SAP
Records Management

Tutorial:
Implementing a Service Provider

Developer Documentation

May 13, 2004

Contents

1 Introduction

This document contains a tutorial about implementing a service provider. Prerequisites are a sound working knowledge of the terminology and architecture of the Records Management Framework. For more information about this, and a systematic representation of service provider methods, see the Records Management reference documentation for developers. We recommend that you study this tutorial and the reference documentation together.

2 Tasks

We want to create a service provider that you can use to display and edit flights (from the table SFLIGHT) for an airline. This service provider must provider the user with the following functions (in Records Management activities):

- Find flights
- Display a flight

In a later step, we will add extra functions to the service provider.

3 Definition of the CONNECTION and SP-POID Parameters

To implement a service provider, we first need some information about the repository, that is, the location to which the business data of the service provider is saved. In the SAP system, flights are stored in the SFLIGHT table, which means that our repository is the R/3 database. To identify a flight uniquely, we must use the primary key to access the SFLIGHT table. We can get the structure of the primary key from the table definition of SFLIGHT:

- CARRID
- CONNID
- FLDATE

The SP-POID of a service provider must always contain the primary key required for accessing the repository, with the exception of any parts that are already defined in the connection parameters. In our example, we assume that the airline is defined in the connection parameters; we specify the connection and flight date in the SP-POID.

Note: The client is not included in the CONNECTION or SP-POID parameters, since it is determined at runtime when the user logs on.

4 Publishing the CONNECTION and SP-POID Parameters

The CONNECTION and S -POID parameters must be registere d in the framework. To do this, you implement an ABAP OO class that satisfies a specific class role.

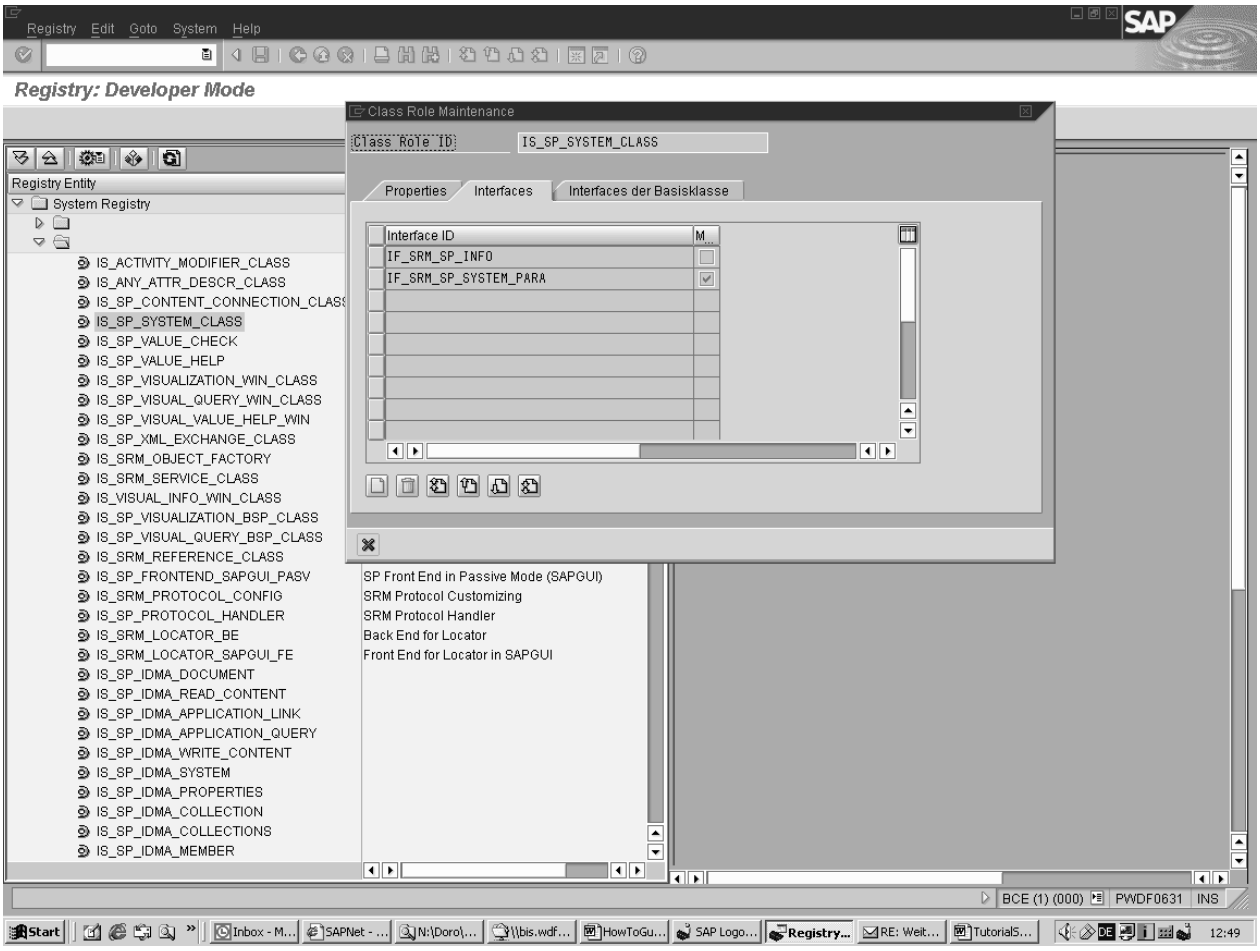
Class Roles
A class role defines certain requirements that must be satisfied by an (ABAP OO) class before it can be used in a specific role (that is, fulfill a specific function). A class role definition defines the (ABAP OO) interfaces that a class must implement, and its superclass.

Example of a class role definition:
A class is an SP client class if it inherits from CL_SRM_SP_CLIENT_OBJ and implements the IF_SRM_SP_CLIENT_WIN interface.

Note:
The class does not have to inherit directly from the class defined in the class role; the class only needs to inherit from another class that inherits from the basis class.

Class roles are managed in the SRM Regis try (transaction SRMREGEDIT). If you want to know which interfaces and basis class are required by a class role, consult this registry.

Transaction SRMREGEDIT shows us the definition of the class role, IS_SP_SYSTEM_CLASS, needed to publish the CONNECTION and SP-POID parameters:



On the *Interfaces* tab, you can see that the class role requires the implementation of the IF_SRM_SP_SYSTEM_PARA interface. On the (hidden) *Properties* tab page, you can see that the class role demands that classes inherit from C L_SRM, the basis class of all RM programming objects.

We can use the development environment to create the system class for our service provider:

- The class inherits from CL_SRM.
- The class implements the IF_SRM_SP_SYSTEM_PARA interface.

For reference, the classes of this tutorial are included in the package SRM_FRAMEWORK_DEMO. The system class of the tutorial SP is CL_SRM_SP_TUTORIAL_SYSTEM.

To implement IF_SRM_SP_SYSTEM_PARA, we need to create the following methods:

- Definition of the CONNECTION parameters – GET_ATTR_DESC_CONNECTION
- Definition of the CONTEXT parameters – GET_ATTR_DESC_CONTEXT
- Definition of the SP-POID parameters – GET_ATTR_DESC_SP_POID

Important: There are no CONTEXT parameters in the first step, but you must still create this method (with no content). If you do not, runtime errors can occur.

In the methods IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_CONNECTION and IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_SP_POID, we use the attribute description objects to publish the required parameters.

Factory Object

The IF_SRM interface of the basis class provides you with a range of functions for programming a service provider. When you call IF_SRM~GET_SRM_OBJECT_FACTORY, you get a reference to the IF_SRM_SRM_OBJECT_FACTORY interface, which provides you with methods for generating Framework objects:

- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ACTIVITY_LIST
  Generates an activity list object.
- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ATTR_DESC_*
  Generates attribute description objects (with various types).
- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ATTRIBUTE_VALUE
  Generates attribute value objects.

Attribute Description Objects

Attribute description objects contain the definition of a data type in Records Management. Attribute value objects contain variants of this definition.

You can use IF_SRM_SRM_OBJECT_FACTORY to get attribute description objects. Different types of attribute description objects are available for different purposes; you must call the appropriate method of IF_SRM_SRM_OBJECT_FACTORY for the object you require:

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_ANY
  Gets a generic attribute description object for various purposes.
- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_CONNECTION
  Gets an attribute description object for the definition of CONNECTION parameters.
- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_CONTEXT
  Gets an attribute description object for the definition of CONTEXT parameters.
- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_SP_POID
  Gets an attribute description object for the definition of SP-POID parameters.
- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_INFO
  Gets an attribute description object for the definition of INFO attributes.

Attribute Description Objects (Continued)

Once you have the attribute description object, you must use IF_SRM_EDIT_ATTRIBUTE_DESC to fill it. You do this in two steps:

- Describe the general properties.
  To do this, call IF_SRM_EDIT_ATTRIBUTE_DESC~SET_GENERAL_DESCRIPTION with a structure of the type SRMADGEN.

- TYPE: Determines the type.
  IF_SRM_ATTRIBUTE_DESC=>STRING
  IF_SRM_ATTRIBUTE_DESC=>INTEGER
  IF_SRM_ATTRIBUTE_DESC=>INTERFACE
- IS_LIST: Attribute can have multiple values.
- IS_MAND: Attribute is a mandatory field.
- IS_HELP: Value help exists for the attribute.
- IS_CHECK: Value check exists for the attribute.
- TEXT: Short text

- Describe the type-specific properties by calling one of the following methods.

  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_STRING_DESCRIPTION
    Defines the specific properties for STRING attributes.
  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_INTEGER_DESCRIPTION
    Defines the specific properties for INTEGER attributes.
  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_INTERFACE_DESCRIPTION
    Defines the specific properties for INTERFACE attributes.

For the description of the SP-POID parameters, we get the following code:

```
method IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_SP_POID .

data:     factory type ref to if_srm_srm_object_factory,
          ead type ref to if_srm_edit_attribute_desc,
          general_desc type srmadgen,
          string_desc type srmadstr.

* get object factory
factory = me->if_srm~get_srm_object_factory( ).

*---------------------------------------------------------------------
* attribute description for SFLIGHT-CONNID
*---------------------------------------------------------------------

* create attribute description for CONNID
ead = factory->create_attr_desc_sp_poid( ).

* set general description
general_desc-id = 'CONNID'.
general_desc-text = text-001.
general_desc-type = IF_SRM_ATTRIBUTE_DESC=>STRING.
general_desc-is_list = if_srm=>false.
general_desc-is_mand = if_srm=>true.
ead->set_general_description( general_desc ).

* set specific description for type STRING
string_desc-max_length = 4.
ead->set_string_description( string_desc ).

append ead to re_desc.

*---------------------------------------------------------------------
* attribute description for SFLIGHT-FLDATE
*---------------------------------------------------------------------

* create attribute description for FLDATE
ead = factory->create_attr_desc_sp_poid( ).

* set general description
general_desc-id = 'FLDATE'.
general_desc-text = text-002.
general_desc-type = IF_SRM_ATTRIBUTE_DESC=>STRING.
general_desc-is_list = if_srm=>false.
general_desc-is_mand = if_srm=>true.
ead->set_general_description( general_desc ).

* set specific description for type STRING
string_desc-max_length = 8.
ead->set_string_description( string_desc ).

append ead to re_desc.

endmethod.
```

The following code publishes our CONNECTION parameter CARRID:

```
method IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_CONNECTION .

data:     factory type ref to if_srm_srm_object_factory,
          ead type ref to if_srm_edit_attribute_desc,
          general_desc type srmadgen,
          string_desc type srmadstr.

* get object factory
factory = me->if_srm~get_srm_object_factory( ).

*---------------------------------------------------------------------
* attribute description for SFLIGHT-CARRID
*---------------------------------------------------------------------

* create attribute description for CARRID
ead = factory->create_attr_desc_connection( ).

* set general description
general_desc-id = 'CARRID'.
general_desc-text = text-003.
general_desc-type = IF_SRM_ATTRIBUTE_DESC=>STRING.
general_desc-is_list = if_srm=>false.
general_desc-is_mand = if_srm=>true.
general_desc-is_help = if_srm=>false.
general_desc-is_check = if_srm=>false.
ead->set_general_description( general_desc ).

* set specific description for type STRING
string_desc-max_length = 4.
ead->set_string_description( string_desc ).

append ead to re_desc.

endmethod.
```

Note: No value check or value help is possible for SP -POID parameters. For t he CONNECTION parameters, we will implement these functions in a later step.

5 Implementing the Service Provider Back End

After we have published the SP parameters, we can start to implement the SP back end. The back end enables the front end of our service provider to access the repository.

We want the front end to use an interface to access the back end. This is the only way that allows us to switch the back-end class later, if necessary.

First, we define an interface for accessing the back end. It contains three methods:

- *get_flight_data* extracts the flight data (to be displayed).
- *get_flights* gets a list of flights (for the search dialog).
- *set_sppoid_para* sets the SP -POID parameters for the transition from the model to the instance.

To implement our SP back end, we use the IF_SRM_SP_TUTORIAL_BACKEND interface from the package SRM_FRAMEWORK_DEMO.

The back end of a service provider must satisfy the IS_SP_CONTENT_CONNECTION_CLASS class role. We get the required data from the RM Registry:

- The back end must inherit from the CL_SRM_SP_CONNECTION class.
- You must implement the IF_SRM_CONNECTION interface.
- The IF_SRM_CONNECTION_NEW interface is optional.
- The IF_SRM_CONTEXT_AUTOMATION interface is optional.
- The IF_SRM_NON_VISUAL_INFO_SP interface is optional.

Of course, we also need to implement our own interface, IF_SRM_SP_TUTORIAL_BACKEND.

You can now use the development interface to create the class (the template is CL_SRM_SP_TUTORIAL_BACKEND).

First, we create a private method, which gets use the value of the connection parameter, the airline (or carrier). This method is called GET_CONNECTION_PARA and has the return value RE_CARRID with the type S_CARR_ID. Errors can occur when the connection parameters are being extracted, which is why the method declares various exception classes:

Parameter Type	**Name**	**Data Element**
RETURNING	RE_CARRID	S_CARR_ID
EXCEPTION	CX_SRM_INITIALIZATION	
EXCEPTION	CX_SRM_POID	
EXCEPTION	CX_SRM_ATTRIBUTE_VALUE	

The code is relatively simple; all it does is extract and return the value.

```
method GET_CONNECTION_PARA .

 data:  lt_values type srm_list_string,
        wa_value type srmliststr.

* get values for CARRID
  lt_values = me->if_srm_connection_attr~get_string_value( 'CARRID' ).

* since CARRID cannot have multiple values, get single value
  loop at lt_values into wa_value.
  endloop.

  re_carrid = wa_value-value.

endmethod.
```

Another private method gets us the key parts from the SP -POID. This method is called GET_SPPOID_PARA:

Parameter Type	**Name**	**Data Element**
EXPORTING	EX_CONNID	S_CONN_ID
EXPORTING	EX_FLDATE	S_DATE
EXCEPTION	CX_SRM_INITIALIZATION	
EXCEPTION	CX_SRM_POID	

Here, the code is also simple:

```
method GET_SPPOID_PARA .

 data:  s_connid type string,
        s_fldate type string.

* get values from SP POID
  s_connid = me->if_srm_poid~get_sp_poid_value_by_id( 'CONNID' ).
  s_fldate = me->if_srm_poid~get_sp_poid_value_by_id( 'FLDATE' ).

* convert values into proper format
  ex_connid = s_connid.
  ex_fldate = s_fldate.

endmethod.
```

Using these methods, we can n ow code the methods of our IF_SRM_SP_TUTORIAL_BACKEND interface:

The GET_FLIGHTS method gets all flights of the airline defined in the connection parameters, and returns them in an internal table:

```
METHOD if_srm_sp_tutorial_backend~get_flights .

  DATA: carrid TYPE s_carr_id.

* get connection parameter CARRID

  carrid = me->get_connection_para( ).

  SELECT * FROM sflight INTO TABLE re_flights WHERE carrid = carrid.

ENDMETHOD.
```

The GET_FLIGHT_DATA method gets a flight from the table and returns it in a structure. If the flight specified by the SP -POID and the connection parameters cannot be found, an exception with the type CX_SRM_CONNEC_FAILED is raised:

```
METHOD if_srm_sp_tutorial_backend~get_flight_data .

  DATA: carrid TYPE s_carr_id,
        connid TYPE s_conn_id,
        fldate TYPE s_date.

* get connection parameter CARRID
  carrid = me->get_connection_para( ).

* get SP POID parameter CONNID and FLDATE
  CALL METHOD me->get_sppoid_para
    IMPORTING
      ex_connid = connid
      ex_fldate = fldate.

* read dataset from database table SFLIGHT
  SELECT SINGLE * FROM sflight INTO re_flight WHERE carrid = carrid AND
                                        connid = connid AND
                                        fldate = fldate.

  IF sy-subrc <> 0.
    RAISE EXCEPTION TYPE cx_srm_connec_failed
      EXPORTING textid = cx_srm_connec_failed=>et_not_exist.
  ENDIF.

ENDMETHOD.
```

The SET_SPPOID_PARA method receives the SP -POID parameters from the search and sets the values in the POID object:

```
METHOD if_srm_sp_tutorial_backend~set_sppoid_para .

* set the SP POID parameters (when changing state from model to instance)

  DATA: wa_poid_tab TYPE srmpoid,
        lt_poid_tab TYPE srm_list_poid.

  wa_poid_tab-id = 'CONNID'.
  wa_poid_tab-value = im_connid.
  APPEND wa_poid_tab TO lt_poid_tab.

  wa_poid_tab-id = 'FLDATE'.
  wa_poid_tab-value = im_fldate.
  APPEND wa_poid_tab TO lt_poid_tab.

  me->if_srm_poid~set_sp_poid( lt_poid_tab ).

ENDMETHOD.
```

Exception Handling
The Records Management Framework uses exception classes to handle exceptions. Just like the old exceptions, exception classes are declared in the interface of a method. One new feature is that exceptions are propagated upwards automatically, if they have also been declared in the interface of the calling method.
Since the two most important exception classes, CX_SRM_FRAMEWORK and CX_SRM_SP_CLIENT, are declared in the interface of almost all methods that implement a service provider, you do not normally need to specify the handling of exceptions when you program your SP.
The framework handles the exceptions automatically and saves them in the application log; you can view the error messages in transaction SLG1.
(Note: You need activate the logging only once, in the Customizing transaction SRM_APPL_LOG.)

Finally, you need to implement the methods of IF_SRM_CONNECTION:

- IF_SRM_CONNECTION~INITIALIZE
  This method initializes the connection to the back end. In our case, this is not necessary, since there is a permanent connection to the database on the Web Application Server. It is enough just to create an empty method body.

- IF_SRM_CONNECTION~CHECK
  This method checks whether the connection to the repository still exists. The connection is permanent on the Web Application Server, which is why we only create an empty method body here as well.

- IF_SRM_CONNECTION~CONNECT_REPOSITORY
  Here, we connect the repository and check whether the data record specified by the connection parameters and the SP-POID exists. This is checked just by reading the data record. If the data record cannot be read, an exception is raised by the GET_FLIGHT_DATA method:

```
METHOD if_srm_connection~connect_repository .

* try to access the database
  me->if_srm_sp_tutorial_backend~get_flight_data( ).

ENDMETHOD.
```

6 Implementing an SAP Front End for SAPGUI

Two class roles are important for the SP front end:

- IS_SP_VISUALIZATION_WIN_CLASS is the class role that displays an element.
- IS_SP_VISUAL_QUERY_WIN_CLASS is the class role for the visual search dialog.

These class roles can be specified by a single class or by two separate classes. Because we are not planning to reuse the search dialog in our example, we can use one class.

We get the following class requirements from the registry:

The IS_SP_VISUALIZATION_WIN_CLASS class role inherits from CL_SRM_SP_CLIENT_OBJ and implements the following.

- IF_SRM_SP_ACTIVITIES Publishes the visual activities.
- IF_SRM_SP_AUTHORIZATION Authorization check
- IF_SRM_SP_CLIENT_WIN Executes activities (in-place).
- IF_SRM_SP_CLIENT_OUTPLACE Optional: Executes activities (out-place).

The IS_SP_VISUAL_QUERY_WIN_CLASS class role inherits from CL_SRM_SP_CLIENT_OBJ and implements IF_SRM_SP_VISUAL_QUERY_WIN.

We now create a class (CL_SRM_SP_TUTORIAL_FRONTEND) that inherits from CL_SRM_SP_CLIENT_OBJ, and implement the following five interfaces.

6.1.1 Authorization Check: IF_SRM_SP_AUTHORIZATION

We do not want to integrate a separate authorization check in the first step, which is why we return the constant IF_SRM=>TRUE:

```
method IF_SRM_SP_AUTHORIZATION~CHECK_ACTIVITY_AUTHORIZATION.

  re_authorized = if_srm=>true.

endmethod.


method IF_SRM_SP_AUTHORIZATION~CHECK_VIEW_AUTHORIZATION.

  re_authorized = if_srm=>true.

endmethod.
```

6.1.2 Publishing Activities: IF_SRM_SP_ACTIVITIES

Activities
Model activities are activities that relate to the element type (or SPS), such as *Find* or *Create*; instance activities relate to a fixed element, such as *Display* or *Delete*.

Standard activities are activities that have the same semantic meaning for a large n umber of service providers. These activities are defined by SAP and cannot be added to by the customer. All standard activities are created in IF_SRM_ACTIVITY_LIST as constants.

IF_SRM_ACTIVITY_LIST=>

CREATE	Model activity	Creates an element.
QUERY	Model activity	Finds an element.
DISPLAY	Instance activity	Displays an element.
EDIT	Instance activity	Displays an element in change mode.
DELETE	Instance activity	Deletes an element.
INFO	Model activity	Displays the information dialog (handled internally).
INFO	Instance activity	Displays the information dialog (handled internally).
PROTOCOL	Instance activity	Displays an element-specific log.

Activities are published by the IF_SRM_SP_ACTIVITIES interface – implemented by every SP front end – using the ac tivity list object (class with the IF_SRM_ACTIVITY_LIST interface). The factory is used to get the activity list object.

As well as standard activities, service providers can also have specific activities (such as *Update Document*). You must specify a func tion code and a label for specific activities (see IF_SRM_ACTIVITY_LIST->ADD_ACTIVITY).

You can nest activity lists by using IF_SRM_ACTIVITY_LIST ->ADD_ACTIVITY_LIST to insert another activity list object.
If no further user interaction is required, you c an trigger default activities, by double -clicking an element, for example.

In the first step, our service provider has two activities: IF_SRM_ACTIVITY_LIST=>DISPLAY and IF_SRM_ACTIVITY_LIST=>QUERY. We set these activities as a default activity, as appropriate:

```
method IF_SRM_SP_ACTIVITIES~GET_INSTANCE_ACTIVITIES .

DATA:  factory TYPE REF TO if_srm_srm_object_factory,
       activity_description type SRMACTTA.
* create activity list
  factory = me->if_srm~get_srm_object_factory( ).
  re_activities = factory->create_activity_list( ).
* activity display (default activity)

  re_activities->add_standard( if_srm_activity_list=>display ).  re_activities-
>set_default( if_srm_activity_list=>display ).

endmethod.
```

6.1.3 Executing Activities: IF_SRM_SP_CLIENT_WIN

IF_SRM_SP_CLIENT_WIN contains various methods that can be called when activities are executed:

- IF_SRM_SP_CLIENT_WIN~GET_EVENT_OBJECT
  Uses the Client Framework to get the event object. This method has a standard implementation that must be integrated by the SP.
  event_object = me->if_srm_sp_client_win~event_object.

- IF_SRM_SP_CLIENT_WIN~SET_EVENT_OBJECT
  Uses the Client Framework to set the event object. This method has a standard implementation that must be integrated by the SP.
  me->if_srm_sp_client_win~event_object = im_event_object.

- IF_SRM_SP_CLIENT_WIN~OPEN
  Called when the SP is opened. When this happens, the client can be initialized internally (controls are constructed, for example).

- IF_SRM_SP_CLIENT_WIN~MY_ACTION
  Called to execute an activity.

- IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH
  Gets the display width for an activity; between 0% (for non-visual activities) and 100%.

- IF_SRM_SP_CLIENT_WIN~ANSWER_ON_EVENT
  Called by the framework to send an asynchronous response to a request to the sender.

- IF_SRM_SP_CLIENT_WIN~SYSTEM_INFO
  Used to send system messages (such as *Framework closed*).

We do not initially need the ANSWER_ON_EVENT and SYSTEM_INFO methods; they are created without content. These methods are used to register system events and send responses to asynchronous requests. The service provider for flights does not send any requests, which means that it cannot receive any asynchronous responses either. System events are not relevant for us, since the server provider does not modify any data and th erefore does not need to react to the framework being closed.

The GET_EVENT_OBJECT and SET_EVENT_OBJECT methods are filled with the appropriate default implementation:

```
method IF_SRM_SP_CLIENT_WIN~GET_EVENT_OBJECT .
* default implementation
  event_object = me->if_srm_sp_client_win~event_object.
endmethod.

method IF_SRM_SP_CLIENT_WIN~SET_EVENT_OBJECT .
* default implementation
  me->if_srm_sp_client_win~event_object = im_event_object.
endmethod.
```

The GET_CLIENT_WIDTH method is used to determine whether a certain activity triggers an in-place representation. The specified value defines the display width required by the service provider (between 0% and 100%). Our *Display* activity has an in -place representation, which is why we specify the value 100.

```
method IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH .

  re_client_width = 100.

endmethod.
```

6.1.4 Methods for Displaying the Flight Information

Dynamic documents and the ALV Grid Control are used for the actual display of the flight information. The programming of dynamic documents and the ALV Grid Control is not part of this tutorial, which is why we copy the appropriate methods from the template class in the development environment:

- CL_SRM_SP_TUTORIAL_FRONTEND->BUILD_VISUALIZATION
- CL_SRM_SP_TUTORIAL_FRONTEND->DISPLAY_FLIGHT
- CL_SRM_SP_TUTORIAL_FRONTEND->DISPLAY_FLIGHT_SELECTION

6.1.5 Generating the Visualization: IF_SRM_SP_CLIENT_WIN~OPEN

If a service provider is being displayed for the first time, the IF_SRM_SP_CLIENT_WIN~OPEN method is called. Here, the controls needed fo r visualizing the SP must be generated (wrapped in the private method BUILD_VISUALIZATION in the example). The service provider gets a reference to a control container from the Client Framework, and must then provide a pointer to its own top container. (Th e Client Framework needs this container to be able to activate and deactivate the visualization of an SP completely.)

```
method IF_SRM_SP_CLIENT_WIN~OPEN .

  re_main_control = build_visualization( im_parent ).

endmethod.
```

6.1.6 Executing an Activity: IF_SRM_SP_CLIENT~MY_ACTION

If you want to execute an activity selected in the Organizer or in the record, the Client Framework uses the IF_SRM_SP_CLIENT~MY_ACTION method to call the service provider. Using the activity in the request object, the service provider must now decide which activity to execute. After the activity has been executed (wrapped in the private method DISPLAY_FLIGHT

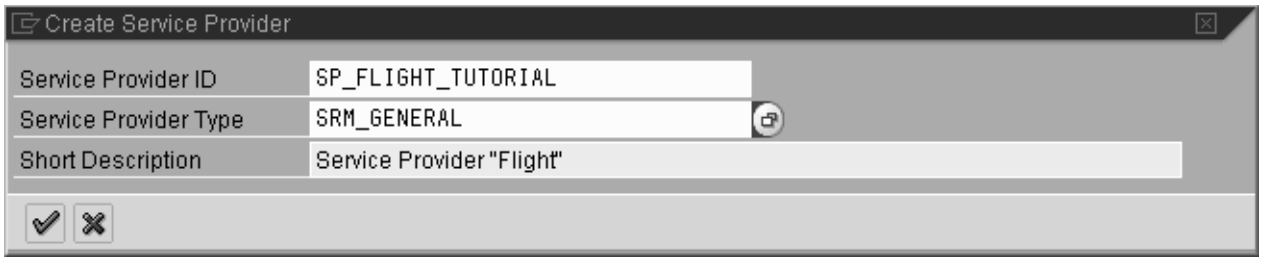
in this tutorial), the service provider must set the result of the activity (a POID) and the state of the activity (constants in IF_SRM_REQUEST=>ACTIVITY_STATE...) in the request object:

```
METHOD if_srm_sp_client_win~my_action .

  DATA: my_backend TYPE REF TO if_srm_sp_tutorial_backend,
        my_poid type ref to if_srm_poid,
        flight_data TYPE sflight.

  CASE im_request->get_activity( ).

    WHEN if_srm_activity_list=>display.
*     get connection to backend
      my_backend ?= me->if_srm_sp_client_obj~get_content_connection_object( ).

*     get data from backend
      flight_data = my_backend->get_flight_data( ).

*     display flight data
      me->display_flight( flight_data ).

  ENDCASE.

* set result and activity state
  my_poid = me->if_srm_sp_object~get_poid( ).
  im_request->set_result( my_poid ).
  im_request->set_activity_state(
                                      if_srm_request=>activity_finished_with_ok ).

ENDMETHOD.
```

7 Registering the Service Provider

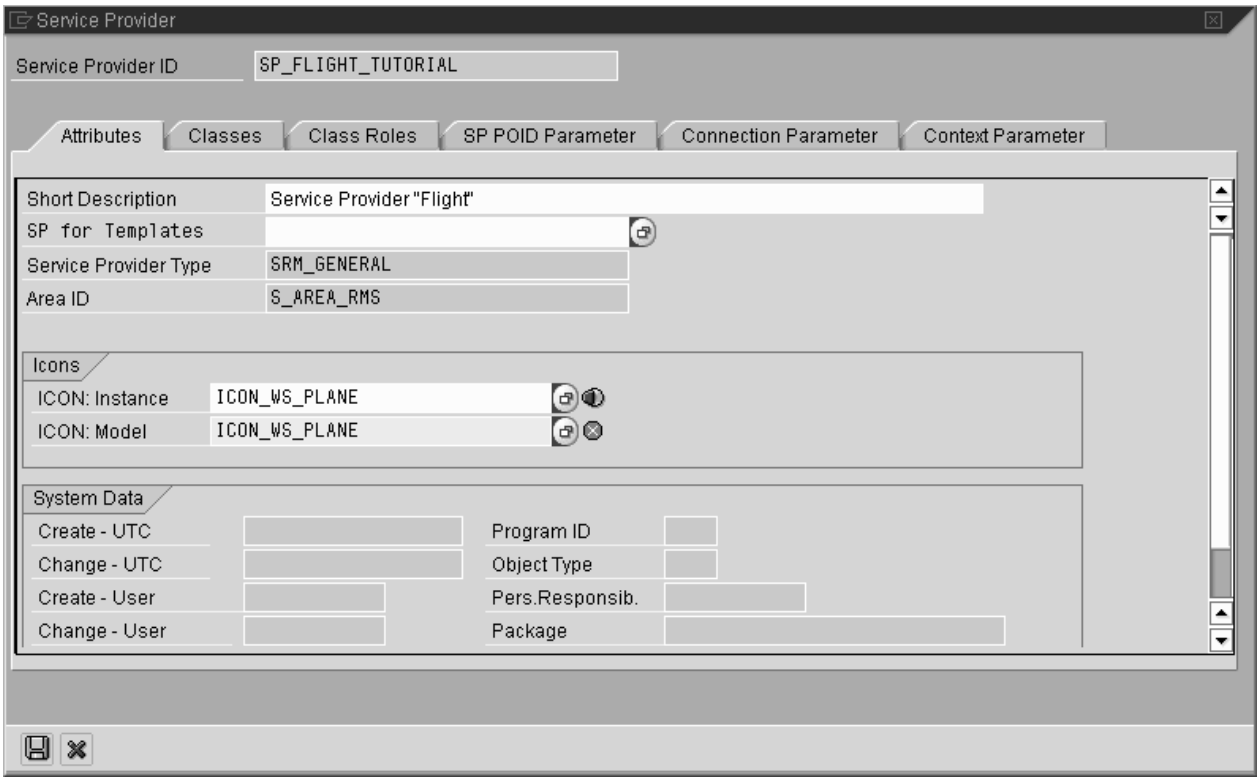
The service provider can now run. To be able to use it, you must first register it in the RM Registry (transaction SRMREGEDIT).

When you call the transaction, select the *S_AREA_RMS* node under the *Application Registry*. Right-click and choose *Create Service Provider*.

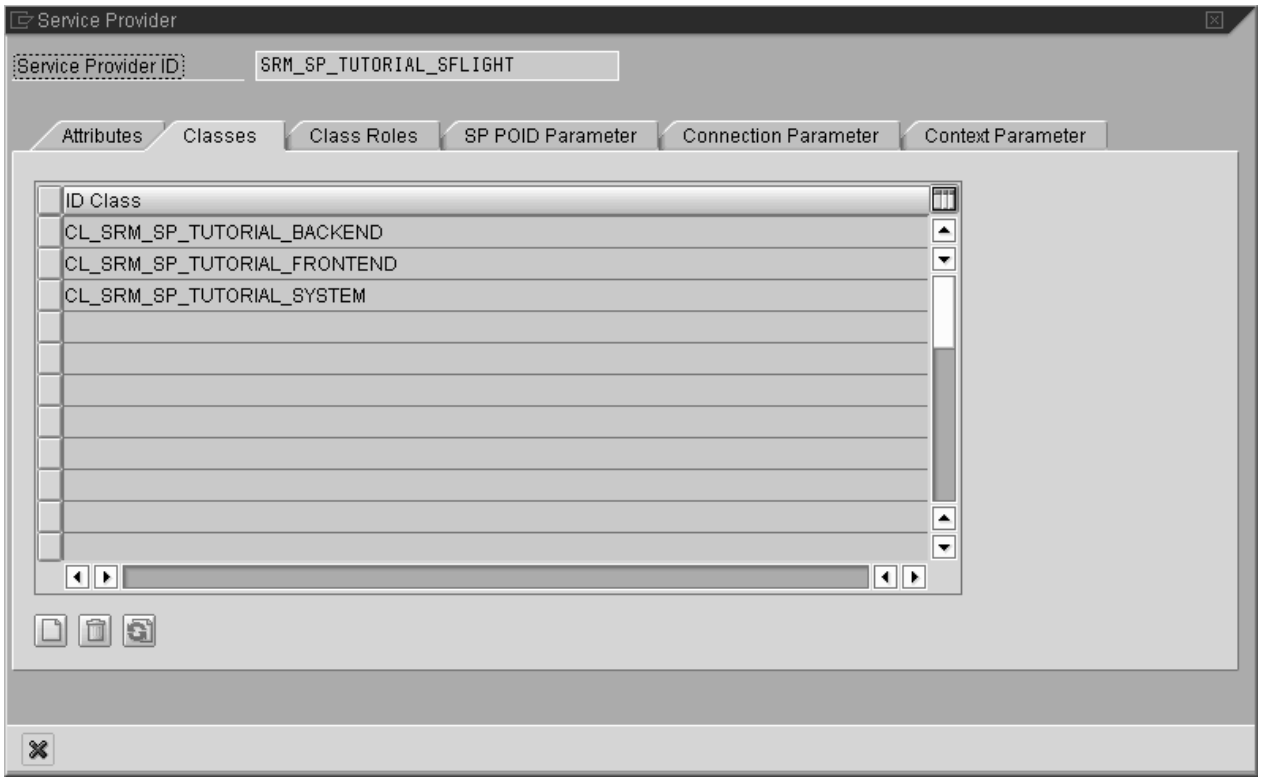


Always choose SRM_GENERAL as the service provider type; the other service provider types are used for special purposes. After you have given the service provider a name, a dialog appears with several tab pages.

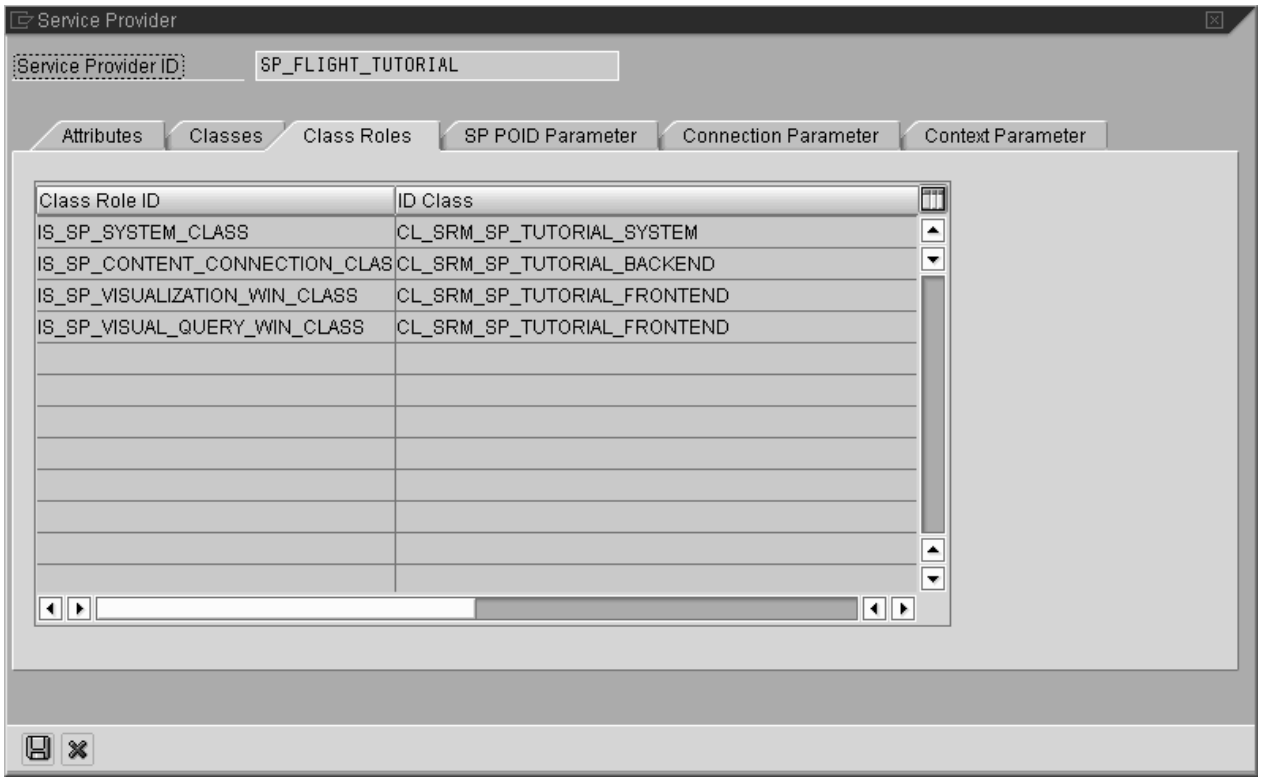
On the *Attributes* tab page, you can specify icons to be displayed in the record and the Organizer.



On the *Classes* tab page, specify the three classes you have created in this tutorial.

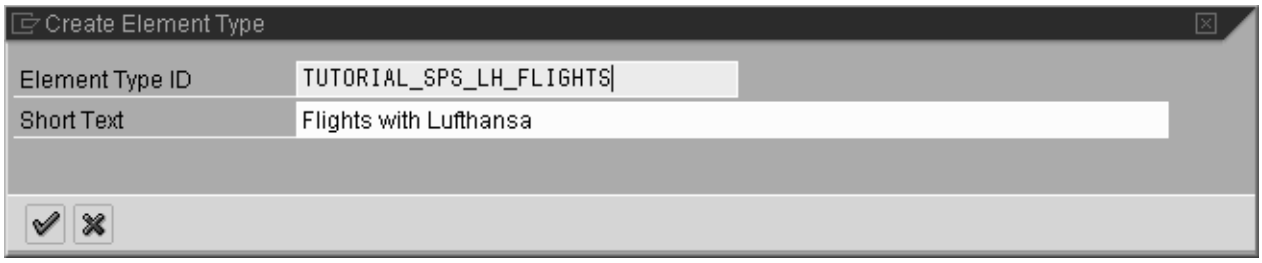


On the *Class Roles* tab page, check whether the required class roles have been satisfied.

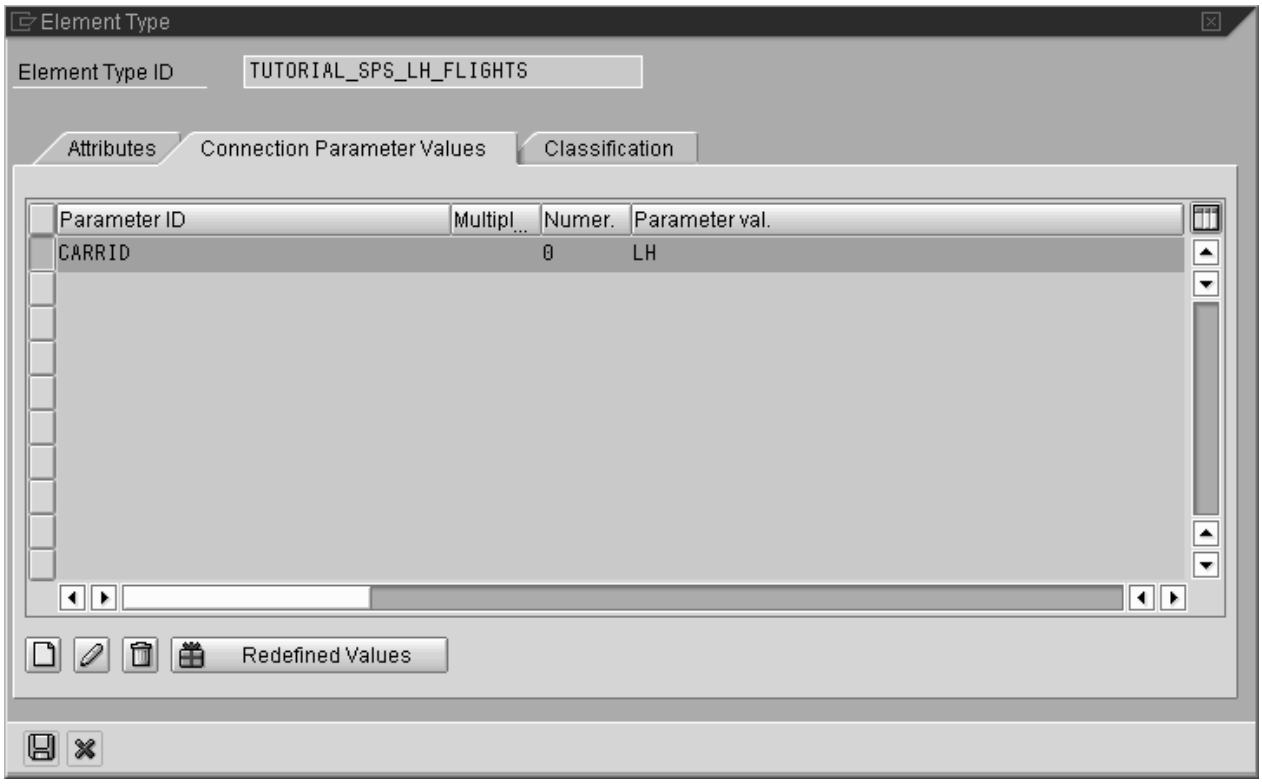


The published parameters are entered on the *SP POID Parameters* and *Connection Parameters* tabs (not shown here).

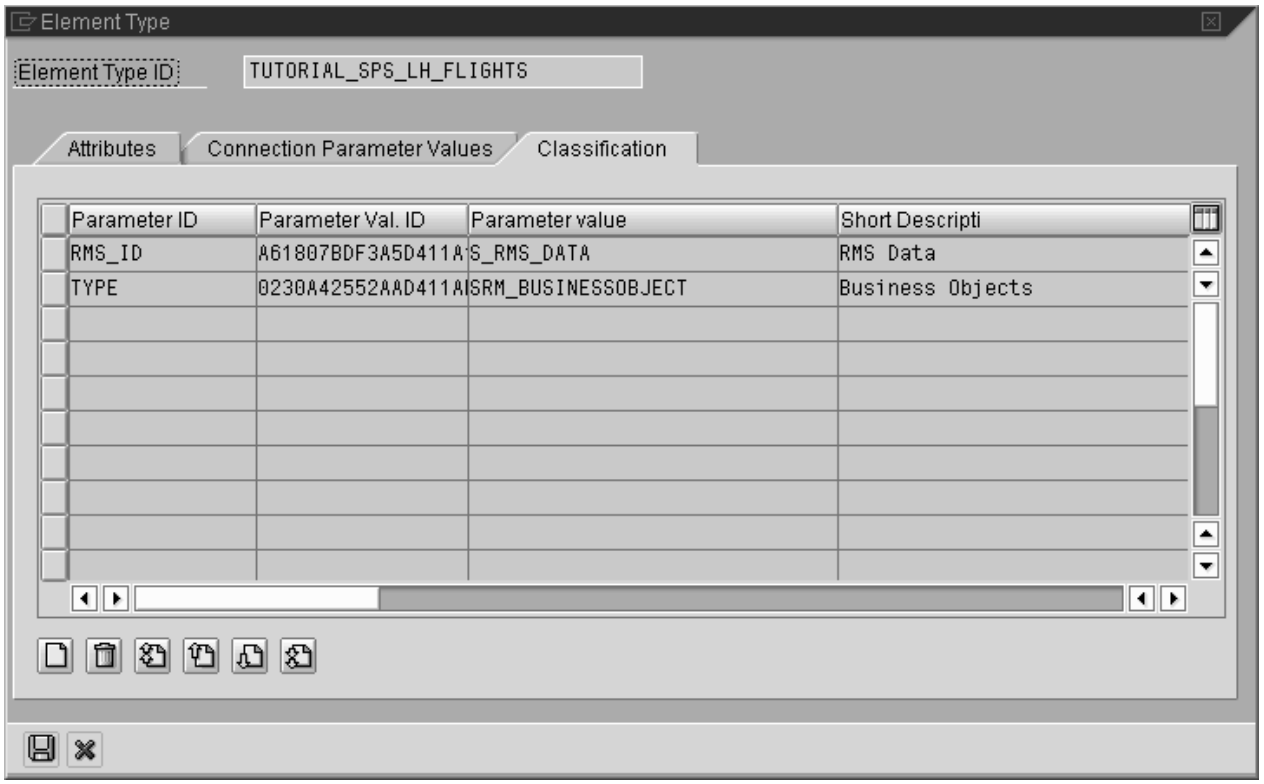
Once you have created the service provider successfully, you must create an SPS (also known as an element type). To do this, select the node of your new service provider. Right -click and choose *Create Element Type*.



A dialog box appears, in which you set the connection parameters. In our case, this is the airline code.



You must now classify the SPS. The Organizer can use this classification to recognize the RMS and SP type (records, documents, business objects, and so on) in which the SPS needs to be displayed. We classify our SPS for the *Business Objects* type and declare it as valid for the RMS ID *S_RMS_DATA*.



When you restart the Records Organizer, the new SPS is visible and can be used.

8 Appendix: Implementing Short Texts

To make it easier to use the history function of the Records Organizer, you can label the new SP with a short text that helps the user to recognize the individual instances of an element type. To do this, you must implement an extra interface in the back -end class IF_SRM_SP_NON_VISUAL_INFO. You do not need the GET_SPECIFIC_INFO_LIST method (create an empty meth od body instead); the GET_STANDARD_INFO_LIST method is implemented as follows:

```
METHOD if_srm_non_visual_info_sp~get_standard_info_list .

* get poid parameters

  DATA: display_name TYPE string,
        carrid TYPE string,
        connid TYPE s_conn_id,
        fldate TYPE s_date,
        s_fldate TYPE string.

  carrid = me->get_connection_para( ).
  CALL METHOD me->get_sppoid_para
    IMPORTING
      ex_connid = connid
      ex_fldate = fldate.

* convert date to readable form

  CALL FUNCTION 'CONVERT_DATE_TO_EXTERNAL'
    EXPORTING
      date_internal = fldate
    IMPORTING
      date_external = s_fldate.


* set display name

  CONCATENATE text-001 carrid connid s_fldate INTO display_name
                                      SEPARATED BY space.

  ex_display_name = display_name.

ENDMETHOD.
```