SAP
Records Management

Tutorial:
Implementierung eines Service Providers

Dokumentation für Entwickler

18. Februar 2004

Inhaltsverzeichnis

1 Einleitung

Das vorliegende Dokument beinhaltet ein Tutorial für die Implementierung eines Service Providers. Vorausgestzt werden die Begriffe und die Architektur des Records Management Frameworks. Diese sowie eine systematische Darstellung der Service Provider Methoden finden Sie in der Records Management Referenzdokumentation für Entwickler. Wir empfehlen, das vorliegende Tutorial und die Referenzdokumentation parallel zu studieren.

2 Aufgabenstellung

Wir möchten einen Service Provider anlegen, der die Anzeige und Bearbeitung von Flügen (Tabelle SFLIGHT) einer Fluggesellschaft übernimmt. Der Service Provider soll dem Benutzer folgende visuelle Funktionen (in Records Management: Aktivitäten) anbieten:

- Suchen nach Flügen
- Anzeigen eines Fluges

In einem späteren Schritt werden wir den Service Provider um zusätzliche Funktionen erweitern.

3 Definition der CONNECTION- und SP-POID Parameter

Um einen Service Provider zu implementieren, benötigen wir zunächst Informationen über das *Repository*, also dem Ort, an dem die betriebswirtschaftlichen Daten des Service Providers abgelegt sein sollen. Flüge sind im SAP System in der Tabelle SFLIGHT abgelegt, unser Repository ist also die R/3 Datenbank. Um einen Flug eindeutig identifizieren zu können, müssen wir auf die Tabelle SFLIGHT über den *Primärschlüssel*
zugreifen. Den Aufbau des Primärschlüssels können wir der Tabellendefinition von SFLIGHT entnehmen:

- CARRID
- CONNID
- FLDATE

Die SP-POID eines Service Providers muß immer den für den Zugriff auf das Repository erforderlichen Primärschlüssel enthalten, mit Ausnahme der Teile, die bereits in den Connection-Parametern definiert sind. Für unser Beispiel nehmen wir an, daß die Fluggesellschaft in den Connection-Parametern definiert werden soll, Verbindung und Flugdatum spezifizieren wir in der SP-POID.

Hinweis: Der *Mandant* ist nicht Bestandteil der CONNECTION- oder SP-POID-Parameter, da er zur Laufzeit aus der Benutzeranmeldung ermittelt wird.

4 CONNECTION- und SP-POID Parameter publizieren

Die CONNECTION - und SP -POID-Parameter müssen dem Fram ework bekannt gemacht werden. Die Publikation erfolgt über die Implementierung einer ABAP -OO-Klasse, die einer bestimmten *Klassenrolle* genügt.

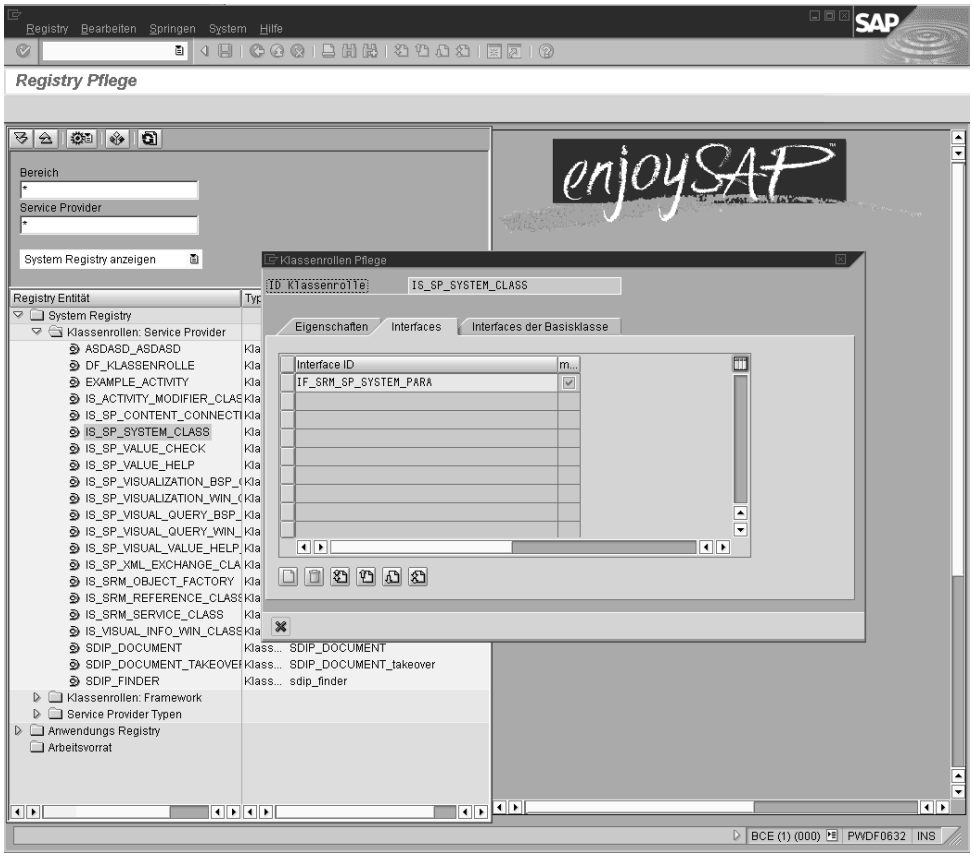
Einschub: Klassenrollen
Eine Klassenrolle definiert bestimmte Anforderungen, die an eine (ABAP -OO-) Klasse geste llt werden, damit sie in einer bestimmten Rolle auftreten (also: eine bestimmte Funktion erfüllen) kann. Eine Klassenrollendefinition legt fest, welche (ABAP-OO-) Interfaces eine Klasse implementieren muß, und von welcher Klasse geerbt werden muß.

Beispiel für eine Klassenrollendefinition:
Eine Klasse ist eine SP Client Klasse, wenn sie von CL_SRM_SP_CLIENT_OBJ erbt und das Interface IF_SRM_SP_CLIENT_WIN implementiert.

Hinweis:
Die Klasse muß nicht direkt von der in der Klassenrolle definierten Klasse er ben; es reicht aus, wenn die Klasse von einem Erben der Basisklasse erbt.

Klassenrollen werden über die *SRM Registry* (Transaktion SRMREGEDIT) verwaltet. Wenn man wissen möchte, welche Interfaces und welche Basisklasse eine bestimmte Klassenrolle verlangt, ist es empfehlenswert, dort nachzusehen.

Ein Blick in die Transaktion SRMREGEDIT zeigt uns die Definition der für das Publizieren der CONNECTION - und SP -POID Parameter notwendigen Klassenrolle IS_SP_SYSTEM_CLASS :



Auf der Registerkarte „Interfaces“ ist zu sehen, daß die Klassenrolle die Implementierung des Interfaces IF_SRM_SP_SYSTEM_PARA verlangt. Auf der (verdeckten) Registerkarte „Eigenschaften“ ist festgelegt, daß die Klassenrolle das Erben von CL_SRM, der Basisklasse aller RM-Programmobjekte, verlangt.

Mit Hilfe der Entwicklungsumgebung können wir jetzt die Systemklasse für unseren Service Provider anlegen:

- sie erbt von CL_SRM
- sie implementiert das Interface IF_SRM_SP_SYSTEM_PARA

Die Klassen des Tutorials existieren zur Referenz im Paket SRM_FRAMEWORK_DEMO. Die Systemklasse des Tutorial-SPs heißt CL_SRM_SP_TUTORIAL_SYSTEM.

Um IF_SRM_SP_SYSTEM_PARA zu implementieren, müssen wir folgende Methoden anlegen:

- Definition der CONNECTION-Parameter: GET_ATTR_DESC_CONNECTION
- Definition der CONTEXT-Parameter: GET_ATTR_DESC_CONTEXT
- Definition der SP-POID Parameter: GET_ATTR_DESC_SP_POID

Wichtiger Hinweis: Im ersten Schritt gibt es keine CONTEXT -Parameter, die Methode muß dennoch (leer) angelegt werden, da sonst Laufzeitfehler auftreten können.

In den M ethode IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_CONNECTION und IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_SP_POID publizieren wir die jeweils benötigten Parameter über *Attributbeschreibungsobjekte*.

Einschub: Das Factory Objekt
Über das Interface IF_SRM der Basisklas se sind eine Reihe von Funktionen zugänglich, die bei der Programmierung eines Service Providers benötigt werden. Ein Aufruf von IF_SRM~GET_SRM_OBJECT_FACTORY liefert eine Referenz auf das Interface IF_SRM_SRM_OBJECT_FACTORY zurück, welches Methoden zur Erzeugung von Framework-Objekten bereitstellt:

- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ACTIVITY_LIST
  Erzeugung eines Aktivitätslisten-Objekts
- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ATTR_DESC_*
  Erzeugung von Attributbeschreibungsobjekten (verschiedene Typen)
- IF_SRM_SRM_OBJECT_FACTORY~CREATE_ATTRIBUTE_VALUE
  Erzeugung von Attribut-Wert-Objekten

Einschub: Attributbeschreibungsobjekte
Attributbeschreibungsobjekte beinhalten die Definition eines Datentyps in Records Management. Ausprägungen dieser Definition sind in Attributwertobjekten enthalten.

Attributbeschreibungsobjekte können über IF_SRM_SRM_OBJECT_FACTORY beschafft werden. Es gibt verschiedene Typen von Attributbeschreibungsobjekten für verschiedene Zwecke, je nach gewünschtem Attributbeschreibungsobjekt muß die entsprechende Methode an IF_SRM_SRM_OBJECT_FACTORY gerufen werden:

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_ANY
  Beschafft ein generisches Attributbeschreibungsobjekt für verschiedene Zwecke.

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_CONNECTION
  Beschafft e in Attributbeschreibungsobjekt zur Definition von CONNECTION-Parametern.

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_CONTEXT
  Beschafft ein Attributbeschreibungsobjekt zur Definition von CONTEXT -Parametern.

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_SP_POID
  Beschafft ein Attributbeschreibungsobjekt zur Definition von SP -POID-Parametern.

- IF_SRM_SRM_OBJECT_FACTORY~GET_ATTR_DESC_INFO
  Beschafft ein Attributbeschreibungsobjekt zur Definition von INFO - Attributen.

Einschub: Attributbeschreibungsobjekte (Forts.)
Nachdem das Attributbeschreibungsobjekt beschafft worden ist, muß es über IF_SRM_EDIT_ATTRIBUTE_DESC gefüllt werden. Dies geschieht in zwei Schritten:

- Beschreibung der allgemeinen Eigenschaften:
  Aufruf von IF_SRM_EDIT_ATTRIBUTE_DESC~SET_GENERAL_DESCRIPTION mit einer Struktur vom Typ SRMADGEN:
  - TYPE: Festlegung des Typs:
    IF_SRM_ATTRIBUTE_DESC=>STRING
    IF_SRM_ATTRIBUTE_DESC=>INTEGER
    IF_SRM_ATTRIBUTE_DESC=>INTERFACE
  - IS_LIST: Attribut ist mehrfach bewertbar
  - IS_MAND: Attribut ist Pflichtfeld
  - IS_HELP: Für das Attribut existiert eine Wertehilfe
  - IS_CHECK: Für das Attribut existiert eine Werteprüfung
  - TEXT: Kurztext

- Beschreibung der typspezifischen Eigenschaften durch Aufruf einer der Methoden:
  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_STRING_DESCRIPTION
    Festlegung der spezifischen Eigenschaften für STRING-Attribute
  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_INTEGER_DESCRIPTION
    Festlegung der spezifischen Eigenschaften für INTEGER-Attribute
  - IF_SRM_EDIT_ATTRIBUTE_DESC~SET_INTERFACE_DESCRIPTION
    Festlegung der spezifischen Eigenschaften für INT ERFACE-Attribute

Für die Beschreibung der SP-POID-Parameter erhalten wir folgendes Coding:

```
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.
```

Das folgende Coding publiziert unseren 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.
```

Hinweis: Für SP -POID-Parameter ist keine Werteprüfung und -hilfe möglich. Für den CONNECTION-Parameter werden wir diese Funktionen in einem späteren Schritt implementieren.

5 Service Provider Backend implementieren

Nach der Publikation der SP -Parameter können wir mit der Implementierung des SP -Backends beginnen. Das Backend dient dem Frontend unseres Service Providers zum Zugriff auf das Repository.

Der Zugriff vom Fron tend auf das Backend sollte über ein Interface geschehen. Nur so ist es später möglich, die Backendklasse bei Bedarf auszutauschen.

Wir definieren also zunächst ein Interface zum Zugriff auf das Backend. Es enthält drei Methoden:

- get_flight_data Einlesen der Flugdaten (für Anzeige)
- get_flights Liefert Liste von Flügen (für Suchdialog)
- set_sppoid_para Setzen der SP -POID Parameter beim Übergang vom Modell zur Instanz

Zur Implementierung unseres SP -Backend verwenden wir das Interface IF_SRM_SP_TUTORIAL_BACKEND aus dem Paket SRM_FRAMEWORK_DEMO.

Das Backend eines Service Providers muß die Klassenrolle IS_SP_CONTENT_CONNECTION_CLASS erfüllen. Aus der RM Registry entnehmen wir die benötigten Daten:

- das Backend muß von der Klasse CL_SRM_SP_CONNECTION erben
- das Interface IF_SRM_CONNECTION muß implementiert werden
- das Interface IF_SRM_CONNECTION_NEW ist optional
- das Interface IF_SRM_CONTEXT_AUTOMATION ist optional
- das Interface IF_SRM_NON_VISUAL_INFO_SP ist optional

Ebenfalls implementieren müssen wir natürlich unser eigenes Interface, IF_SRM_SP_TUTORIAL_BACKEND.

Die Klasse kann nun über die Entwicklungsoberfläche angelegt werden: (Vorlage: CL_SRM_SP_TUTORIAL_BACKEND).

Wir legen zunächst eine private Methode an, die uns den Wert des Connection -Parameters (also die Fluggesellschaft) zurückliefert. Sie heißt GET_CONNECTION_PARA und hat als Rückgabewert RE_CARRID vom Typ S_CARR_ID. Beim Auslesen der Connection Parameter können Fehler auftreten, daher deklariert die Methode verschiedene Exception-Klassen:

Parametertyp	Name	Datenelement
RETURNING	RE_CARRID	S_CARR_ID
EXCEPTION	CX_SRM_INITIALIZATION	
EXCEPTION	CX_SRM_POID	
EXCEPTION	CX_SRM_ATTRIBUTE_VALUE	

Das Coding ist relativ simpel, es wird nur der Wert ausgelesen und zurückgegeben:

```
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.
```

Eine weitere privateMethode liefert uns die Schlüsselbestandteile aus der SP -POID. Sie heißt GET_SPPOID_PARA:

Parametertyp	Name	Datenelement
EXPORTING	EX_CONNID	S_CONN_ID
EXPORTING	EX_FLDATE	S_DATE
EXCEPTION	CX_SRM_INITIALIZATION	
EXCEPTION	CX_SRM_POID	

Hier ist das Coding auch nicht komplizierter:

```
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.
```

Mit Hilfe dieser Methoden können wir jetzt die Methoden unseres Interfaces IF_SRM_SP_TUTORIAL_BACKEND codieren:

Die Methode GET_FLIGHTS liest alle Flüge der in den Connection -Parametern definierten Fluggesellschaft und gibt sie in einer internen Tabelle zurück:

```
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.
```

Die Methode GET_FLIGHT_DATA liest einen Flug aus der Tabelle und gibt ihn in einer Struktur zurück. Wird der über SP -POID und Connection Parameter gesetzte Flug nicht gefunden, wird eine Ausnahme vom Typ CX_SRM_CONNEC_FAILED (Verbindung fehlgeschlagen) ausgelöst:

```
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.
```

Die Methode SET_SPPOID_PARA nimmt die SP POID Parameter aus der Suche entgegen und setzt die Werte am POID Objekt:

```
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.
```

Einschub: Exception Handling
Im Records Management Framework werden Ausnahmen über Exception -Klassen behandelt. Wie die alten Exceptions werden Exception -Klassen an der Schnittstelle einer Methode deklariert. Neu ist, daß Exceptions automatisch nach oben propagiert werden, wenn sie an der Schnittstelle der aufrufenden Methode ebenfalls deklariert worden sind.
Da an der Schnittstelle fast aller Methoden, die ein Service Provider implementiert, die beiden wichtigsten Excep tion-Klassen CX_SRM_FRAMEWORK und CX_SRM_SP_CLIENT deklariert sind, braucht man sich als SP -Programmierer in der Regel nicht um die Behandlung von Ausnahmen zu kümmern.
Das Framework behandelt die Ausnahmen automatisch und speichert sie im Application Log; über die Transaktion SLG1 können die gespeicherten Fehlermeldungen betrachtet werden.
(Hinweis: Das Logging muß einmalig über die Customizing -Transaktion SRM_APPL_LOG aktiviert werden).

Zum Schluß müssen noch die Methoden von IF_SRM_CONNECTION implementiert werden:

- IF_SRM_CONNECTION~INITIALIZE
  In dieser Methode erfolgt die Initialisierung der Verbindung zum Backend. Dies ist in unserem Fall nicht notwendig, da die Verbindung zur Datenbank auf dem Web Application Server immer besteht. Es reicht also, ein en leeren Methodenrumpf anzulegen.

- IF_SRM_CONNECTION~CHECK
  Diese Methode dient der Prüfung, ob die Verbindung zum Repository noch besteht. Da dies auf dem Web Application Server immer der Fall ist, legen wir hier ebenfalls nur einen leeren Methodenrumpf an.

- IF_SRM_CONNECTION~CONNECT_REPOSITORY
  Hier konnektieren wir das Repository und prüfen, ob der über Connection - Parameter und SP -POID spezifizierte Datensatz existiert. Diese Prüfung geschieht, indem wir einfach versuchen, den Datensatz zu lesen. Sollte das Lesen nicht möglich sein, wird von der Methode GET_FLIGHT_DATA automatisch eine Ausnahme ausgelöst:

```
METHOD if_srm_connection~connect_repository .

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

ENDMETHOD.
```

6 Implementierung eines SP Frontend für SAPGUI

Für das SP Frontend sind zwei Klassenrollen maßgeblich:

- IS_SP_VISUALIZATION_WIN_CLASS: Klassenrolle für die Anzeige eines Elements
- IS_SP_VISUAL_QUERY_WIN_CLASS: Klassenrolle für visuellen Suchdialog

Diese Klassenrollen können entweder gemeinsam von einer oder von zwei separaten Klassen erfüllt werden. Da in unserem Beispiel keine Wiederverwendung des Suchdialoges geplant ist, kann eine gemeinsame Klasse verwendet werden.

Aus der Registry entnehmen wir die Anforderungen an die Klassen:

Klassenrolle IS_SP_VISUALIZATION_WIN_CLASS:
erbt von CL_SRM_SP_CLIENT_OBJ
implementiert

- IF_SRM_SP_ACTIVITIES Publikation der visuellen Aktivitäten
- IF_SRM_SP_AUTHORIZATION Berechtigungsprüfung
- IF_SRM_SP_CLIENT_WIN Ausführen von Aktivitäten (inplace)
- IF_SRM_SP_CLIENT_OUTPLACE optional: Ausführen von Aktivitäten (outplace)

Klassenrolle IS_SP_VISUAL_QUERY_WIN_CLASS
Erbt von CL_SRM_SP_CLIENT_OBJ
Implementiert IF_SRM_SP_VISUAL_QUERY_WIN

Wir legen also nun eine Klasse an (CL_SRM_SP_TUTORIAL_FRONTEND), die von CL_SRM_SP_CLIENT_OBJ erbt, und implementieren die fünf Interfaces:

6.1.1 Berechtigungsprüfung: IF_SRM_SP_AUTHORIZATION

Im ersten Schritt wollen wir keine separate Berechtigungsprüfung einbauen, wir geben daher immer die Konstante IF_SRM=>TRUE zurück:

```
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 Aktivitäten publizieren: IF_SRM_SP_ACTIVITIES

Einschub: Aktivitäten
Modellaktivitäten sind Aktivitäten, die sich auf die Elementart (den SPS) beziehen, z.B. Suchen, Anlegen etc., wogegen *Instanzaktivitäten* sich auf ein konkretes Element beziehen, z.B. Anzeigen, Löschen.

Standardaktivitäten sind Aktivitäten, die für eine Vielzahl von Service Providern eine gleiche semantische Bedeutung haben. Sie werden von SAP festgelegt und können vom Kunden nicht erweitert werden. Alle Standardaktivitäten sind als Konstanten an IF_SRM_ACTIVITY_LIST angelegt:

IF_SRM_ACTIVITY_LIST=>

CREATE	Modellaktivität	Anlegen eines Elements
QUERY	Modellaktivität	Suche nach Element
DISPLAY	Instanzaktivität	Anzeigen eines Elements
EDIT	Instanzaktivität	Anz. im Änderungsmodus
DELETE	Instanzaktivität	Löschen eines Elements
INFO	Modellaktivität	Anzeigen des Informationsdialogs (wird intern behandelt)
INFO	Instanzaktivität	Anzeigen des Informationsdialogs (wird intern behandelt)
PROTOCOL	Instanzaktivität	Anzeigen eines elementbezogenen Protokolles

Die Publikation von Aktivitäten erfolgt über das –von jedem SP Frontend implementierte- Interface IF_SRM_SP_ACTIVITIES, mit Hilfe des Aktivitätslisten -Objekts (Klasse mit Interface IF_SRM_ACTIVITY_LIST). Das Aktivitätslisten -Objekt wird über die Factory besorgt.

Service Provider können neben den Standardaktivitäten auch spezifische Aktivitäten besitzen (z.B. „Beleg verbuchen“). Bei spezifischen Aktivitäten muß sowohl ein Funktionscode als auch ein Kurztext angegeben werden (siehe IF_SRM_ACTIVITY_LIST->ADD_ACTIVITY)

Aktivitätslisten können geschachtelt werden, indem über IF_SRM_ACTIVITY_LIST->ADD_ACTIVITY_LIST ein weiteres Aktivitätslisten-Objekt eingefügt wird.
Default-Aktivitäten können ausgeführt werden, wenn keine weitere Benutzerinteraktion gewünscht oder möglich ist, z.B. bei Doppelklick auf ein Element.

Unser Service Provider hat im ersten Schritt zwei Aktivitäten: IF_SRM_ACTIVITY_LIST=>DISPLAY und IF_SRM_ACTIVITY_LIST=>QUERY. Wir setzen diese Aktivitäten jeweils als Default-Aktivität:

```
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 Ausführen von Aktivitäten: IF_SRM_SP_CLIENT_WIN

IF_SRM_SP_CLIENT_WIN enthält verschiedene Methoden ,die im Zusammenhang mit der Ausführung von Aktivitäten aufgerufen werden:

- IF_SRM_SP_CLIENT_WIN~GET_EVENT_OBJECT
  Auslesen des Event Objektes durch Client Framework. Diese Methode hat eine Standardimplementierung, die vom SP eingebaut werden muß:
  event_object = me->if_srm_sp_client_win~event_object.

- IF_SRM_SP_CLIENT_WIN~SET_EVENT_OBJECT
  Setzen des Event Objektes durch Client Framework. Diese Methode hat eine Standardimplementierung, die vom SP eingebaut werden muß:
  me->if_srm_sp_client_win~event_object = im_event_object.

- IF_SRM_SP_CLIENT_WIN~OPEN
  Wird beim Öffnen des SP aufgerufen. Zu diesem Zeitpunkt können interne Initialisierungen des Client vorgenommen werden, z.B. Aufbau von Controls.

- IF_SRM_SP_CLIENT_WIN~MY_ACTION
  Wird zum Ausführen einer Aktivität aufgerufen.

- IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WITDH
  Liefert die Darstellungsbreite zu einer Aktivität, zwischen 0% (für nichtvisuelle Aktivitäten) und 100%.

- IF_SRM_SP_CLIENT_WIN~ANSWER_ON_EVENT
  Wird vom Framework aufger ufen, um eine asynchrone Antwort auf einen Request dem Sender zuzustellen.

- IF_SRM_SP_CLIENT_WIN~SYSTEM_INFO
  Dient der Zustellung von Systemnachrichten, z.B. Beendigung des Frameworks.

Die Methoden ANSWER_ON_EVENT und SYSTEM_INFO benötigen wir zunächst n icht, sie werden daher nur leer angelegt. Diese Methoden dienen der Benachrichtigung auf Systemevents und der Zustellung von Antworten auf asynchrone Requests. Da der Service Provider für Flüge keine Requests versendet, kann er auch keine asynchronen Antworten erhalten. Systemevents sind für uns irrelevant, da der Service Provider keine

Daten verändert und daher auf die Beendigung des Framework auch nicht reagieren muß.

Die Methoden GET_EVENT_OBJECT und SET_EVENT_OBJECT werden jeweils mit der Standardimplementierung ausgefüllt:

```
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.
```

Die Methode GET_CLIENT_WIDTH dient insbesondere der Feststellung, ob eine bestimmte Aktivität eine Inplace-Darstellung zur Folge hat. Der zurückgelieferte Wert legt die vom Service Provider gewünschte Darstellungsbreite (von 0 – 100%) fest. Unsere Aktivität „Anzeigen“ hat eine Inplace-Darstellung, daher geben wir den Wert 100 zurück:

```
method IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH .

  re_client_width = 100.

endmethod.
```

6.1.4 Methoden zur Darstellung der Fluginformationen

Für die eigentliche Darstellung der Fluginformationen werden dynamische Dokumente sowie das ALV-Grid-Control verwendet. Da die Programmierung dynamischer Dokumente und des ALV Grid Controls nicht Teil des Tutorials ist, sollten die entsprechenden Methoden über die Entwicklungsumgebung aus der Vorlagenklasse kopiert werden:

- 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 Erzeugen der Visualisierung: IF_SRM_SP_CLIENT_WIN~OPEN

Wird ein Service Provider zum ersten Mal angezeigt, erfolgt ein Aufruf der Methode IF_SRM_SP_CLIENT_WIN~OPEN.An dieser Stelle sollte die Erzeugung der für die Visualisierung notwendigen Controls (im Beispiel in der privaten Methode BUILD_VISUALIZATION verschalt) stattfinden. Der Service Provider bekommt vom Client Framework eine Referenz auf einen Control-Container, und muß seinerseits wieder einen Zeiger auf den eigenen, obersten Container zurückliefern (diesen benötigt das Client Framework, um die Visualisierung eines SP komplett aus- und einschalten zu können):

```
method IF_SRM_SP_CLIENT_WIN~OPEN .
  re_main_control = build_visualization( im_parent ).
endmethod.
```

6.1.6 Ausführen einer Aktivität: IF_SRM_SP_CLIENT~MY_ACTION

Soll ei ne z.B. im Organizer oder in der Akte per Kontextmenü ausgewählte Aktivität ausgeführt werden, so ruft das Client Framework den Service Provider über die Methode IF_SRM_SP_CLIENT~MY_ACTION. Anhand der im Request-Objekt enthaltenen Aktivität muß der Service Provider nun zunächst entscheiden, welche Aktivität ausgeführt werden soll. Nach Ausführung der Aktivität (in diesem Tutorial in der privaten Methode DISPLAY_FLIGHT verschalt) muß der Service Provider am Request Objekt das Ergebnis der Aktivität (eine POI D) sowie den Zustand der Aktivität (Konstanten an IF_SRM_REQUEST=>ACTIVITY_STATE...) setzen:

```
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.
```

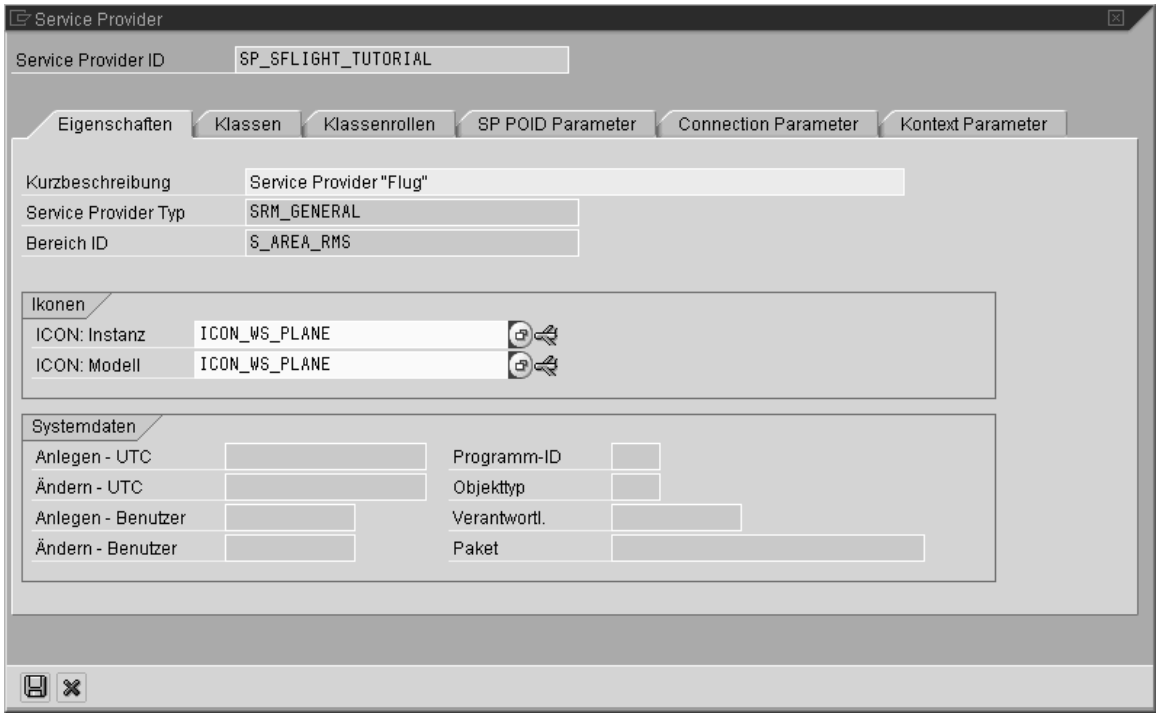
7 Registrieren des Service Providers

Der Service Provider ist nun lauffähig. Um ihn benutzen zu können, muß er in der RM Registry (Transaktion SRMREGEDIT) registriert werden.

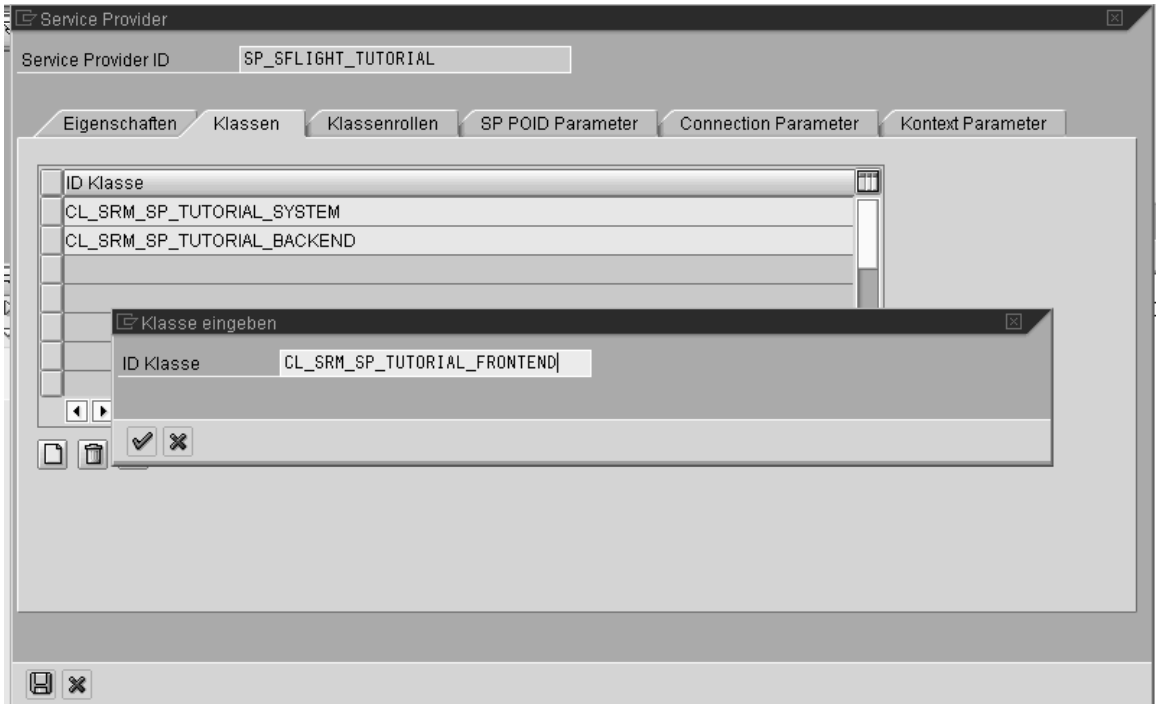
Nach dem Start d er Transaktion wird der Knoten S_AREA_RMS unterhalb des Knotens „Anwendungs Registry“ selektiert und aus dem Kontextmenü der Befehl „Service Provider anlegen“ ausgeführt:

Als Service Provider Typ ist immer SRM_GENERAL auszuwählen; die anderen Service Provider Typen dienen speziellen Zwecken. Nach der Namensvergabe erscheint ein Dialog mit mehreren Registerkarten.

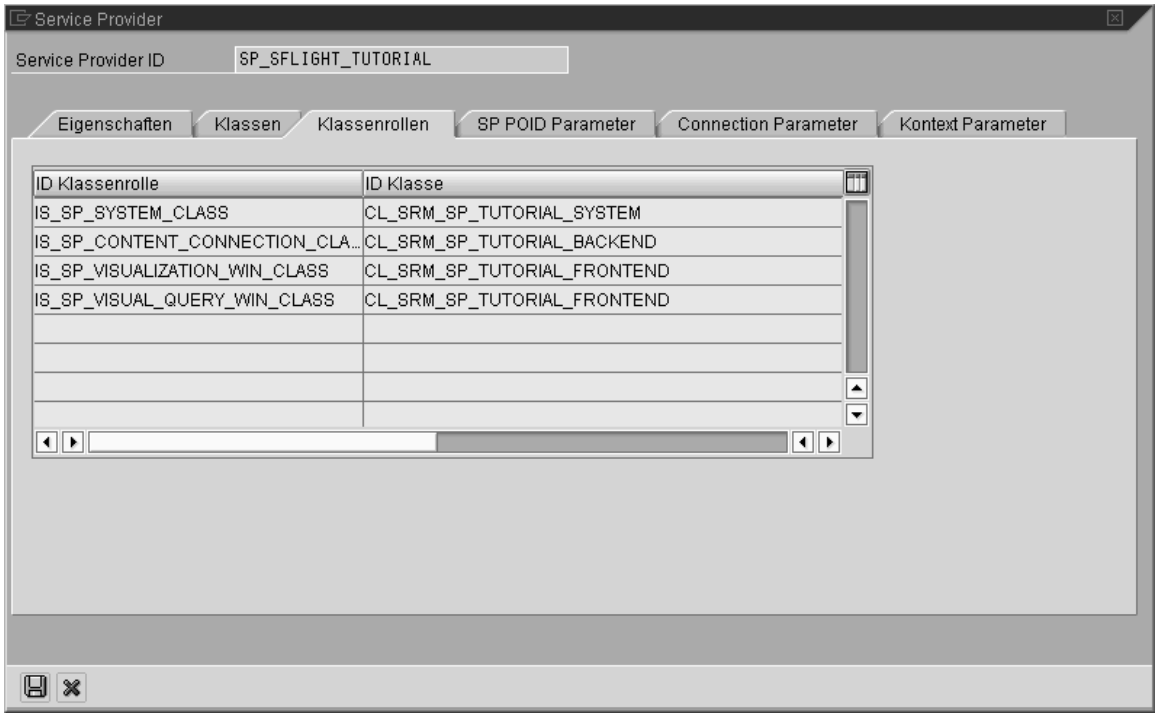
Auf der Registerkarte „Eigenschaften“ können die Ikonen zur Darstellung in Akte und Organizer vergeben werden:



Auf der Registerkarte „Klassen“ müssen die drei im Verlauf dieses Tutorials erzeugten Klassen angegeben werden :

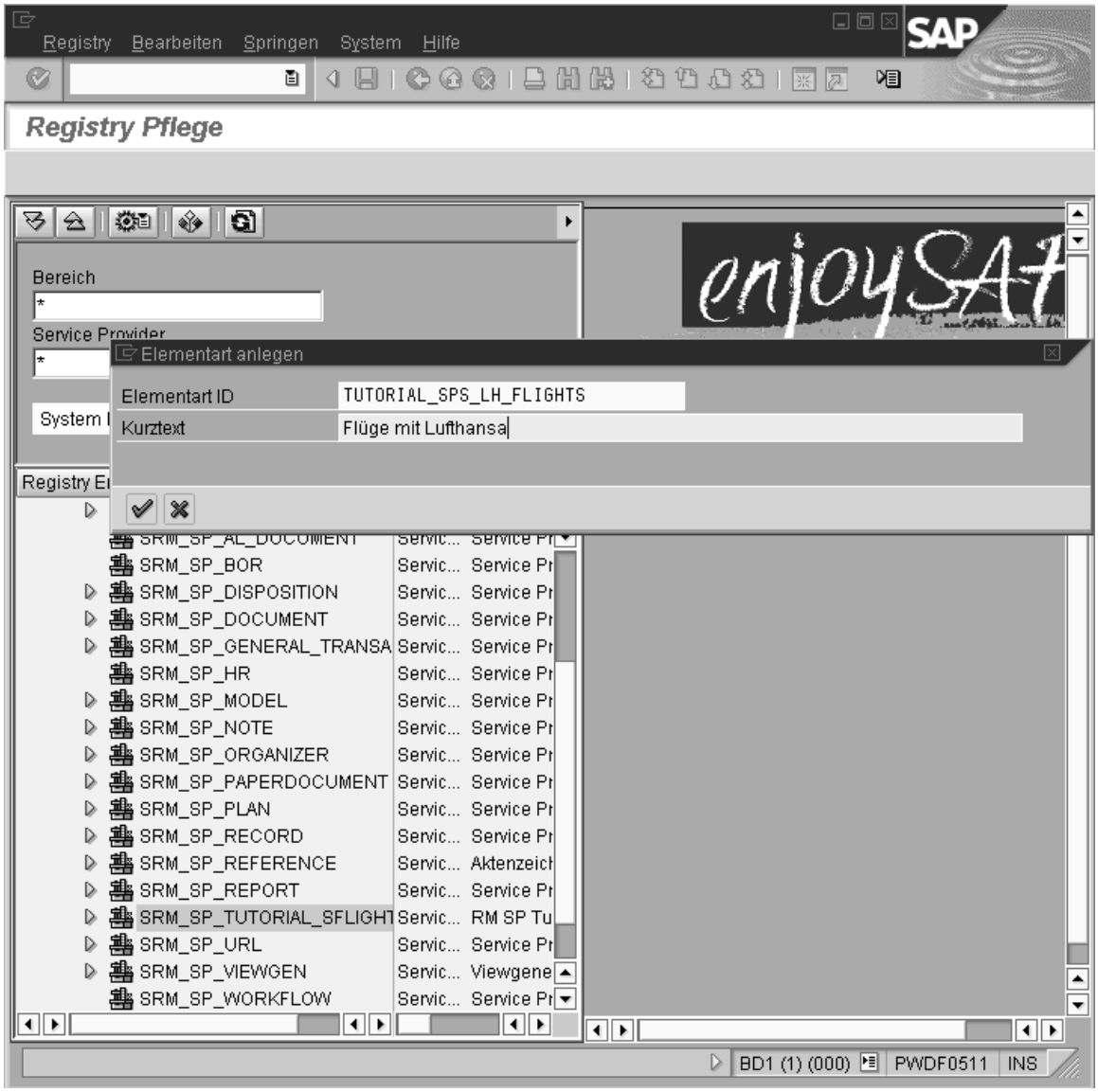


Anschließend kann auf der Registerkarte „Klassenrollen“ geprüft werden, ob auch alle benötigten Klassenrollen erfüllt werden:

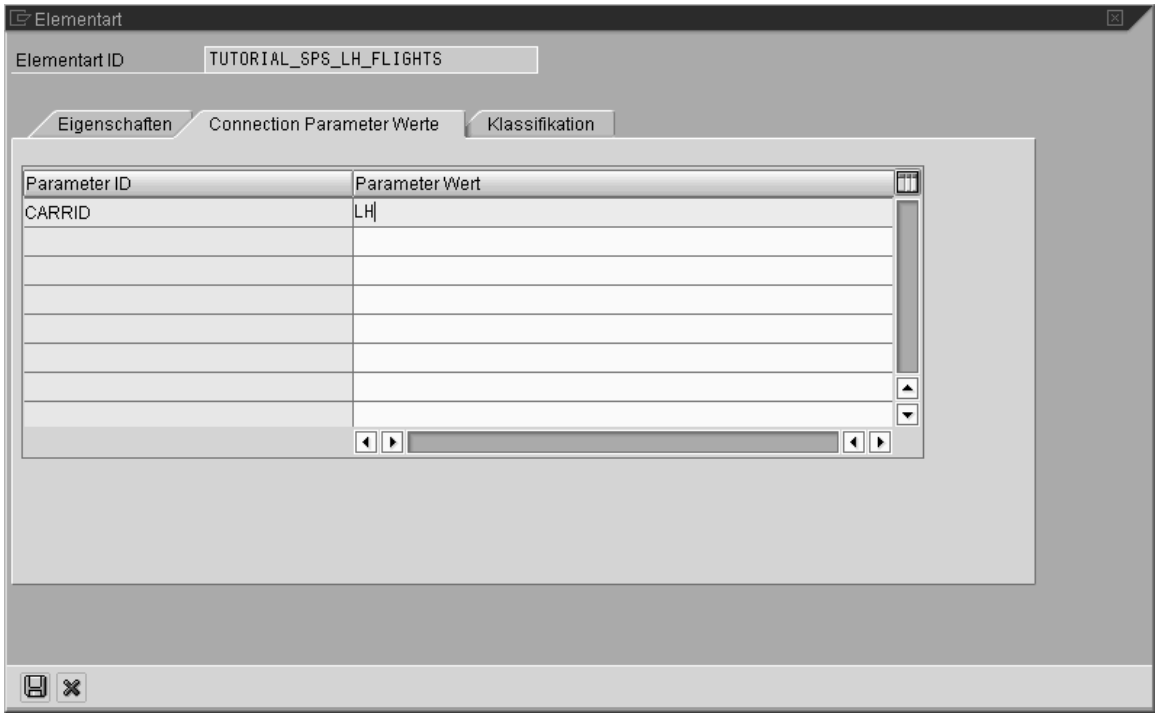


Auf den Registerkarten „SP POI D Parameter“ und „Connection Parameter“ sind die jeweils publizierten Parameter eingetragen (ohne Abbildung).

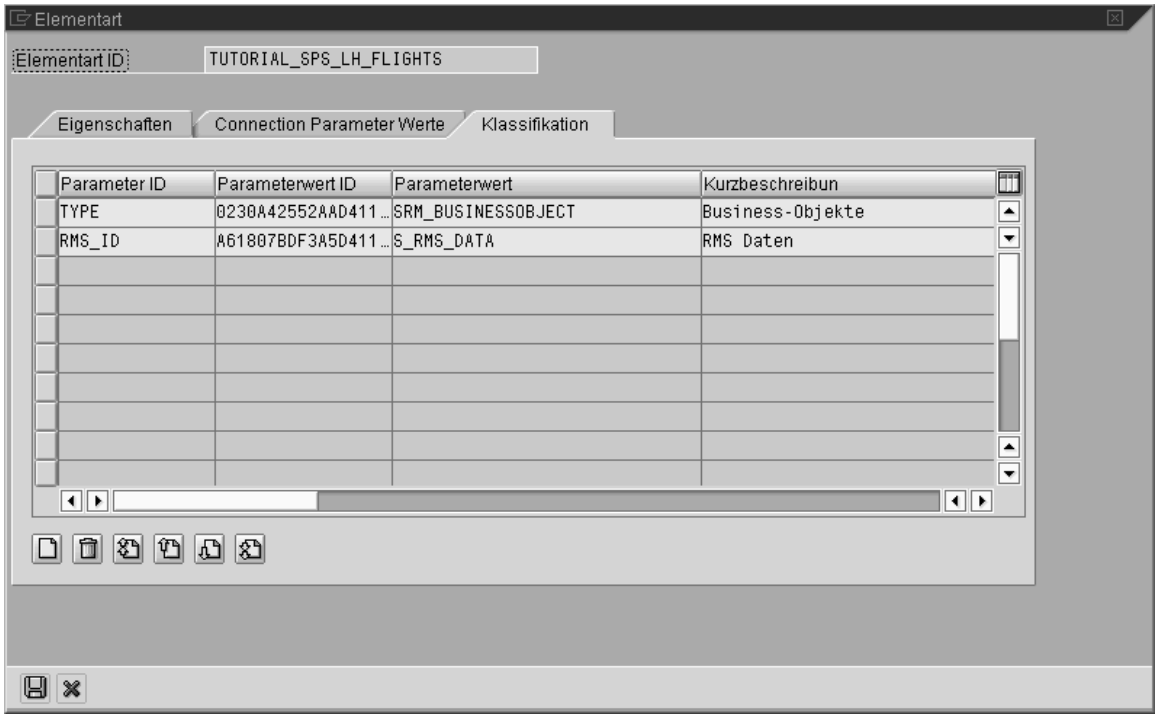
Nachdem der Service Provider erfolgreich angelegt wurde, muß noch ein SPS (eine Elementart) angelegt werden. Hierzu wählt man den Knoten des neu angelegten Service Providers aus und ruft über das Kontextmenü den Befehl „Elementart anlegen“ auf:



Es erscheint ein Dialog, in dem die Connection Parameter gesetzt werden müssen, in unserem Fall die Abkürzung der Fluglinie:



Der SPS muß jetzt n och klassifiziert werden. Anhand der Klassifikation kann der Organizer erkennen, in welchem RMS und in welcher SP -Art (Akten, Dokumente, Business-Objekte etc.) der SPS angezeigt werden soll. Wir klassifizieren unseren SPS für die Art „Business -Objekte“ und deklarieren ihn als gültig für die RMS ID S_RMS_DATA(„RMS Daten“):



Nach einem Neustart des Records Organizer ist der neue SPS jetzt sichtbar und kann benutzt werden:


RMSystem Bearbeiten Springen System Hilfe
Records Management System
Rollenbasierte Sicht
Favoriten
Aktivitäten
RMS Daten
Business-Objekte
Flüge LH
Transaktionen
Ablagepläne
Akten
Aktenmodelle
Dokumente
Berichte
Historie
Aktivitäten
erste Akte
Flug LH 0454 17.11.1995
Records
Management
SAP
BD1 (1) (000)
PWDF0511
INS

8 Zusatz: Kurztexte implementieren

Um die Historienfunktion des Records Organiz er übersichtlicher zu machen, kann der neue SP noch mit einem Kurztext ausgestattet werden, der dem Anwender ermöglicht, die einzelnen Instanzen einer Elementart besser zu unterscheiden. Hierzu muß an der Backend-Klasse ein zusätzliches Interface implement iert werden: IF_SRM_SP_NON_VISUAL_INFO. Die Methode GET_SPECIFIC_INFO_LIST wird nicht benötigt (leeren Methodenrumpf anlegen!), die Implementierung der Methode GET_STANDARD_INFO_LIST sieht folgendermaßen aus:

```
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.
```