select it under *xteched* → *Entity Services*, and choose (*Add…*).

   The entity service appears under *Available Services* (figure 12).

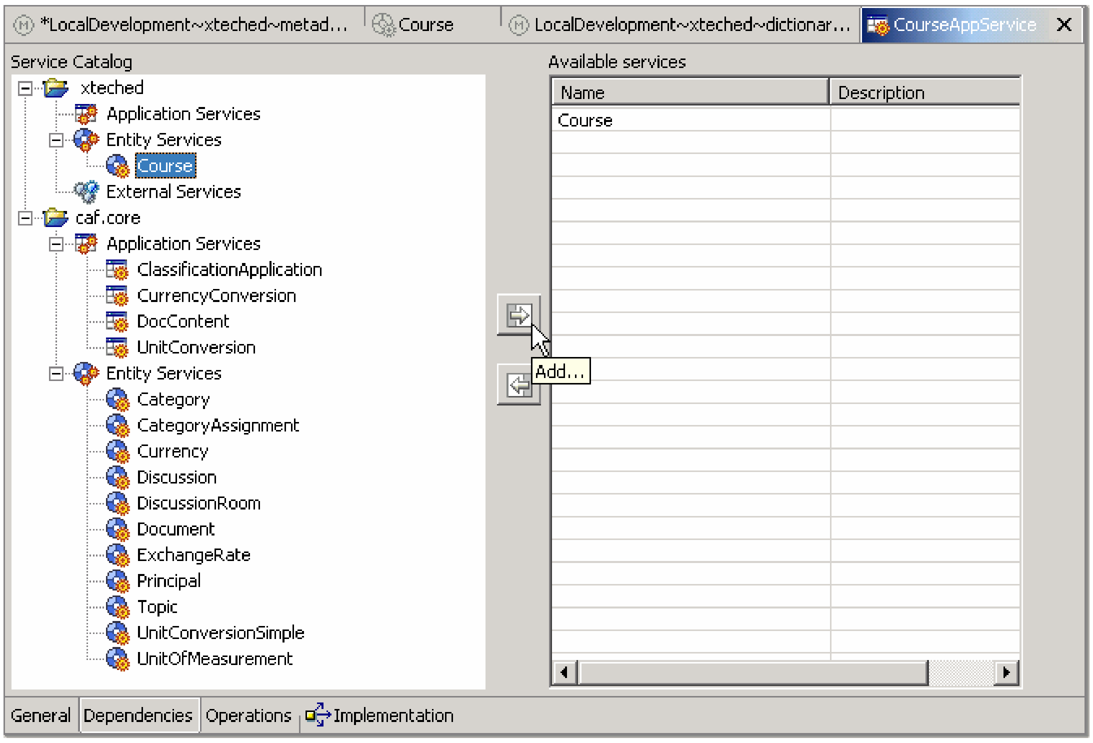


Figure 12

2.2.3 Add an Operation to the Application Service

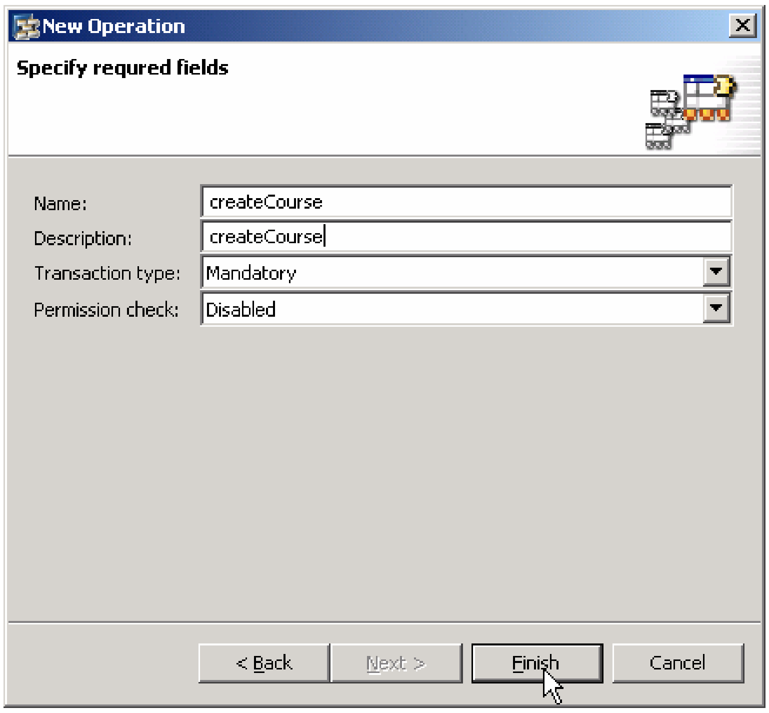
1. Open the *Operations* tabstrip and choose *Add…*.

   You can add any predefined lifecycle methods, as well as custom methods.

   For this example, you need a `CREATE` operation that returns the key of the newly created entity.
2. In the first screen of the *New Operation* wizard, leave the *Custom* option selected as it is by default, and choose *Next*.

   The predefined method Create is not appropriate in this case, as it returns the entity instance itself and not its key.

3. Enter **`createCourse`** for the operation name anddescription for the operation (figure 13).
4. Choose *Finish*.



New Operation

Specify requred fields

Name: createCourse

Description: createCourse

Transaction type: Mandatory

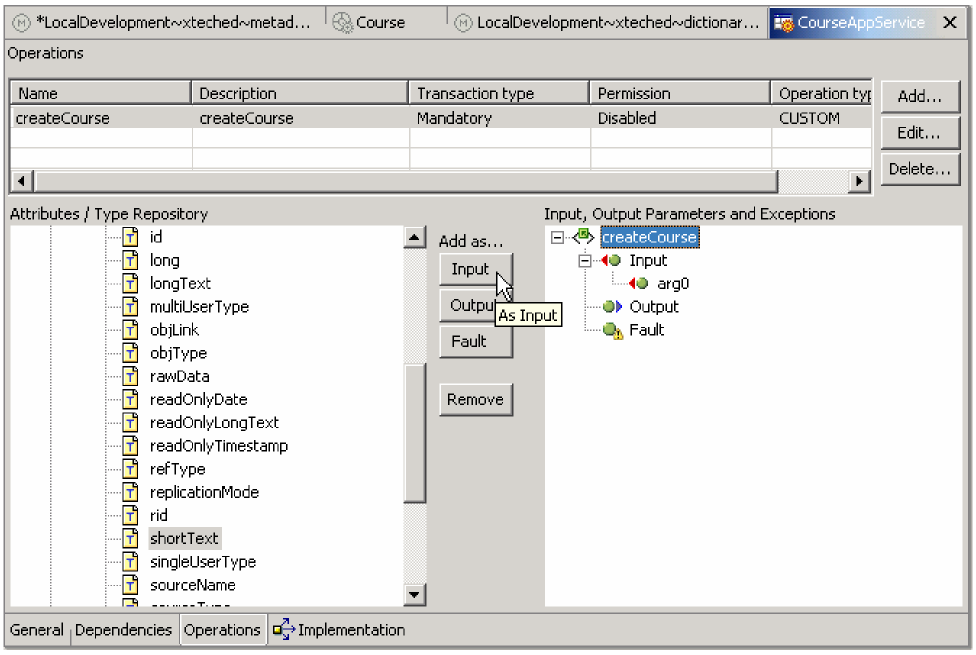
Permission check: Disabled

Figure 13

5. Provide input parameters for the operation.
   a. Select the `createCourse` entry in the *Operations* list.

      Using the buttons Input, Output, Fault, and Remove, you can define parameters and exceptions for the application service operation.
   b. To add a new input parameter, select *Catalog* → *Simple Types* → *com.sap.caf.core* → *shortText*, and choose *Input.*

      The input parameter is added in the right-hand area with the default name `arg0`.



*LocalDevelopment~xteched~metad... | Course | LocalDevelopment~xteched~dictionar... | CourseAppService

Operations

Name	Description	Transaction type	Permission	Operation typ
createCourse	createCourse	Mandatory	Disabled	CUSTOM

Attributes / Type Repository

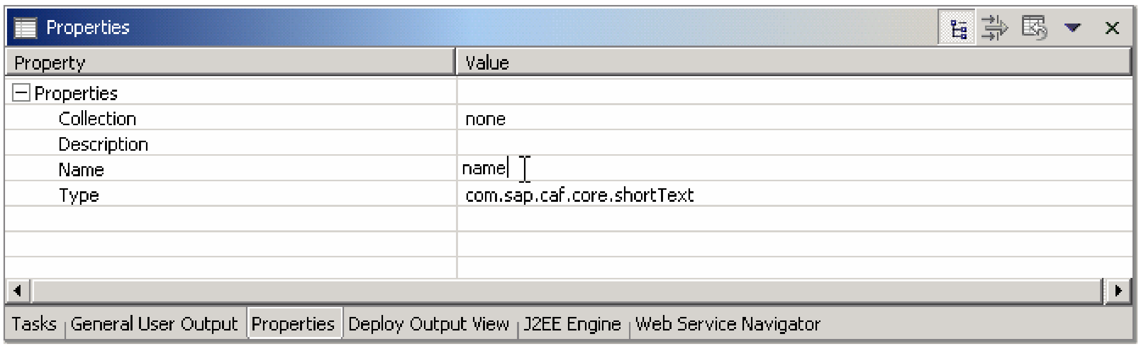
id, long, longText, multiUserType, objLink, objType, rawData, readOnlyDate, readOnlyLongText, readOnlyTimestamp, refType, replicationMode, rid, shortText, singleUserType, sourceName

Input, Output Parameters and Exceptions

createCourse: Input (arg0), Output, Fault

Figure 14

c. To rename the input parameter, select `arg0` and open the *Properties* tabstrip. Overwrite the default name value in the *Value* column (figure 15).



Properties

Property	Value
Properties	
Collection	none
Description	
Name	name
Type	com.sap.caf.core.shortText

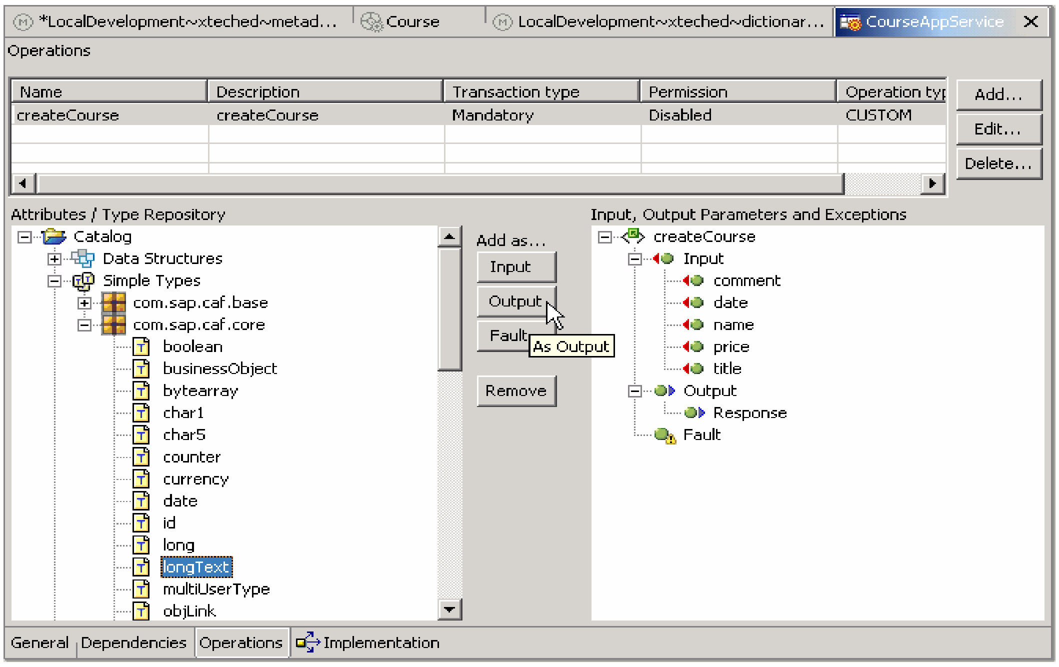
Figure 15

d. Repeat steps 2 and 3 to create the parameters **`date`**, **`title`**, **`price`** (all of type **`shortText`**), and **`comment`** (of type **`longText`**).

2. Provide an output parameter for the operation.

The createCourse operation must return the entity instance key. In CAF the entity instance keys are string values.

To add an output string parameter, select Catalog → *Simple Types* → *com.sap.caf.core* → *longText* and choose *Output*. A new parameter `Response` is added to the structure on the right-hand side (figure 16).



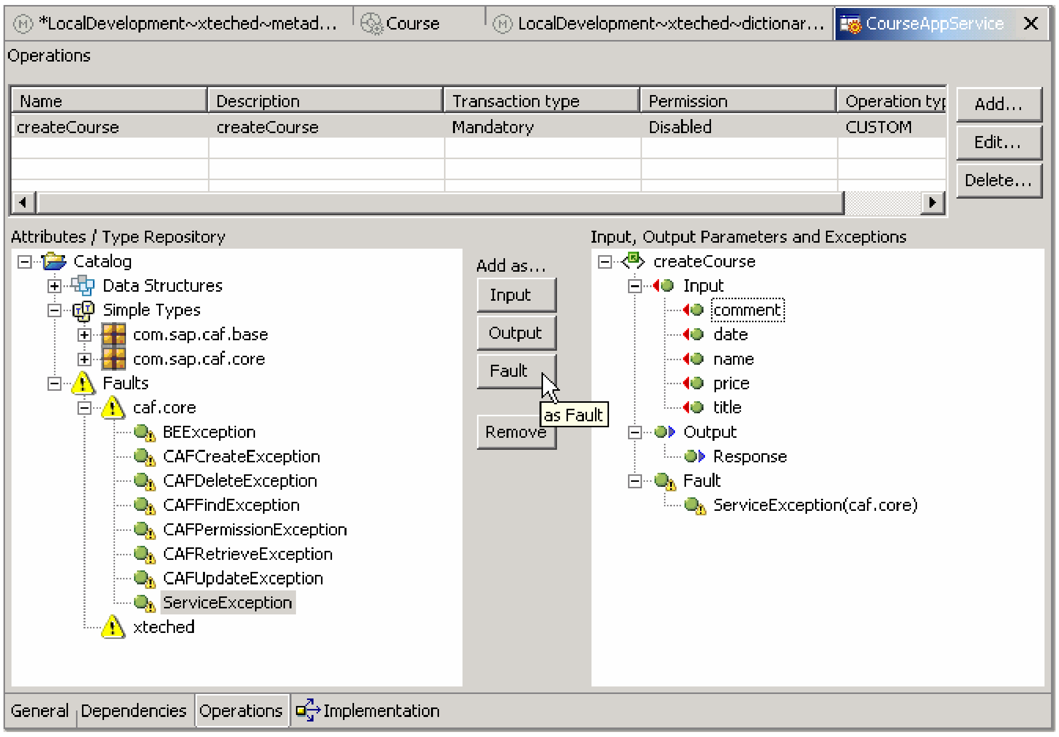
Name	Description	Transaction type	Permission	Operation typ
createCourse	createCourse	Mandatory	Disabled	CUSTOM

Figure 16

2. Add an exception for the operation.

   Exception notify the service caller of application service errors. You can define different types of exceptional cases.

   To add an exception, select *Catalog* → *Faults* → *caf.core* → *ServiceException,* and choose *Fault*. A new exception is added in the structure on the right-hand side (figure 17).



Name	Description	Transaction type	Permission	Operation typ
createCourse	createCourse	Mandatory	Disabled	CUSTOM

Figure 17

3. Save the changes using (*Save All Metadata*).

2.2.4 Implement the Operation

1. Open the *Implementation* tabstrip of the application service editor.

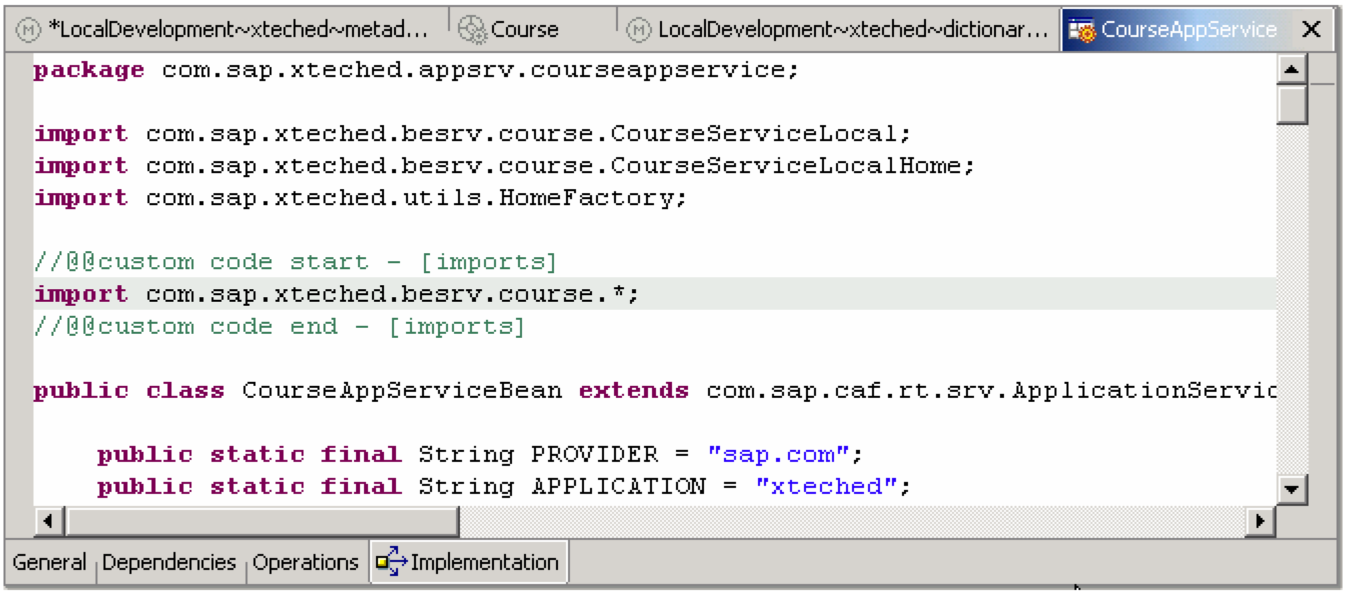
   The generated Java code of the application service is displayed.

   In contrast to entity services, here you can modify the source code. However, it is important to know that you are only allowed to place custom Java code in designated sections, which are marked with special Java comments. You can write your own code between the `//@@custom code start` and `//@@custom code end` comments. Such comments are also provided for custom import and parameter declarations.

2. Add an import statement to the code.

   You can do this between the `//@@custom code start – [imports]` and `//@@custom code end – [imports]` lines. Add the following import statement:

   `import com.sap.xteched.besrv.course.*;`



*LocalDevelopment~xteched~metad... | Course | LocalDevelopment~xteched~dictionar... | CourseAppService

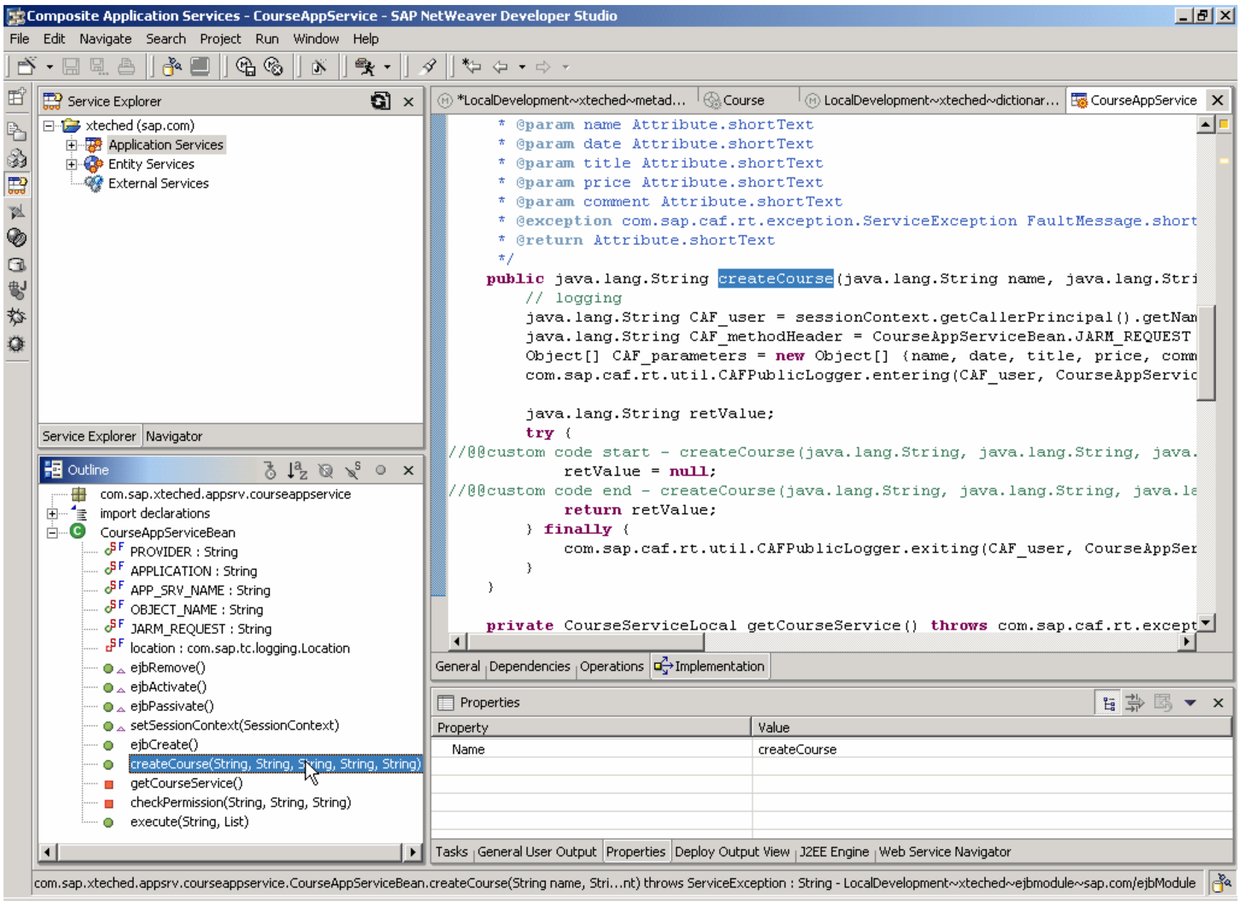
```
package com.sap.xteched.appsrv.courseappservice;

import com.sap.xteched.besrv.course.CourseServiceLocal;
import com.sap.xteched.besrv.course.CourseServiceLocalHome;
import com.sap.xteched.utils.HomeFactory;

//@@custom code start - [imports]
import com.sap.xteched.besrv.course.*;
//@@custom code end - [imports]

public class CourseAppServiceBean extends com.sap.caf.rt.srv.ApplicationServic

    public static final String PROVIDER = "sap.com";
    public static final String APPLICATION = "xteched";
```

General | Dependencies | Operations | Implementation

Figure 18

3. Implement the createCourse operation.

   The operation implementation is done in the CourseAppServiceBean. You can go to the relevant methor by selecting it in the Outline view of the service (figure 19).



```
com.sap.xteched.appsrv.courseappservice
import declarations
CourseAppServiceBean
    PROVIDER : String
    APPLICATION : String
    APP_SRV_NAME : String
    OBJECT_NAME : String
    JARM_REQUEST : String
    location : com.sap.tc.logging.Location
    ejbRemove()
    ejbActivate()
    ejbPassivate()
    setSessionContext(SessionContext)
    ejbCreate()
    createCourse(String, String, String, String, String)
    getCourseService()
    checkPermission(String, String, String)
    execute(String, List)
```

```
     * @param name Attribute.shortText
     * @param date Attribute.shortText
     * @param title Attribute.shortText
     * @param price Attribute.shortText
     * @param comment Attribute.shortText
     * @exception com.sap.caf.rt.exception.ServiceException FaultMessage.short
     * @return Attribute.shortText
     */
    public java.lang.String createCourse(java.lang.String name, java.lang.Stri
        // logging
        java.lang.String CAF_user = sessionContext.getCallerPrincipal().getNam
        java.lang.String CAF_methodHeader = CourseAppServiceBean.JARM_REQUEST
        Object[] CAF_parameters = new Object[] {name, date, title, price, comm
        com.sap.caf.rt.util.CAFPublicLogger.entering(CAF_user, CourseAppServic

        java.lang.String retValue;
        try {
//@@custom code start - createCourse(java.lang.String, java.lang.String, java.
            retValue = null;
//@@custom code end - createCourse(java.lang.String, java.lang.String, java.la
            return retValue;
        } finally {
            com.sap.caf.rt.util.CAFPublicLogger.exiting(CAF_user, CourseAppSer
        }
    }

    private CourseServiceLocal getCourseService() throws com.sap.caf.rt.except
```

Property	Value
Name	createCourse

```
com.sap.xteched.appsrv.courseappservice.CourseAppServiceBean.createCourse(String name, Stri...nt) throws ServiceException : String - LocalDevelopment~xteched~ejbmodule~sap.com/ejbModule
```

Figure 19

4. Insert the following code between the comments `//@@custom code start – createCourse(…)` and `//@@custom code end – createCourse(…)`:

```
retValue = null;
CourseServiceLocal cs = this.getCourseService();
Course course = cs.create();
course.setName(name);
course.setDate(date);
course.setTitle(title);
course.setPrice(price);
course.setComment(comment);
cs.update(course);
retValue = course.getKey();
```

5. Save the changes using (*Save All Metadata*).
6. Generate the project code, build the project, and deploy it to the Java server, as described in section *Save and Test the Entity*. Make sure that all projects were deployed successfully.
7. To test your application service,
   a. In the *Service Explorer* view, expand *Application Services* and select *CourseAppService*. Open its context menu using the secondary mouse button, and choose *Test*.

b. The *Service Browser* opens in a new window. For testing purposes, you need to choose the relevant operation. Under *Available Services*, select *sap.com* → *xteched* → *CourseAppService* → *createCourse_Response* → *createCourse*.

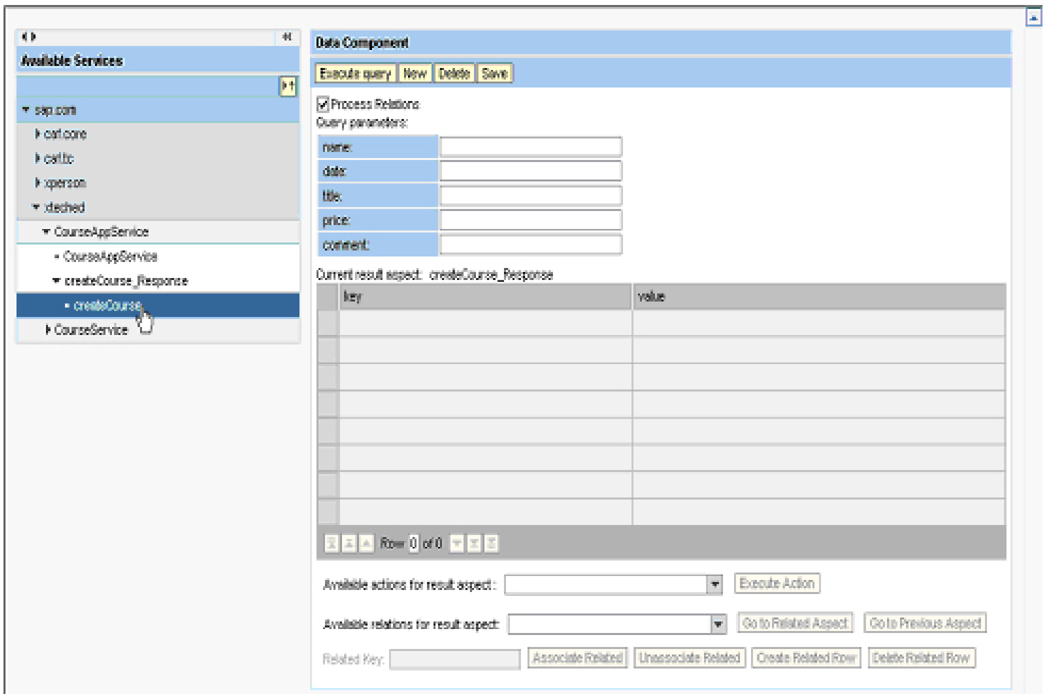


Figure 20

c. Provide test input values – for example, **`Sally Summer`**, **`01.12.2005`**, **`CAF GP`**, **`1.399,- $`**, **`GP is relevant for next project`**, and choose *Execute query*.

As a result, the generated entity instance key is returned.

2.3 Result

You have implemented the services required for the implementation of the Course Request and Approval process. The next step is to integrate the service functions in Guided Procedures using callable objects.

3 Component Layer: Creating the Callable Objects

3.1 Prerequisites

3.1.1 Create Endpoint Aliases

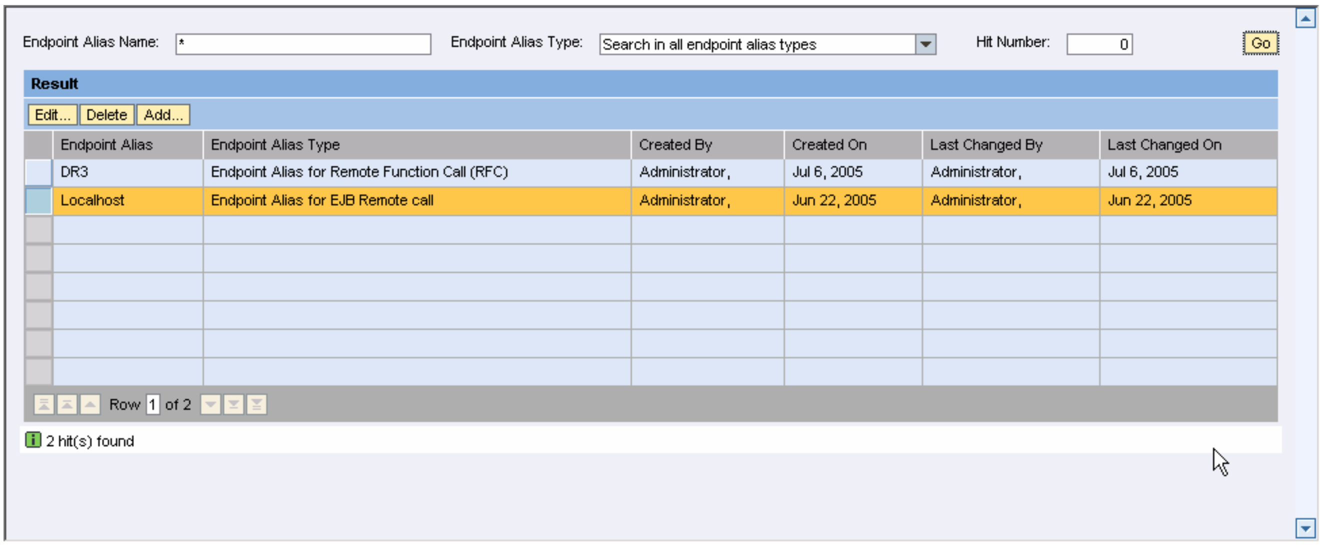
You need to create the following endpoint aliases:

- An endpoint alias for Remote Function Calls (RFC) – this requires access to an SAP system
- An endpoint alias for EJB remote call – this requires access to a Java server

To create an endpoint alias:

1. Launch the Guided Procedures Administration workset and choose *SAP System* → *Configure End Points.*
2. To add a new alias, choose Add.
3. Select the endpoint alias type, and enter the required parameters.
4. Test and save the alias.
5. Repeat steps 2 to 4 to add other aliases.

For more information about creating endpoint aliases, see the Guided Procedures administration documentation at **`http://help.sap.com`**.



Endpoint Alias	Endpoint Alias Type	Created By	Created On	Last Changed By	Last Changed On
DR3	Endpoint Alias for Remote Function Call (RFC)	Administrator,	Jul 6, 2005	Administrator,	Jul 6, 2005
Localhost	Endpoint Alias for EJB Remote call	Administrator,	Jun 22, 2005	Administrator,	Jun 22, 2005

Figure 21

3.1.2 Create Folders for the GP Content

1. Launch the Guided Procedures design time.
2. In the gallery, choose *Create Folder*.
3. Enter a name and a description – for example, *Course Request and Approval*.
4. Choose *Create*.
5. Select the folder that you created, and choose *Create Folder* again.
6. Enter a name and a description – for example, *Callable Objects*. Choose to create the folder as a member of the folder Course Request and Approval.
7. Repeat steps 5 and 6 to create the following folders as well:
    - Actions

- Blocks
- Processes

For more information about the Guided Procedures design time, see the Guided Procedures business expert documentation at **`http://help.sap.com`**.

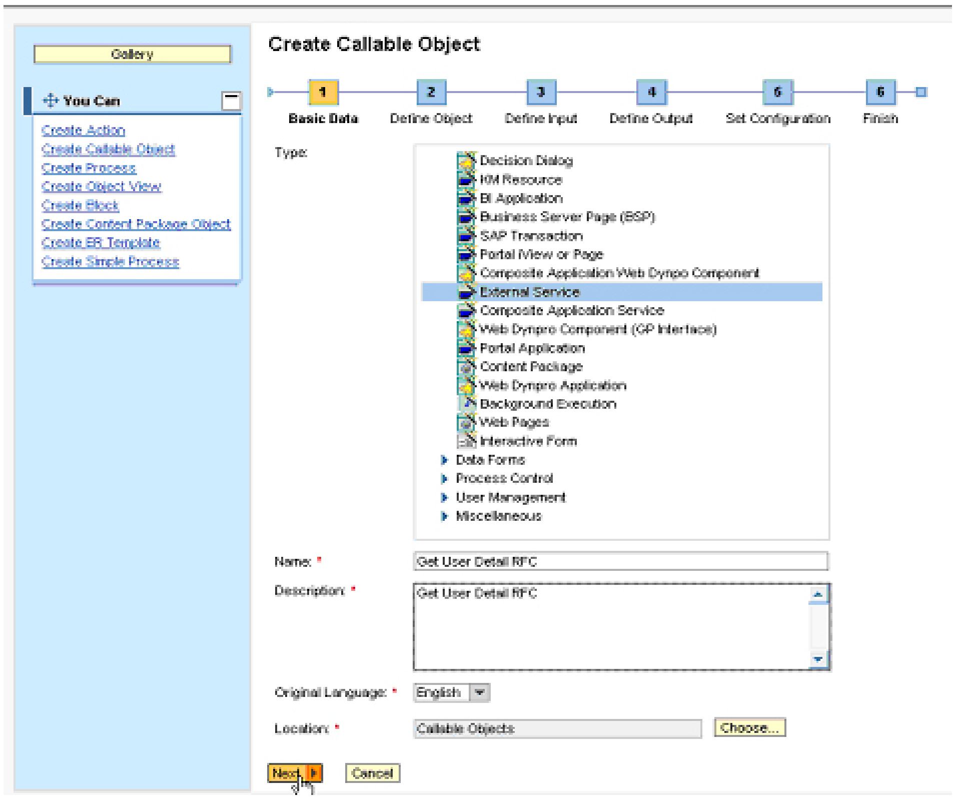
3.2 Create Callable Objects

3.2.1 Create an RFC Callable Object

1. Launch the Guided Procedures design time.
2. Select thCe folder *\Course Request and Approval\Callable Objects*.
3. Choose *Create Callable Object* from the contextual panel (You Can menu).

   A wizard guides you through the steps for creating a callable object.
4. Select *External Service* as the callable object type.

   For name and description, enter *Get User Detail RFC*.



Type: External Service

Name: * Get User Detail RFC

Description: * Get User Detail RFC

Original Language: * English

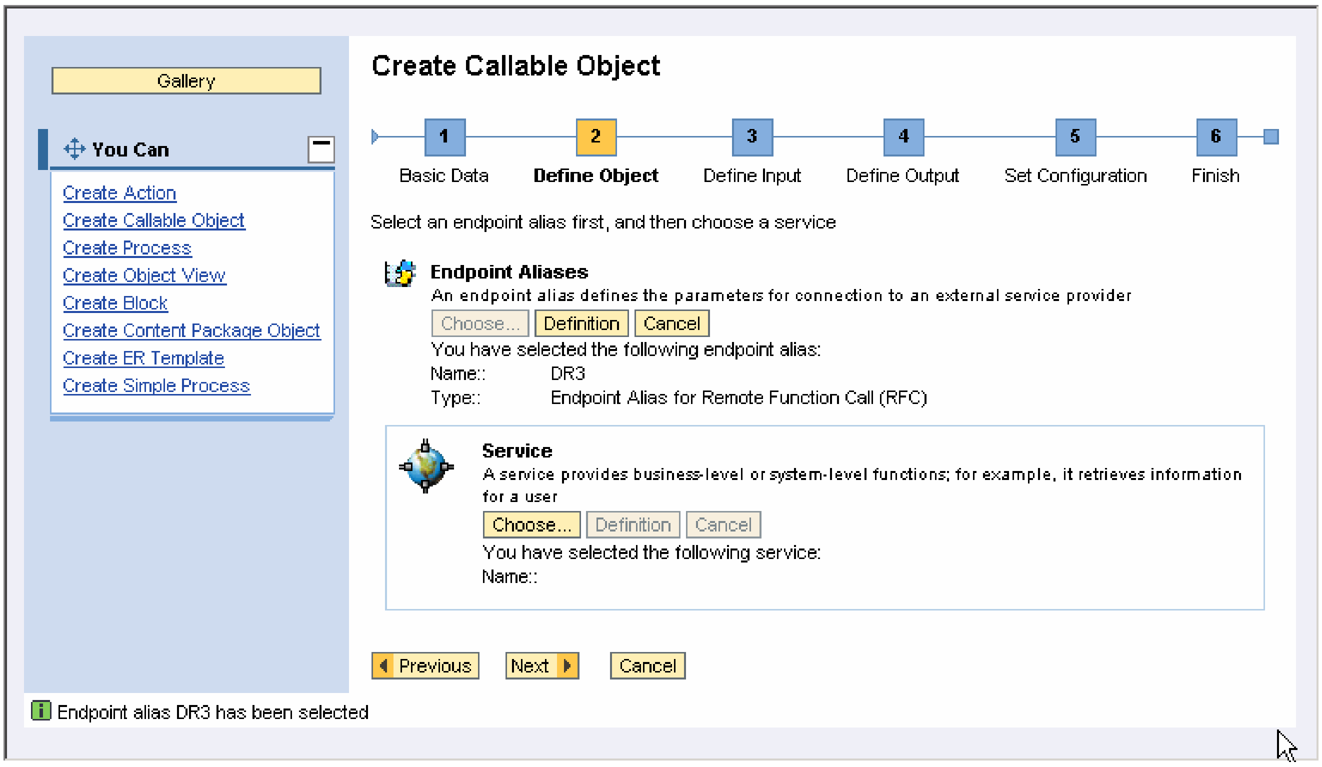
Location: * Callable Objects

Figure 22

5. Choose *Next*.

   Choose the endpoint alias for RFC that you have created as a prerequisite for this part of the example implementation. To do that, use *Choose...*. Select the relevant alias and use *Choose* to confirm.

   Make sure you get a confirmation for the selected endpoint as shown in figure 23.



Create Callable Object

Select an endpoint alias first, and then choose a service

Endpoint Aliases

An endpoint alias defines the parameters for connection to an external service provider

You have selected the following endpoint alias:

Name:: DR3

Type:: Endpoint Alias for Remote Function Call (RFC)

Service

A service provides business-level or system-level functions; for example, it retrieves information for a user

You have selected the following service:

Name::

Endpoint alias DR3 has been selected

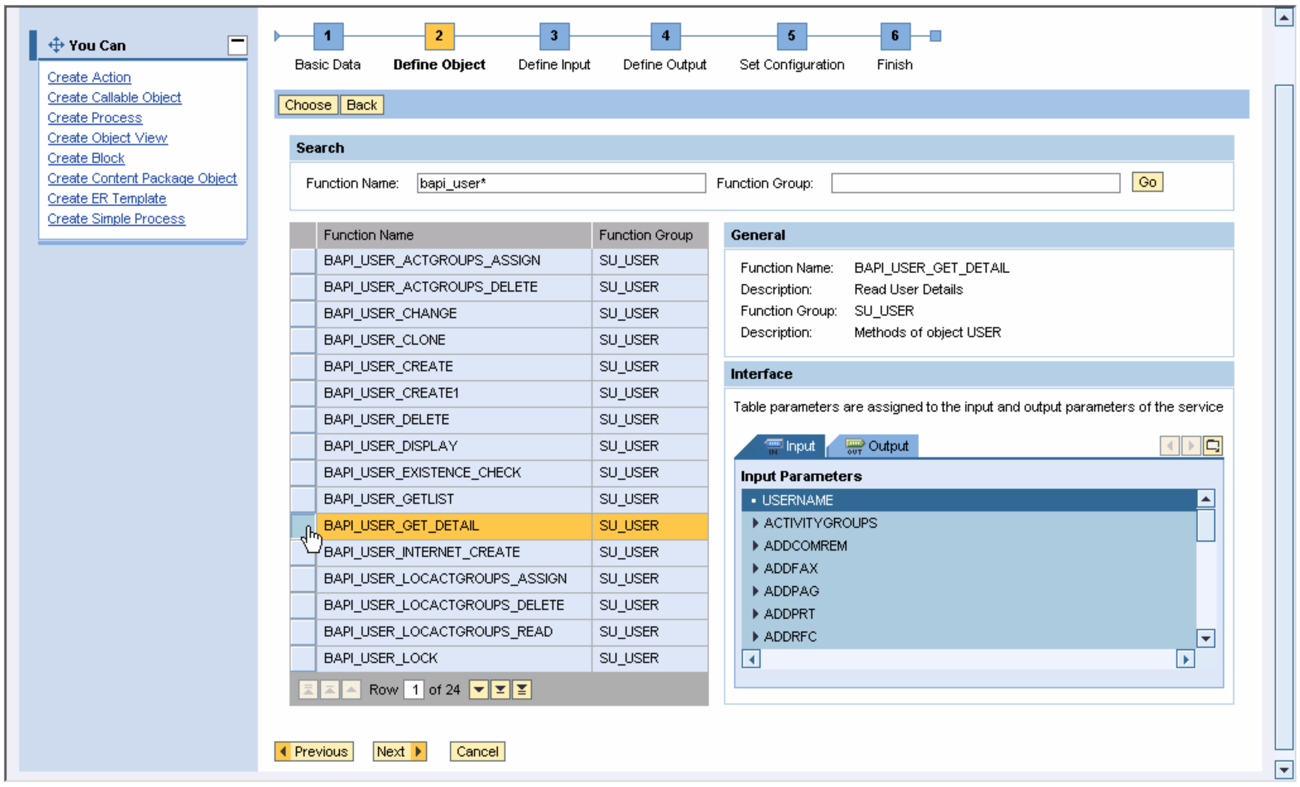
Figure 23

6. Now you can select the relevant service.

   To do that, use *Choose* under *Service*. Search for the required function by entering **`bapi_user*`** for the function name.

   Choose BAPI_USER_GET_DETAIL from the list of available functions, and choose *Next*.

   You have selected the service.



Function Name: bapi_user*

Function Group:

Function Name	Function Group
BAPI_USER_ACTGROUPS_ASSIGN	SU_USER
BAPI_USER_ACTGROUPS_DELETE	SU_USER
BAPI_USER_CHANGE	SU_USER
BAPI_USER_CLONE	SU_USER
BAPI_USER_CREATE	SU_USER
BAPI_USER_CREATE1	SU_USER
BAPI_USER_DELETE	SU_USER
BAPI_USER_DISPLAY	SU_USER
BAPI_USER_EXISTENCE_CHECK	SU_USER
BAPI_USER_GETLIST	SU_USER
BAPI_USER_GET_DETAIL	SU_USER
BAPI_USER_INTERNET_CREATE	SU_USER
BAPI_USER_LOCACTGROUPS_ASSIGN	SU_USER
BAPI_USER_LOCACTGROUPS_DELETE	SU_USER
BAPI_USER_LOCACTGROUPS_READ	SU_USER
BAPI_USER_LOCK	SU_USER

Function Name:	BAPI_USER_GET_DETAIL
Description:	Read User Details
Function Group:	SU_USER
Description:	Methods of object USER

Table parameters are assigned to the input and output parameters of the service

- USERNAME
- ACTIVITYGROUPS
- ADDCOMREM
- ADDFAX
- ADDPAG
- ADDPRT
- ADDRFC

Figure 24

7. Choose *Next*.

   The input parameters are displayed in read-only mode.

8. Choose *Next*.

   The input parameters are displayed in read-only mode.

9. Choose *Next*.

   Define the error handling mode. For simplicity, error handling is not addressed in this example. Therefore, select No Error Handling from the dropdown list.

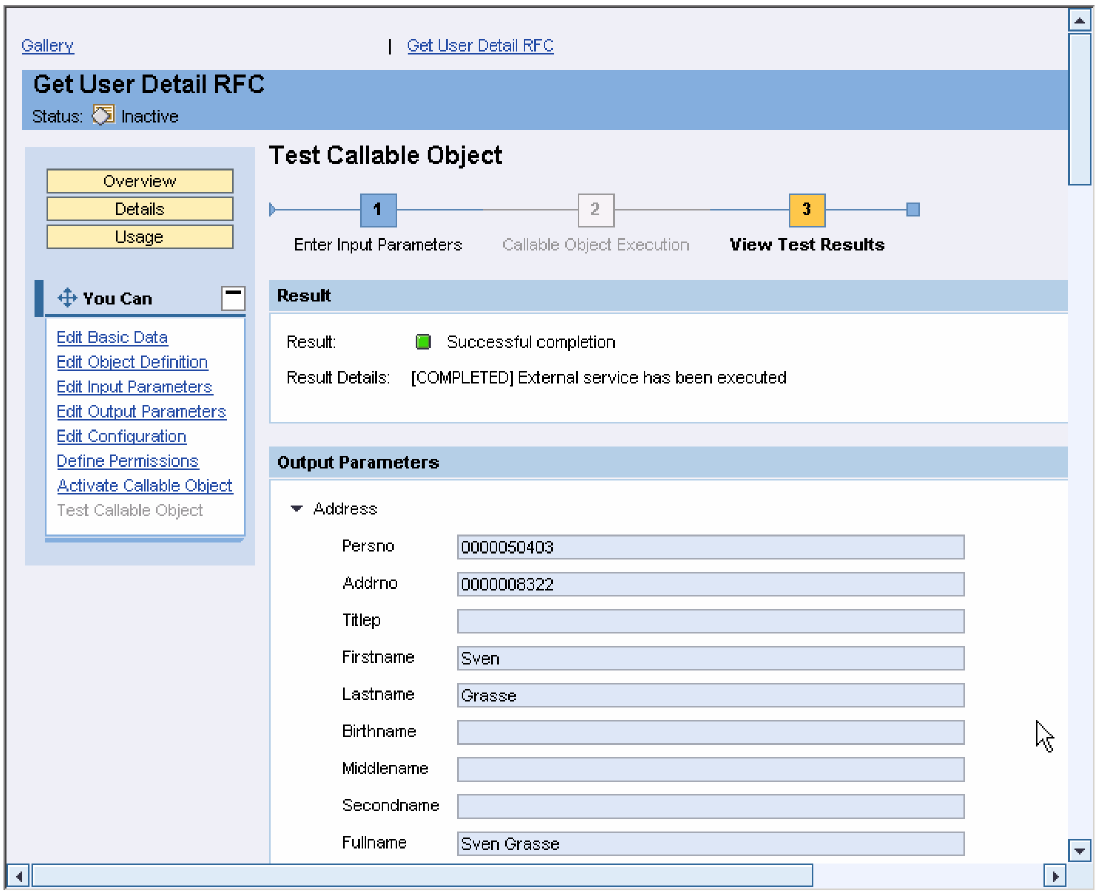
10. To view a summary of the callable object details and to complete the procedure, choose *Next* and then *Finish and Open*.

    The callable object design time opens.

11. To test the object, choose *Test Callable Object* in the contextual panel.

    Enter a user name as an input parameter. You must use a name that exists in the backend system – for example, the name you use to log on to the system.

    Choose *Execute*. If the test has been successful, choose *Close*.



Gallery | Get User Detail RFC

Get User Detail RFC
Status: Inactive

Test Callable Object

Result

Result: 🟩 Successful completion

Result Details: [COMPLETED] External service has been executed

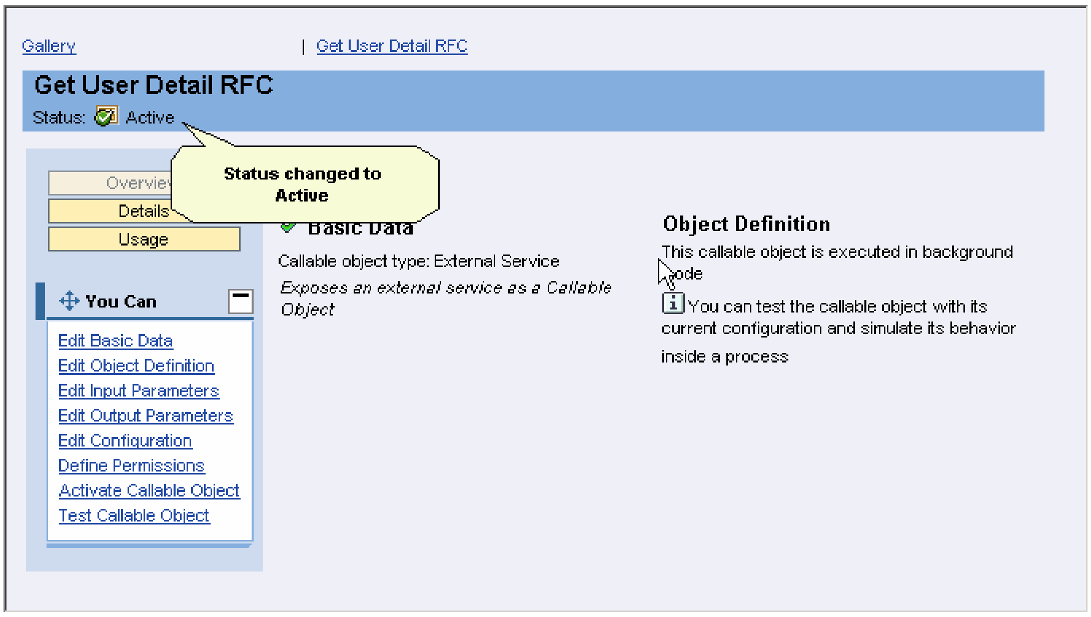
Output Parameters

Address

Persno	0000050403
Addrno	0000008322
Titlep	
Firstname	Sven
Lastname	Grasse
Birthname	
Middlename	
Secondname	
Fullname	Sven Grasse

Figure 25

12. Choose *Activate Callable Object* from the contextual panel, and confirm the activation. Make sure the object's status has changed to *Active* (figure 26).



Gallery | Get User Detail RFC

Get User Detail RFC
Status: Active

Basic Data

Callable object type: External Service

Exposes an external service as a Callable Object

Object Definition

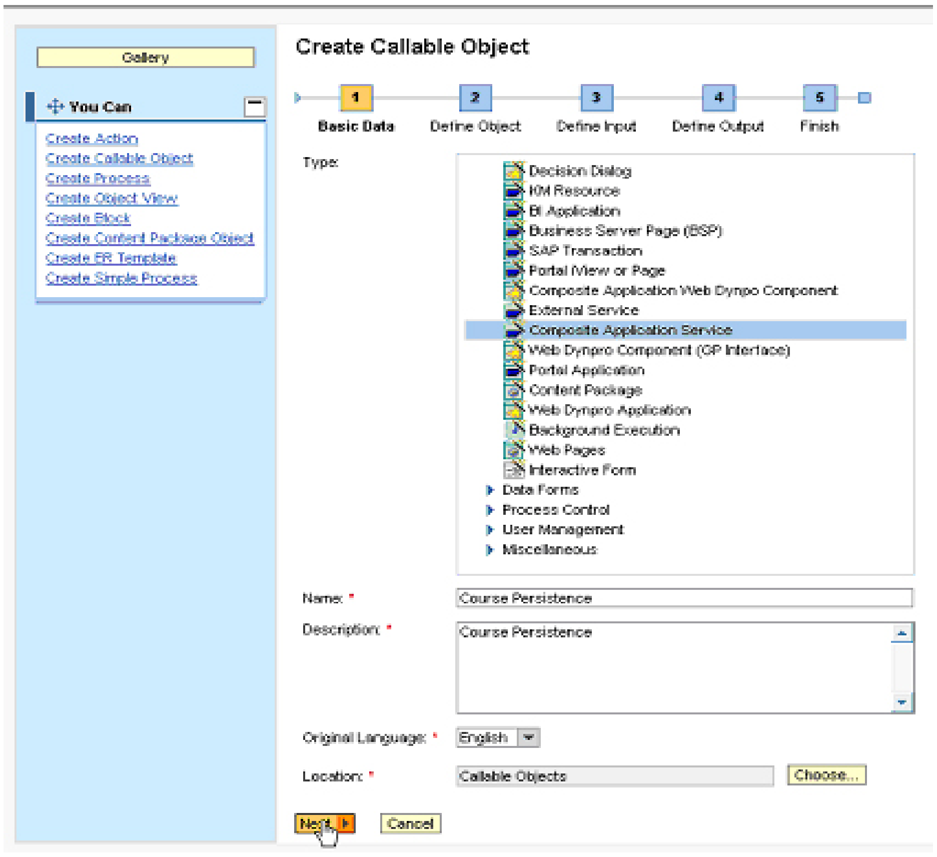
This callable object is executed in background mode

You can test the callable object with its current configuration and simulate its behavior inside a process

Figure 26

3.2.2 Create a Callable Object for the Application Service

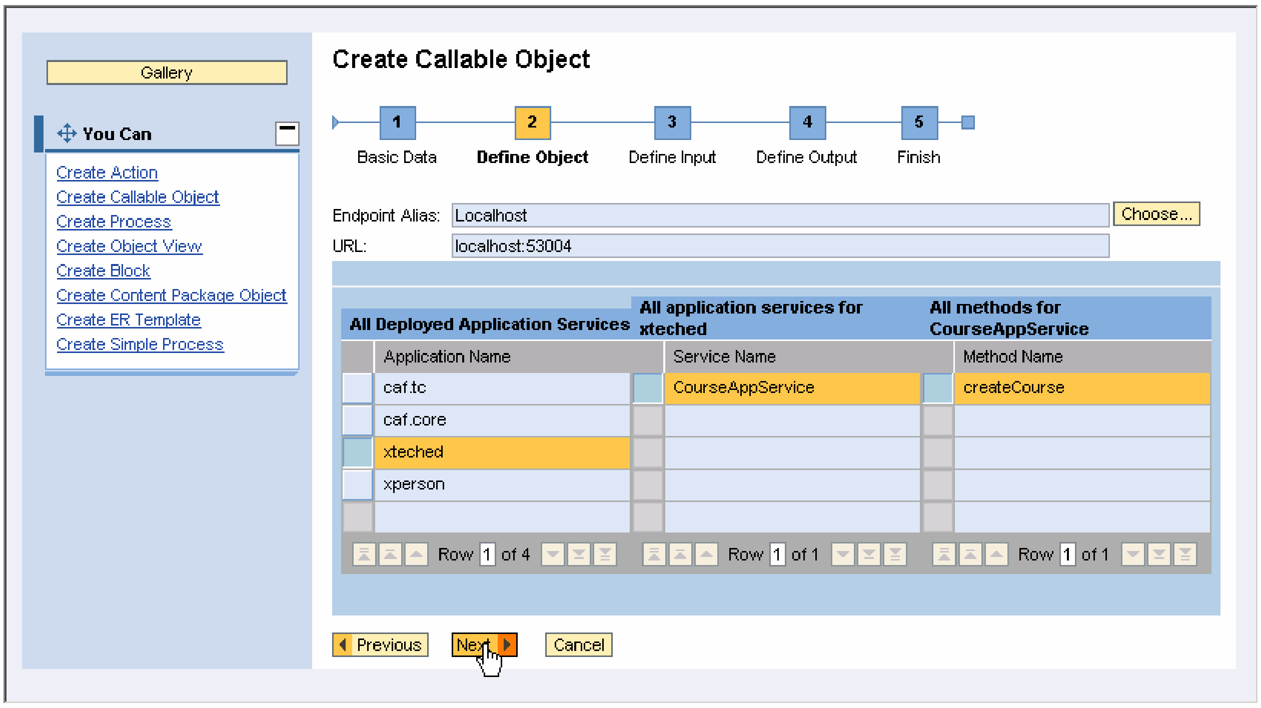
1. To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder \Course Request and Approval\\Callable Objects.
3. Choose *Create Callable Object* from the contextual panel.
   A wizard guides you through the steps for creating a callable object.
4. Select *Composite Application Service* as the callable object type.
   For name and description, enter *Course Persistence*.



Name: * Course Persistence
Description: * Course Persistence
Original Language: * English
Location: * Callable Objects

Figure 27

5. Choose *Next*.
   Choose the endpoint alias for EJB remote calls that you have created as a prerequisite for this part of the example implementation. To do that, use *Choose...*. Select the relevant alias and use *Choose* to confirm.
6. Now you can select the relevant service.
   In table *All Deployed Application Services*, select the *xteched* application. The `CourseAppService` application service appears under *Service Name*. Select it and then select *createCourse* under *Method Name*.



Create Callable Object

Endpoint Alias: Localhost

URL: localhost:53004

All Deployed Application Services	All application services for xteched	All methods for CourseAppService
Application Name	Service Name	Method Name
caf.tc	CourseAppService	createCourse
caf.core		
xteched		
xperson		
Row 1 of 4	Row 1 of 1	Row 1 of 1

Figure 28

7. Choose *Next*.

   The input parameters are displayed in read-only mode.

8. Choose *Next*.

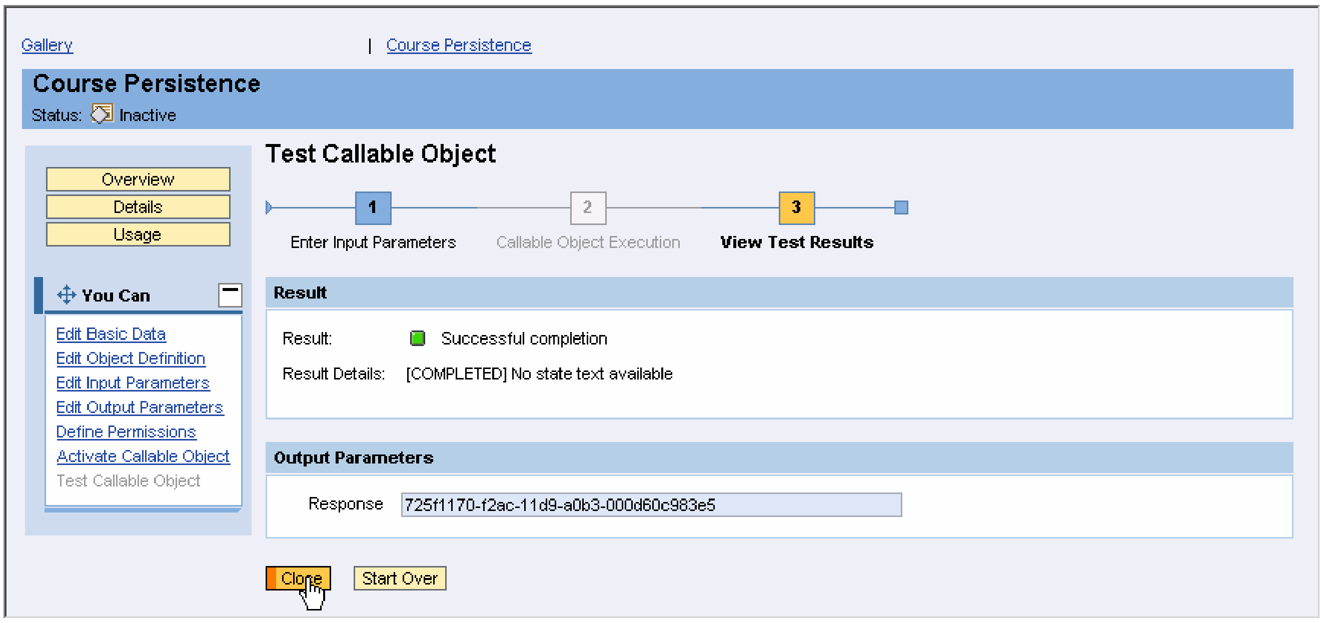
   The input parameters are displayed in read-only mode.

9. To view a summary of the callable object details and to complete the procedure, choose *Next* and then *Finish and Open*.

   The callable object design time opens.

10. To test the object, choose *Test Callable Object* in the contextual panel.

    Enter input data and choose *Execute*. The output is an entity instance key. To finalize the test, choose *Close*.


Gallery | Course Persistence

Course Persistence

Status: Inactive

Test Callable Object

Result

Result: Successful completion

Result Details: [COMPLETED] No state text available

Output Parameters

Response 725f1170-f2ac-11d9-a0b3-000d60c983e5

Figure 29

11. Choose *Activate Callable Object* from the contextual panel, and confirm the activation. Make sure the object's status has changed to *Active*.

3.2.3 Create a Callable Object for the Data Input Form

1. To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder *\Course Request and Approval\Callable Objects.*
3. Choose *Create Callable Object* from the contextual panel.

   A wizard guides you through the steps for creating a callable object.
4. Select *Data Forms* → *Input* as the callable object type.

   For name and description, enter *Course Input*.

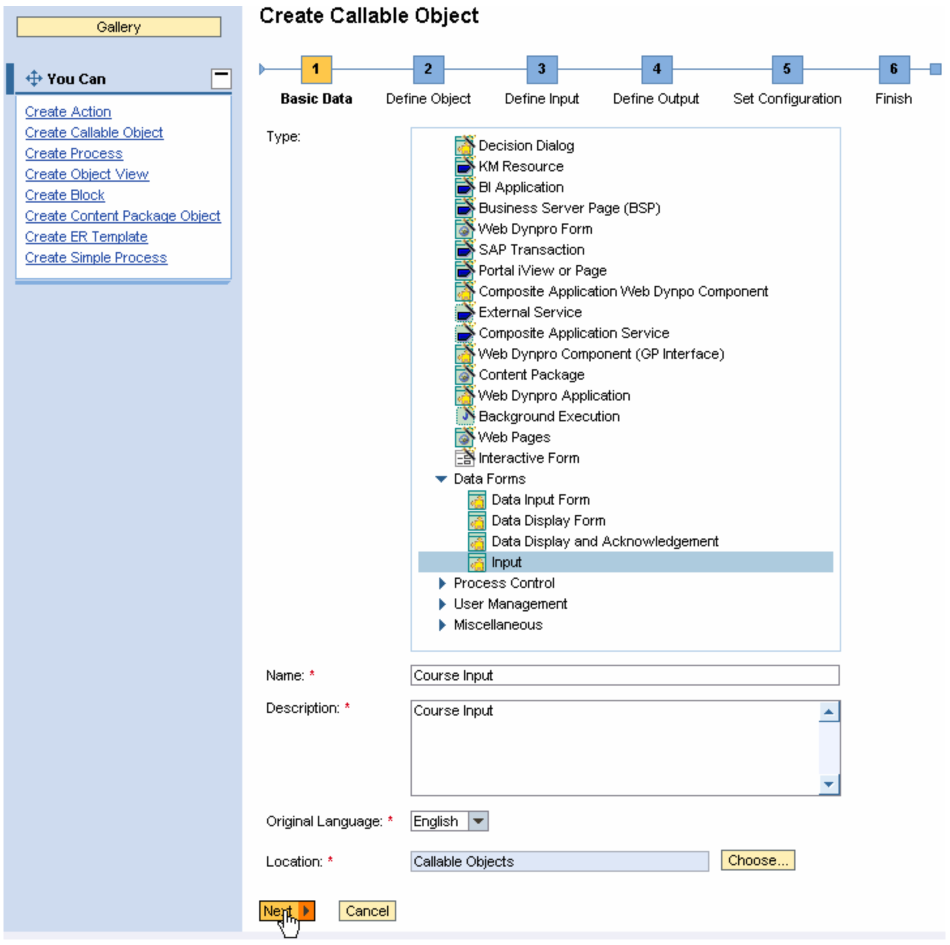


Figure 30

5. Choose *Next*.

   The next screen displays information about the Web Dynpro component that is used for the input form.

6. Choose *Next*.

   You must define input parameters for the data that the course requestor must enter. For that purpose, you create the following parameters of type String:

   - Name
   - Title
   - Price
   - Date
   - Comment

   To define the parameters:

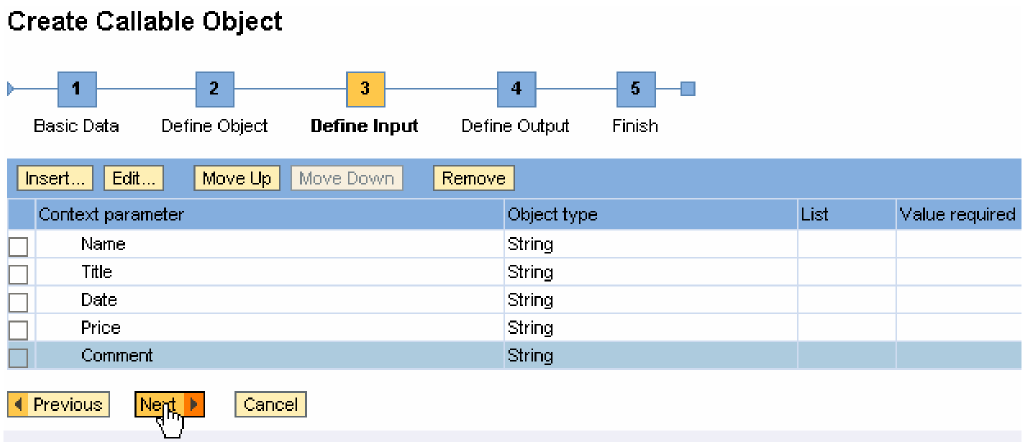
   a. Choose *Insert…*.

   b. Enter a technical name. You are only allowed to use upper- and lower-case letters from the Latin alphabet, underscore, and numbers.

   c. Enter an arbitrary name. It is used for display purposes.

   d. Choose the parameter type, and define this is a required parameter.

   e. Choose *Insert*.


Create Callable Object

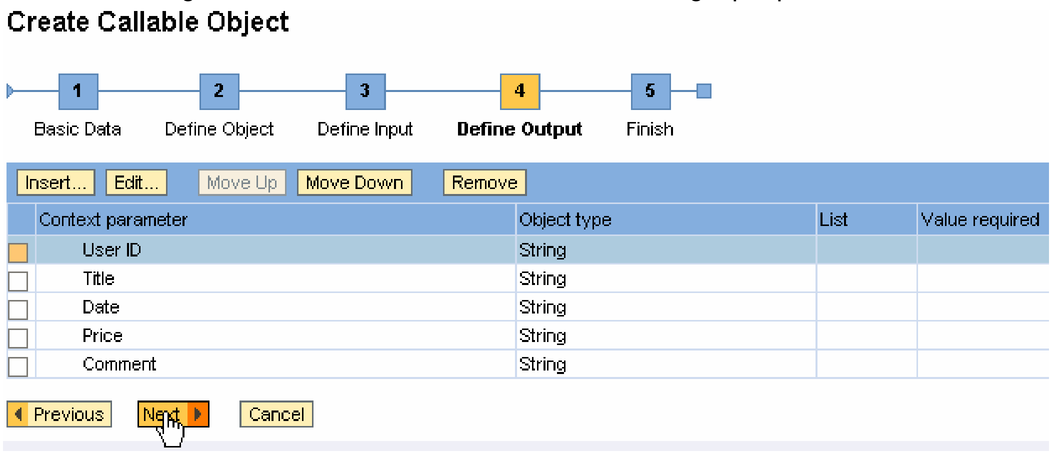
Context parameter	Object type	List	Value required
Name	String		
Title	String		
Date	String		
Price	String		
Comment	String		

Figure 31

7. Choose *Next*.

   You must define the following output parameters of type String:
   - User ID
   - Title
   - Date
   - Price
   - Comment

   The procedure for defining them is similar to the one used for defining input parameters.


Create Callable Object

Context parameter	Object type	List	Value required
User ID	String		
Title	String		
Date	String		
Price	String		
Comment	String		

Figure 32

8. To view a summary of the callable object details and to complete the procedure, choose *Next* and then *Finish and Open*.

   The callable object design time opens.
9. You may skip the testing step, as this is a very simple object. Choose *Activate Callable Object* from the contextual panel, and confirm the activation. Make sure the object's status has changed to *Active*.

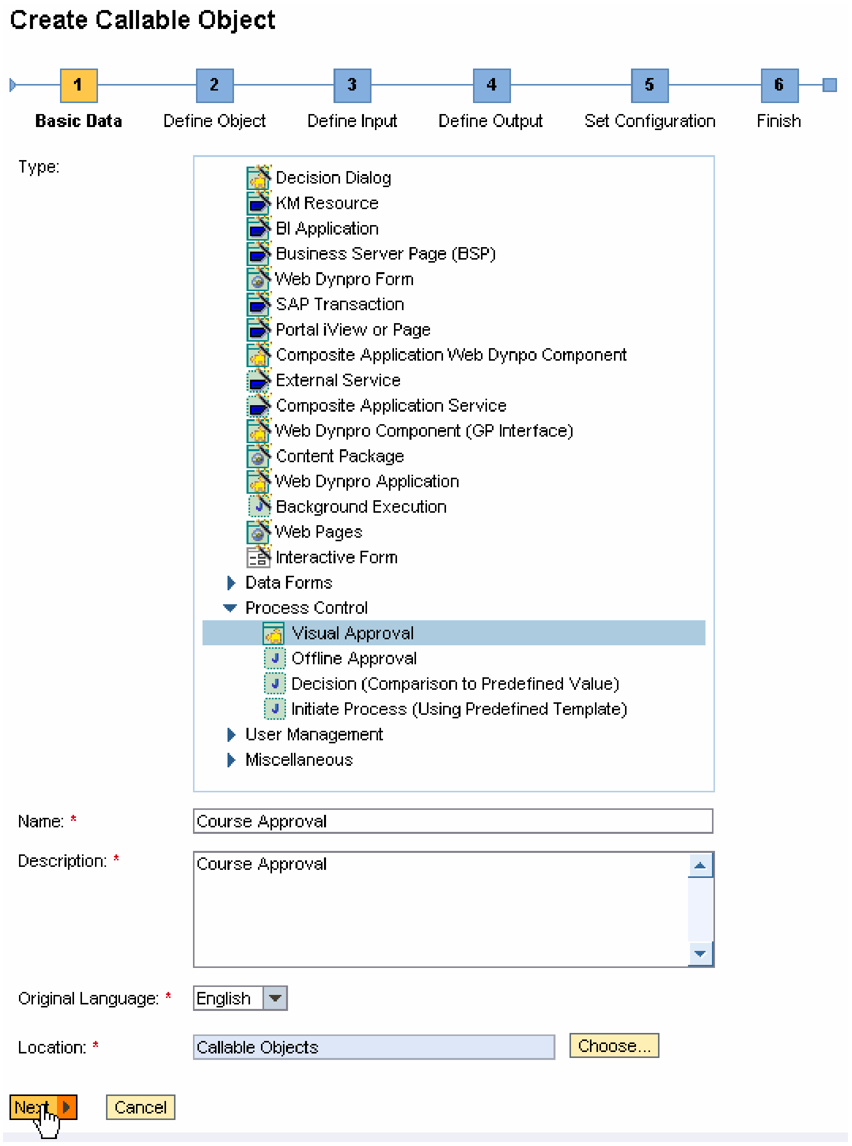
3.2.4 Create a Callable Object for the Approval Step

1. To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder *\Course Request and Approval\Callable Objects.*
3. Choose *Create Callable Object* from the contextual panel.

A wizard guides you through the steps for creating a callable object.

4. Select *Process Control* → *Visual Approval* as the callable object type.

   For name and description, enter *Course Approval*.



Create Callable Object

1 Basic Data — 2 Define Object — 3 Define Input — 4 Define Output — 5 Set Configuration — 6 Finish

Type:
- Decision Dialog
- KM Resource
- BI Application
- Business Server Page (BSP)
- Web Dynpro Form
- SAP Transaction
- Portal iView or Page
- Composite Application Web Dynpo Component
- External Service
- Composite Application Service
- Web Dynpro Component (GP Interface)
- Content Package
- Web Dynpro Application
- Background Execution
- Web Pages
- Interactive Form
- Data Forms
- Process Control
  - Visual Approval
  - Offline Approval
  - Decision (Comparison to Predefined Value)
  - Initiate Process (Using Predefined Template)
- User Management
- Miscellaneous

Name: *	Course Approval
Description: *	Course Approval
Original Language: *	English
Location: *	Callable Objects

Figure 33

5. Choose *Next*.

   The next screen displays information about the Web Dynpro component that is used for the input form.

6. Choose *Next*.

   The Course Approval process step must display the entries made using the data input form, also complemented by the requestor's name retrieved using the RFC callable object. Therefore, you must define the following input fields of type String:

   - Name
   - Title
   - Date

- Price
- Comment

The procedure for creating the input parameters is the same as described for the data input form.

7. Choose *Next*.

   This callable object type already contains predefined output parameters. You do not need to define additional ones.

8. Choose Next.

   In the Set Configuration screen of the wizard, you can configure the callable object to send an approval or rejection e-mail to certain recipients. In this example, you do not need to set mail recipients.

9. Set Configuration: During this step you could add additional recipients for approval or rejection. As we don't want to send e-mail messages, simply click on **Next**.
10. To view a summary of the callable object details and to complete the procedure, choose *Next* and then *Finish and Open*.

    The callable object design time opens.

11. Choose *Activate Callable Object* from the contextual panel, and confirm the activation. Make sure the object's status has changed to *Active*.

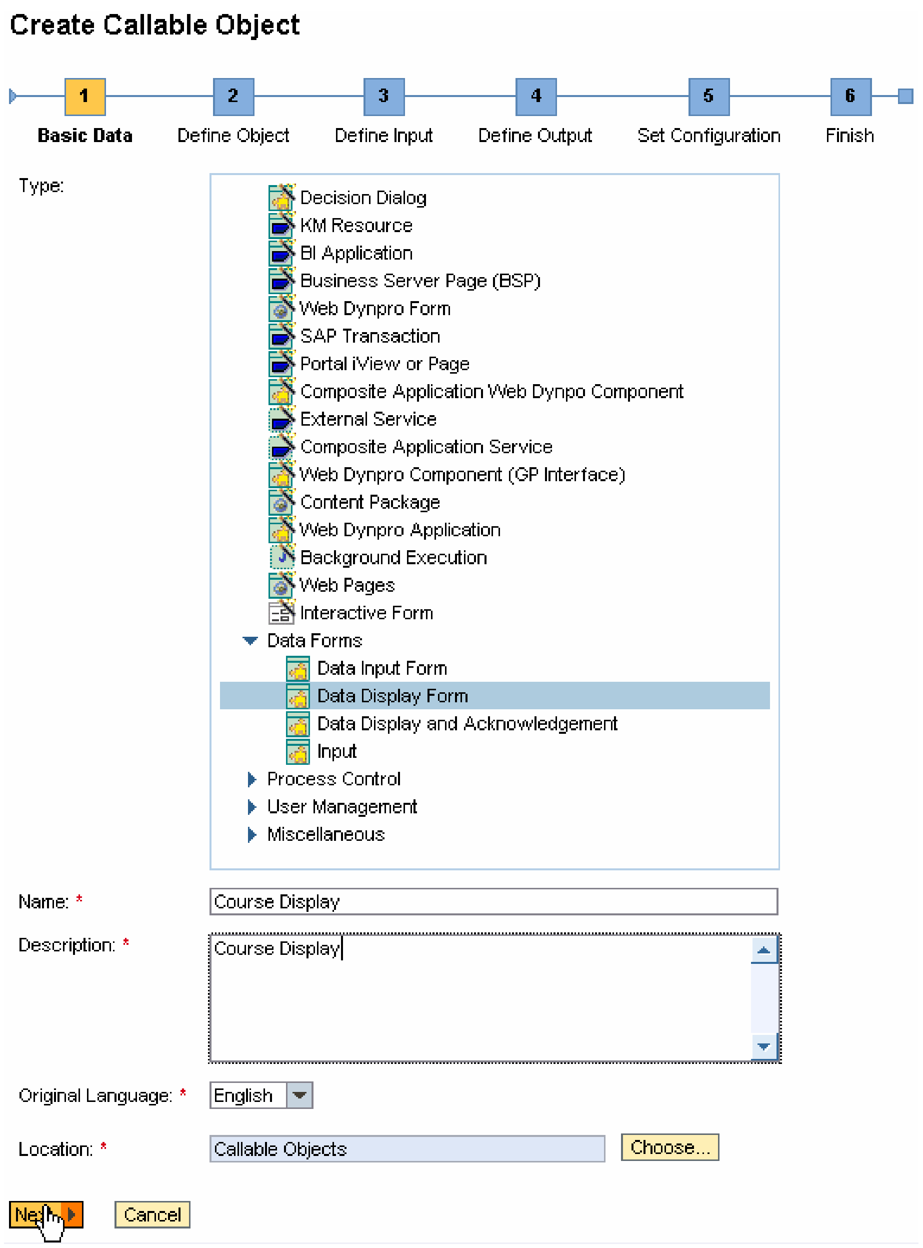
3.2.5 Create a Callable Object for the Data Display Form

1. I To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder *\Course Request and Approval\Callable Objects.*
3. Choose *Create Callable Object* from the contextual panel.

   A wizard guides you through the steps for creating a callable object.

4. Select *Data Forms* → *Data Display Form* as the callable object type.

   For name and description, enter *Course Display*.



Create Callable Object

Type:

- Decision Dialog
- KM Resource
- BI Application
- Business Server Page (BSP)
- Web Dynpro Form
- SAP Transaction
- Portal iView or Page
- Composite Application Web Dynpo Component
- External Service
- Composite Application Service
- Web Dynpro Component (GP Interface)
- Content Package
- Web Dynpro Application
- Background Execution
- Web Pages
- Interactive Form
- Data Forms
  - Data Input Form
  - Data Display Form
  - Data Display and Acknowledgement
  - Input
- Process Control
- User Management
- Miscellaneous

Name: * Course Display

Description: * Course Display

Original Language: * English

Location: * Callable Objects

Figure 34

5. Choose *Next*.

   The next screen displays information about the Web Dynpro component that is used for the input form.

6. Choose *Next*.

   The Course Display process step must display the data gathered in the previous steps. Therefore, you must define the following input parameters of type String:

   - Name
   - Title
   - Price
   - Date
   - Comment

The procedure for creating the input parameters is the same as described for the data input form.

7. To view a summary of the callable object details and to complete the procedure, choose *Next* and then *Finish and Open*.

   The callable object design time opens.

8. Choose *Activate Callable Object* from the contextual panel, and confirm the activation. Make sure the object's status has changed to *Active*.

3.3 Result

Now you have created the callable objects, and you can proceed with the process modelling phase.

4 Process Layer: Modelling the Process

4.1 Create Actions

In this part of the example implementation, you need to create an action for each callable object that you defined in the previous section. Using actions, you can integrate the callable objects into blocks and processes.

To create an action:

1. Open the gallery, and select folder *\Course Request and Approval\Actions.*
2. Choose *Create Action* form the contextual panel.
3. Enter a name and a description for the action.
4. Choose *Save and Open*.

   The action's design time opens.
5. Choose *Attach Callable Objects* from the contextual panel.
6. Browse to select a callable object for the *Object for Execution* field. Choose Save.
7. To activate the action, choose *Activate Action* from the contextual panel.

Use this procedure to create the following actions:

Action Name	**Callable Object for Execution**
Get User Detail RFC	Get User Detail RFC
Course Persistence	Course Persistence
Course Input	Course Input
Course Display	Course Display
Course Approval	Course Approval

4.2 Create a Sequential Block

4.2.1 Create a Block

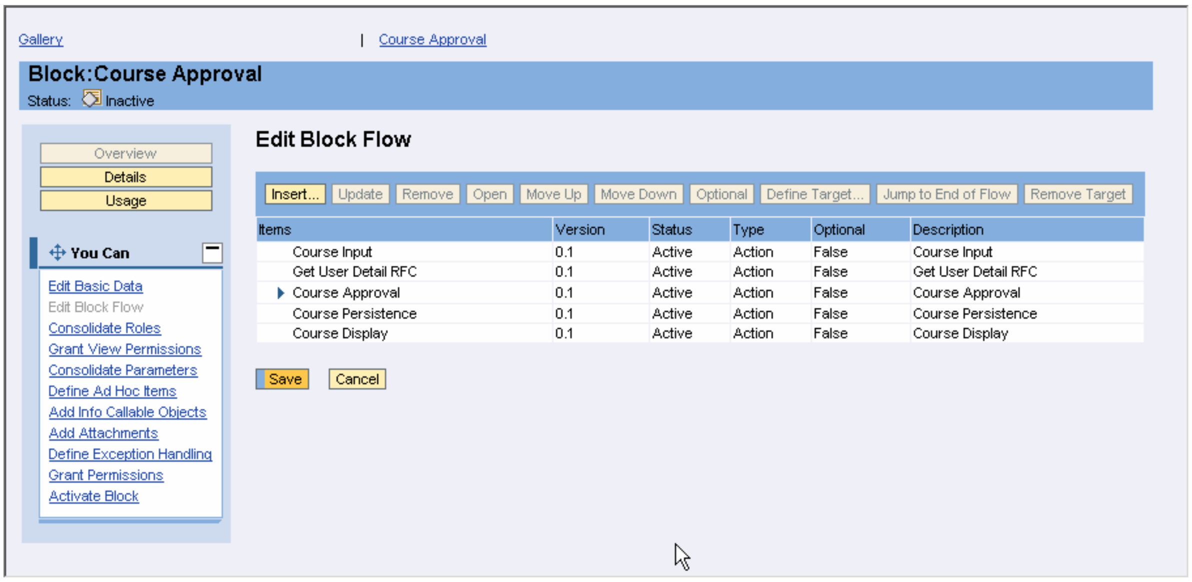
1. To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder *\Course Request and Approval\Blocks.*
3. Choose *Create Block* from the contextual panel.
4. For name and description, enter *Course Approval*. Make sure the selected block type is *Sequential*.
5. Choose *Save and Open*.

   The block's design time opens.

4.2.2 Define the Block Flow

1. Choose *Edit Block Flow* from the contextual panel.
2. To add the actions that you created to the block flow:
   a. Choose *Insert*.
   b. Use *Select…* to insert an existing action.
   c. Navigate to the *\Course Request and Approval\Actions* and select *Course Input*.
   d. Choose *Select*.
   e. Repeat the above steps to add the other actions in the following order:

- Get User Detail RFC
- Course Approval
- Course Persistence
- Course Display



Gallery | Course Approval

Block:Course Approval

Status: Inactive

Edit Block Flow

Items	Version	Status	Type	Optional	Description
Course Input	0.1	Active	Action	False	Course Input
Get User Detail RFC	0.1	Active	Action	False	Get User Detail RFC
Course Approval	0.1	Active	Action	False	Course Approval
Course Persistence	0.1	Active	Action	False	Course Persistence
Course Display	0.1	Active	Action	False	Course Display

Figure 35

3. For action *Course Approval*, you also need to define a target action that is to be executed in case the request is rejected.

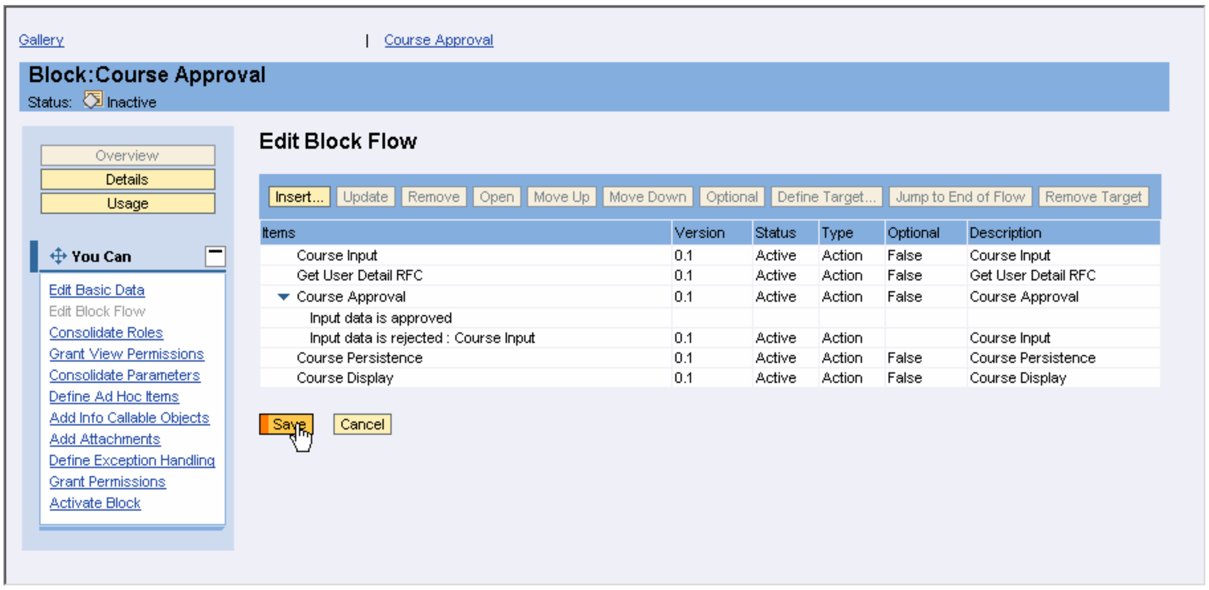
   To do that:

   a. Extend the *Course Approval* node in the block flow, and select the entry *Input data is rejected*.

   b. Choose *Define Target…*

   c. Select *Course Input*, and confirm using *Choose*.

4. Choose *Save*.



Gallery | Course Approval

Block:Course Approval

Status: Inactive

Edit Block Flow

Items	Version	Status	Type	Optional	Description
Course Input	0.1	Active	Action	False	Course Input
Get User Detail RFC	0.1	Active	Action	False	Get User Detail RFC
Course Approval	0.1	Active	Action	False	Course Approval
Input data is approved					
Input data is rejected : Course Input	0.1	Active	Action		Course Input
Course Persistence	0.1	Active	Action	False	Course Persistence
Course Display	0.1	Active	Action	False	Course Display

Figure 36

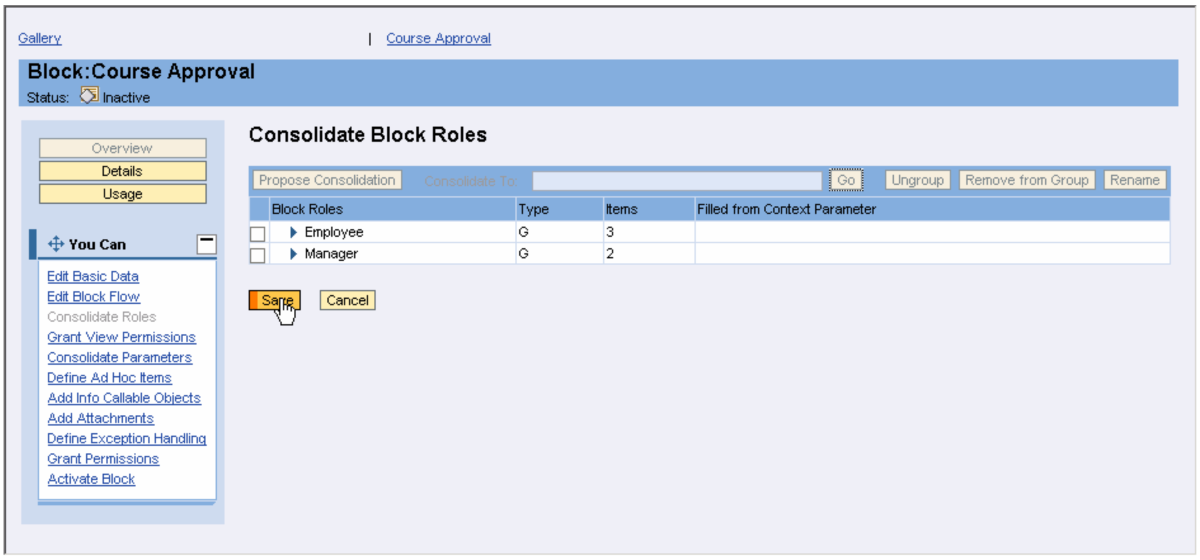
4.2.3 Consolidate Roles

1. Choose *Consolidate Roles* from the contextual panel.
2. Decide who is the processor of each step:
   - Course Input: **Employee**
   - Get User Detail RFC: background step executed on behalf of the **Employee**
   - Course Approval: **Manager**
   - Course Persistence: background step executed on behalf of the **Manager**
   - Course Display: **Employee**

   According to the list, you must consolidate the available roles in two groups – Employee and Manager.
3. Select the following roles:
   - Processor of Course Input
   - Processor of Get User Detail RFC
   - Processor of Course Approval

   Enter *Employee* in the *Consolidate To* field, and choose *Go*.
4. Select the other two roles:
   - Processor of Course Approval
   - Processor of Course Persistence

   Enter *Manager* in the *Consolidate To* field, and choose *Go*.
5. Choose *Save*.



Block:Course Approval

Status: Inactive

Consolidate Block Roles

Block Roles	Type	Items	Filled from Context Parameter
Employee	G	3	
Manager	G	2	

Figure 37

4.2.4 Consolidate Parameters

1. Choose *Consolidate Parameters* from the contextual panel.

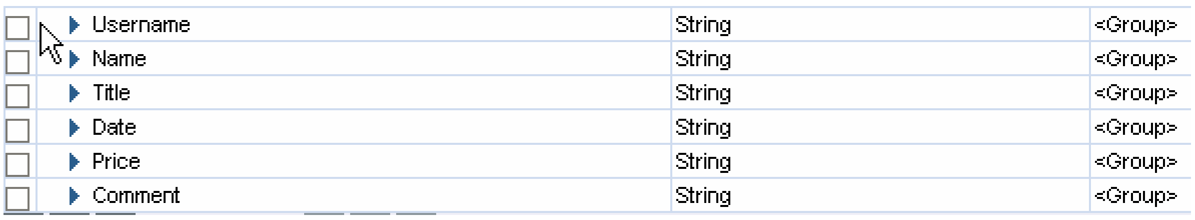
   The *Consolidate Parameter* screen displays all input and output parameters defined for the actions in the block flow. You must group the parameters, so that data is transferred across actions.
2. To consolidate a set of parameters, you must select them, enter a name for the group in the *Consolidate To* field, and choose *Go*.

3. Finally, choose *Save*.

For this example, you must consolidate the parameters as described in the following table.

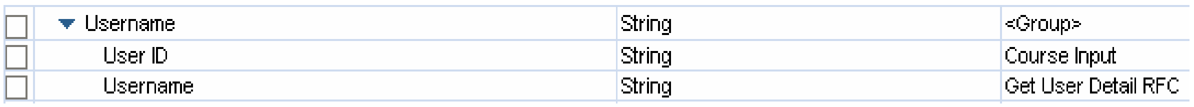
Parameter	**Action**	**Consolidated Parameter**	**Comment**
Name	Course Input	Name	
Address\Fullname	Get User Detail RFC		To select this parameter, expand the Address context parameter and scroll down to find the Fullname entry.
Name	Course Approval		
/name	Course Persistenc		
Name	Course Display		
Title	Course Input	Title	
Title	Course Approval		
/title	Course Persistence		
Title	Course Display		
Date	Course Input	Date	
Date	Course Approval		You will find two Date entries for the Course Approval action. Select the first parameter.
/date	Course Persistence		
Date	Course Display		
Price	Course Input	Price	
Price	Course Approval		
/price	Course Persistence		
Price	Course Display		
Comment	Course Input	**Comment**	
Comment	Course Approval		Select both entries that you will find.
/comment	Course Persistence		
Comment	Course Display		

As a result, six parameter groups are created, as shown in figures 38 to 44.



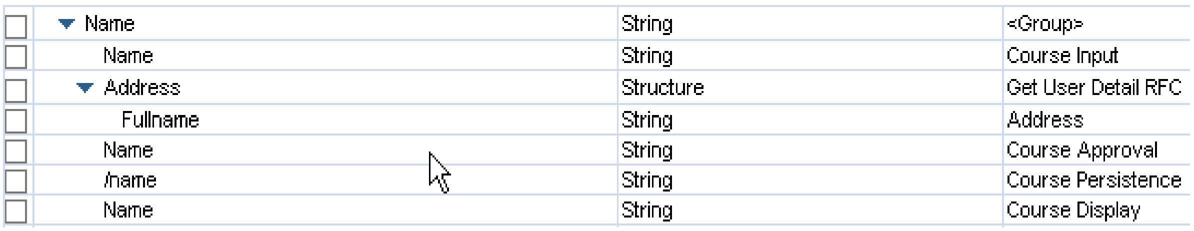
Username	String	<Group>
Name	String	<Group>
Title	String	<Group>
Date	String	<Group>
Price	String	<Group>
Comment	String	<Group>

Figure 38



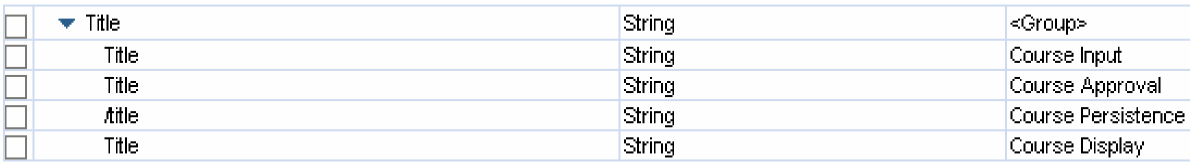
Username	String	<Group>
User ID	String	Course Input
Username	String	Get User Detail RFC

Figure 39



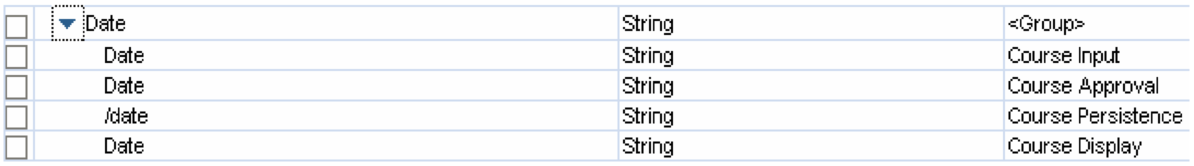
Name	String	<Group>
Name	String	Course Input
Address	Structure	Get User Detail RFC
Fullname	String	Address
Name	String	Course Approval
/name	String	Course Persistence
Name	String	Course Display

Figure 40



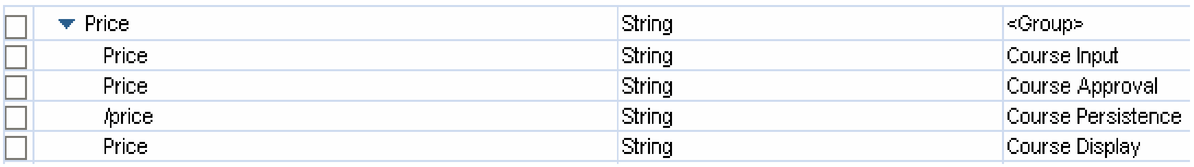
Title	String	<Group>
Title	String	Course Input
Title	String	Course Approval
/title	String	Course Persistence
Title	String	Course Display

Figure 41



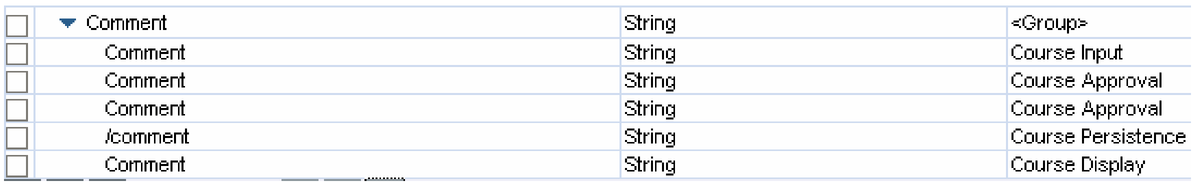
Date	String	<Group>
Date	String	Course Input
Date	String	Course Approval
/date	String	Course Persistence
Date	String	Course Display

Figure 42



Price	String	<Group>
Price	String	Course Input
Price	String	Course Approval
/price	String	Course Persistence
Price	String	Course Display

Figure 43



☐	Comment	String	<Group>
☐	Comment	String	Course Input
☐	Comment	String	Course Approval
☐	Comment	String	Course Approval
☐	/comment	String	Course Persistence
☐	Comment	String	Course Display

Figure 44

4.2.5 Activate the Block

To activate the block, choose *Activate Block* from the contextual panel, and confirm the activation.

To activate the block, you must have activated all callable objects and actions that you used.

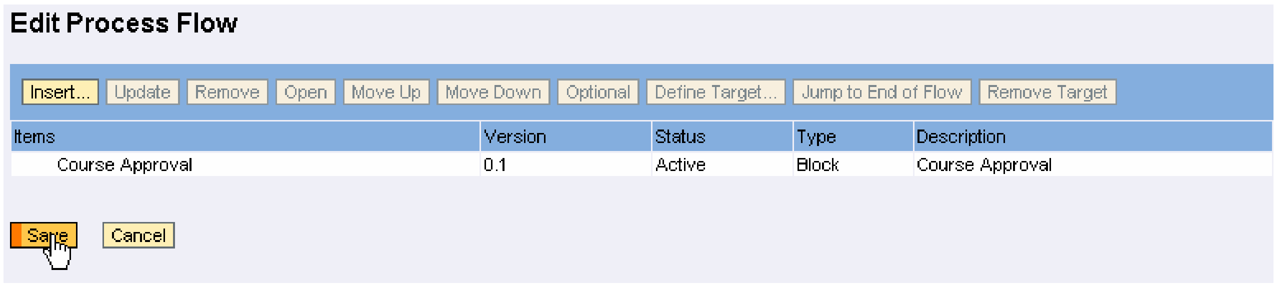
4.3 Create the Process Template

4.3.1 Create a Process

1. To return to the gallery, click on the link *Gallery* in the upper left corner of the screen.
2. Select the folder *\Course Request and Approval\Processes.*
3. Choose *Create Process* from the contextual panel.
   For name and description, enter *Course Approval*.
4. Make sure that the following options are enabled:
   - *Multiple instances are permitted*
   - *Process is started automatically*
5. Choose *Save and Open*.
   The process' design time opens.

4.3.2 Define the Process Flow

1. Choose *Edit Process Flow* from the contextual panel.
2. Choose *Insert*.
3. Use *Select…* to insert an existing block.
4. Navigate to the *\Course Request and Approval\Blocks* and select *Course Approval*.
5. Choose *Select*.
6. Choose *Save*.



Edit Process Flow

Items	Version	Status	Type	Description
Course Approval	0.1	Active	Block	Course Approval

Figure 45

4.3.3 Consolidate Roles

Choose *Consolidate Roles* from the contextual panel and make sure that the roles are consolidated to the following groups:

- Employee
- Manager

To confirm the consolidation, choose *Save*.

4.3.4 Consolidate Parameters

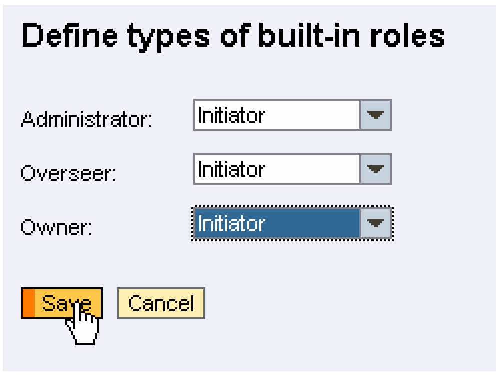
Choose *Consolidate Parameters* from the contextual panel and make sure the parameters are consolidated as described in the section on creating a block.

To confirm the consolidation, choose *Save*.

4.3.5 Define Types of Built-In Roles

1. Choose *Define Types of Built-In Roles* from the contextual panel.
2. From the dropdown list, select Initiator for all three default process roles (Administrator, Overseer, and Owner).

   At process initiation, you will not be asked to assign users to these roles. The user that initiates the process will be automatically assigned to them.
3. Choose *Save*.



Define types of built-in roles

Administrator:	Initiator
Overseer:	Initiator
Owner:	Initiator

Figure 46

4.3.6 Activate the Process

To activate the process, choose *Activate Process* from the contextual panel, and confirm the activation.

To activate the process, you must have activated the block you use.

4.3.7 Define Default Roles

To complete this section, you must first create the users that you will assign to the Employee and Manager roles.

To do that, use the User Management Console in the portal. Create the following users, for example:

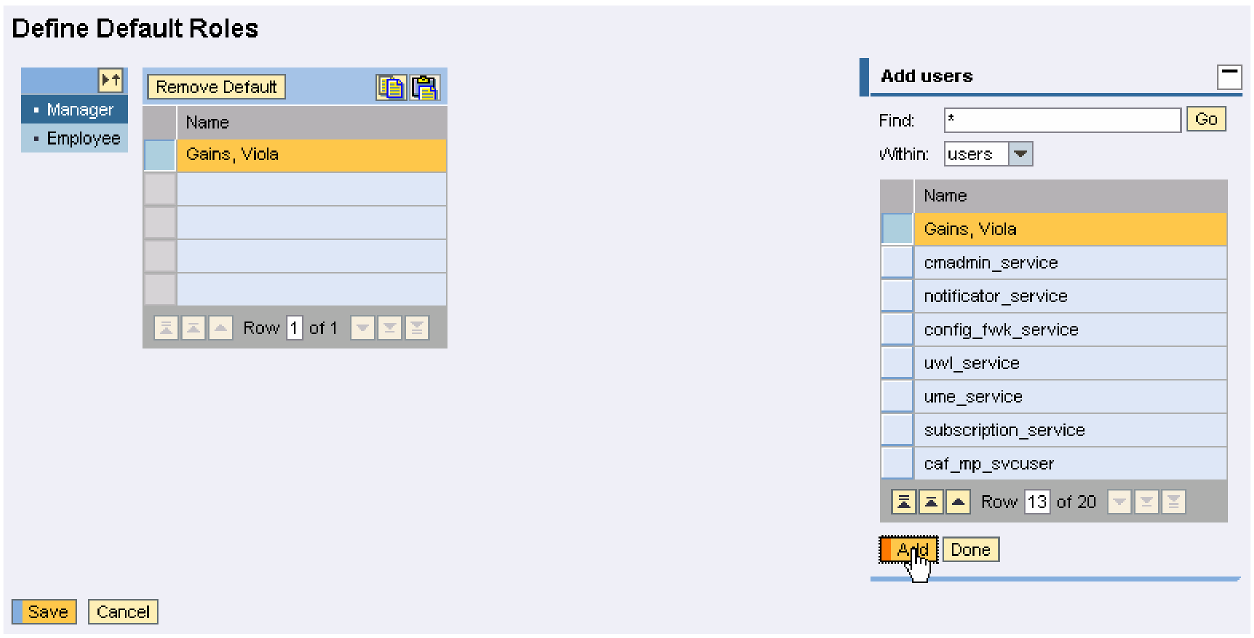
- Viola Gains (user ID `Gainsv`)
- Randy Gordon (user ID `Gordonr`)

Assign the following portal role to both users:

- GP User (`com.sap.caf.eu.gp.roles. user`)

1. In the process' design time, open the *Default Roles* view.
2. Select *Manager* in the left-hand side of the screen.

3. In the *Add users* area, enter an asterisk (*) to overwrite the string `<search term>`, and choose *Go*.
4. Select *Gains, Viola* in the result list, and choose *Add* (figure 47).



Define Default Roles

- Manager
- Employee

Remove Default

	Name
	Gains, Viola

Row 1 of 1

Add users

Find: * Go

Within: users

	Name
	Gains, Viola
	cmadmin_service
	notificator_service
	config_fwk_service
	uwl_service
	ume_service
	subscription_service
	caf_mp_svcuser

Row 13 of 20

Add Done

Save Cancel

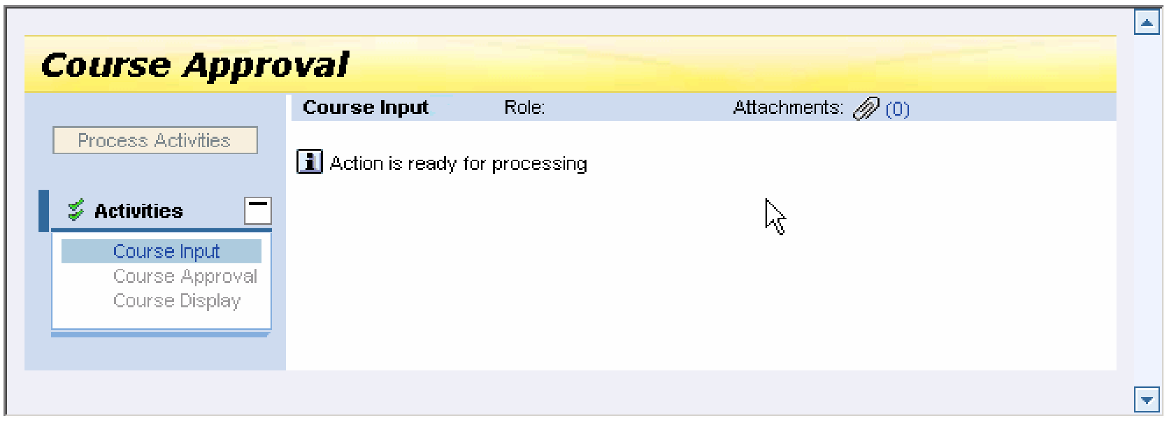
Figure 47

5. Repeat steps 2 to 4 to add *Gordon, Randy* to the *Employee* role.
6. Choose *Done*, and then *Save*.

4.4 Test the Process

4.4.1 Start the Process

1. In the process design time, open the Process Details view.
2. Open the Basic Data tabstrip, and choose Generate Instantiate URL.
   An URL used for process instantiation is generated.
3. Choose Open Instantiate Application.
   As a result, the process has been started (figure 48).



Course Approval

Process Activities

Activities

- Course Input
- Course Approval
- Course Display

Course Input   Role: Attachments: (0)

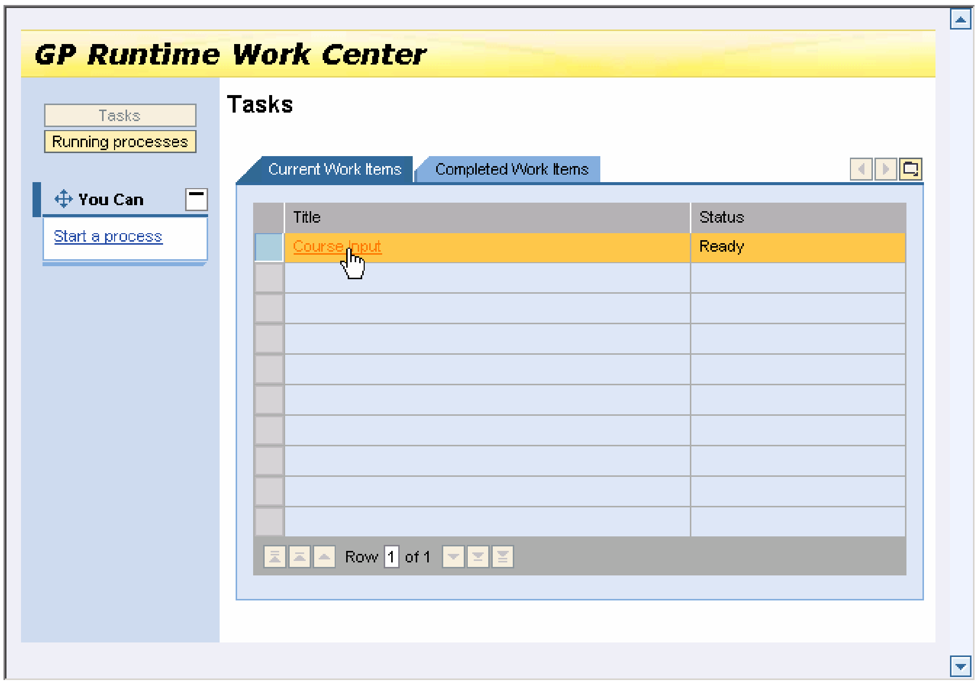
Action is ready for processing

Figure 48

4.4.2 Execute the Process

1. Log on to the portal with the user ID of Randy Gordon.
2. Open the Guided Procedures runtime.

   The Course Input item should appear in Randy’s work list, ready for processing. Choose the item.

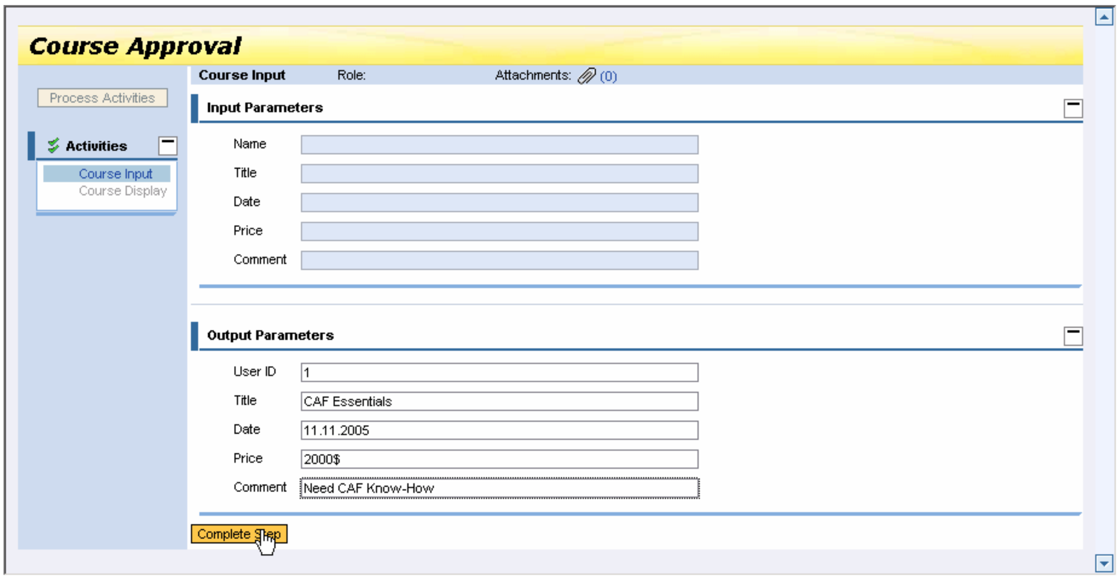


Title	Status
Course Input	Ready

Figure 49

3. The Course Input screen is displayed in a new window.

   Enter the following sample data in the output parameters, as shown in figure 50:
   - UserId: 1
   - Title: CAF Essentials
   - Date: 11.11.2005
   - Price: 2.000$
   - Comment: Need CAF Know-How

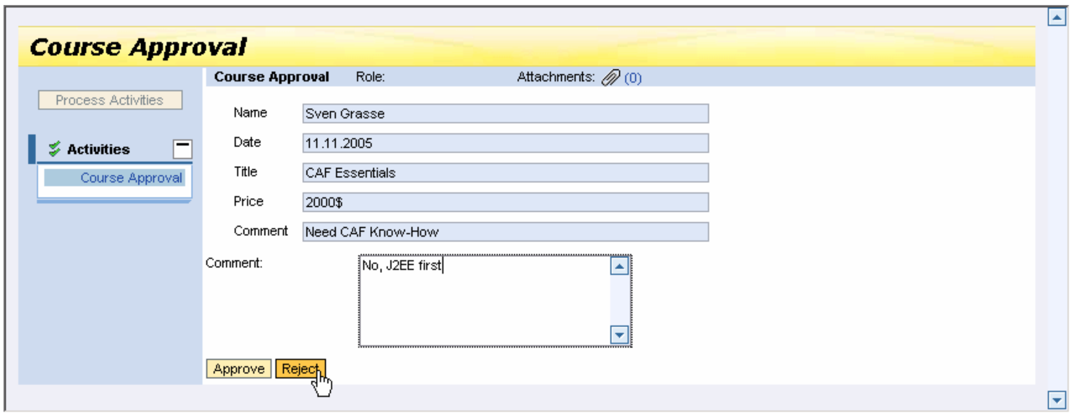


User ID: 1
Title: CAF Essentials
Date: 11.11.2005
Price: 2000$
Comment: Need CAF Know-How

Figure 50

4. Choose *Complete Step*, and close the window.
5. Refresh Randy Gordon's task list. The *Course Input* work item disappears.
6. Log off from the portal, and log on again as Viola Gains.

   The work item Course Approval is ready for processing.
7. Choose the item, which opens in a new window.
8. Test the *Reject* option first. As a confirmation, you get a screen saying that the task has been completed. You can close the window now.



Name: Sven Grasse
Date: 11.11.2005
Title: CAF Essentials
Price: 2000$
Comment: Need CAF Know-How
Comment: No, J2EE first

Figure 51

9. Refresh Viola's task list. The *Course Approval* work item disappears.
10. Log off from the portal, and log on as Randy.

    The Course Input item is displayed again, and Randy must complete the step again entering another data – for example:
    - UserId: **1**
    - Title: J2EE Basics
    - Date: 06.12.2005
    - Price: **2.500$**

- Comment: Hope this one fits

11. Log on as Viola once again, and approve the request.
12. Switch to Randy's taks list. The Course Display item is ready for processing.
13. Choose it and review the data. Then choose OK to complete the process.
14. Check if the course request has been saved in the database.

    To do that, open the CAF Service Browser using the following URL:

    `http://<host>:<port>/webdynpro/dispatcher/sap.com/caf~UI~servicebrowser/ServiceBrowser?cafsource=true`

    Make sure you enter the correct host and port for your Java server.
15. In the *Available Services* screen area, select *sap.com* → *xteched* → *CourseService* → *Course* → *findByTitle.*
16. In the *Data Component* area, enter an asterisk (*) in the *title* input field and choose *Execute query*.

    You should see the entry created during the process execution.

4.5 Result

You have successfully created and tested the Course Request and Approval process. You are now familiar with creating composite applications using SAP CAF.