SAP
Records Management

# Records Management Reference Documentation

Documentation for Developers

May 12, 2004

# Contents

# 1 Introduction

The service provider framework enables different applications to communicate with each oth er so that the sum of the applications appears externally as one logical unit, although the individual applications do not communicate directly with each other.

Applications are integrated into the framework by the implementation of interfaces. The framework provides the applications with services in the form of service classes.

## 1.1 How To Use This Documentation

This documentation is aimed at developers who have sound knowledge of ABAP objects.

Unit 1 defines the most important terms of the Records Management F ramework, and provides an overview of the architecture. This unit is important for the understanding of the subsequent units.

Unit 2 refers to the development of a service provider and describes the methods you need to implement to enable your new service provider to run in the framework.

Unit 3 refers to the development of a service provider and describes the methods implemented by the framework that are available through the inheritance hierarchy of your service provider classes.

Unit 4 describes the obje cts of the framework that you can request, and at which you can call further framework methods. You will probably not need all of these methods. This unit provides an overview only.

Unit 5 describes the procedure for working with attribute description obje cts and attribute value objects. A service provider defines its parameters using attribute value objects and attribute description objects instead of in the Data Dictionary. It is therefore important that you understand this technology.

Unit 6 describes pr operty unification, a service with which a service provider can publish its attributes and display them in the standard search dialog and standard attribute maintenance dialog.

Unit 7 describes optional framework services:

Implementing the input help enabl es you to assign values that are stored in the registry to users, or to make parameter values of another service provider available in input help. You can implement the value check to check that the user entries exist.

Implementing logging enables you to l og all the activities a performed by a particular user on elements and subelements of the service provider.

## 1.2 Definition of Terms

The following terms are central to the development of service providers for integration into the framework. For more term defini tions, see the SAP Library under *SAP NetWeaver Components → SAP Records Management → Introduction to Terms*.

**Class role**

A class role is an object -oriented interface definition for classes that serve a specific purpose

("role"). From a technical perspective, a class role is defined as a set of interfaces. Interfaces are defined either as optional or as obligatory. A class role is fulfilled when all obligatory interfaces have been implemented. The optional interfaces do not have to be implemented to fulfill th e class role. The class roles used by Records Management are supplied by SAP.

**Service Provider (SP)**

A service provider must be implemented for each application that is integrated in the framework. Service providers enable integration of elements into Reco rds Management and access to elements.

From a technical perspective, a service provider is defined by:

- a set of ABAP OO classes
- a set of SP POID parameter definitions (see below)
- a set of SP connection parameter definitions (see below)
- a set of SP context parameter definitions (see below)

**SP POID Parameters**

(SP stands for service provider, POID stands for **P**ersistent **O**bject **ID**.

Parameters required by the SP for the unique addressing of one of its own data objects in its repository.

An example from the SP for transactions is a transaction code.

**Connection Parameters**

Parameters required by the SP for addressing its content repository.

An example from the SP for transactions is the RFC destination of the target system.

**Context Parameters**

Optional parameters that the caller can set when making a request, but which are not obligatory.

An example for the se rvice provider for transactions is the SPA/GPA parameter (filling values when the transaction is called).

**Element Type/Service Provider Space (SPS)**

An element type is a division of the elements of a service provider, based on the values entered for certain parameters. It is formed by entering values for the connection parameters of the SP. You can only use a service provider if at least one element type exists in this service provider. A service provider can contain any number of element types. You can adap t element types in customizing according to customer-specific requirements.

Technically, an element type is defined by:

- exactly one SP
- a set of connection parameter values

Note: Element type and service provider space are used as synonyms. Element type is the user-oriented term, and service provider space is the technical term.

**Area**

Records Management has an underlying framework that can also be used for other applications.

Each application has its own framework area. For Records Management, only the ar ea S_AREA_RMS is relevant. This has, among others, the parameters RMS (see below) and type. These parameters classify the element types within the area. The RMS is the most important element for service provider development.

**Records Management System (RMS)**

An RMS is a logical structure entity within Records Management. The records of a company or institution can be divided into more than one RMS (similar to a client or company code). This division means that is it possible to provide particular groups of us ers access to particular records. You separate the different RMSs by assigning element types to one (or more) RMS. An RMS can contain any number of element types.

From a technical perspective, an RMS is defined by a group of element types.

## 1.3 Architecture

**Communication Using the Framework**

The framework can integrate diverse elements and bring them into contact with each other. The different elements communicate with each other only via the framework. General rules for integration:

- SP objects that communicate with the framework are instanced only by the framework.
- SP development does not program using classes, only interfaces. Exceptions to this are cases where programming is performed internally in the SP.
- Service classes for SPs are never instanced by an SP. Instances of the classes are always called using methods of already instanced objects.

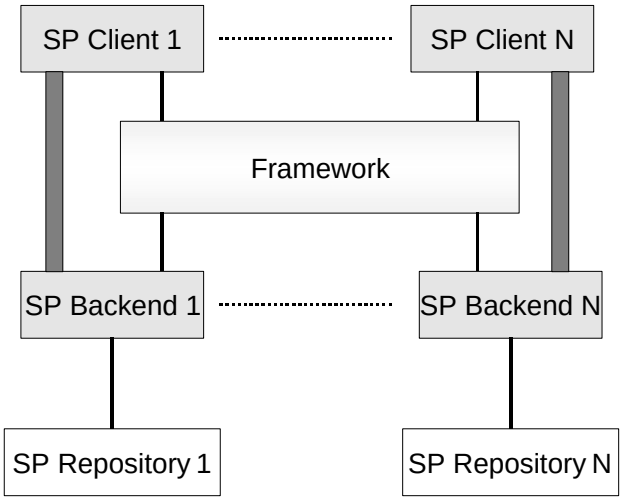



**Figure 1** The Service Provider Communication Framework

**The POID Object**

The POID object is an abstract reference to an element from th e repository of a service provider. The POID object uniquely identifies this element.

POID objects can have different technical representations. Regardless of this technical representation, the POID object is always made up of exactly three information co mplexes:

- SRM POID
- AREA POID
- SP POID

**SRM POID**

The SRM POID is the addressing schema for an element type. The SRM POID is used by the framework to identify the responsible SP and the responsible element type. It is composed of a name-value pair for the element type ID (SPS ID), and a name -value pair for the element type status.

**AREA POID**

The AREA POID is an addressing schema for an element type. It is composed of a name -value pair for the RMS ID.

**SP POID**

The SP POID is an addressing schema for an indivi dual element type. It is used by the SP to identify its SP repository objects. The SP POID is composed of a two -column internal table of name-value pairs (table type SRM_LIST_POID, line type SRMPOID) for the POID parameters of an SP.

The following restrictions apply to the addressing schema of the SP POID:

- The value components of the SP POID are of the data type String.
- The value components of the SP POID can only be assigned single values (there are no multiple-value components)
- Names can be a maximum of 32 characters long.
- Values can be a maximum of 10,000 characters long.

**Example of a POID Object from the SP for Documents:**

SRM POID:

| **Name** | **Value** |
|---|---|
| SPS ID | SRM_SPS_DOCUMENT |
| STATE | INSTANCE |

AREA POID:

| **Name** | **Value** |
|---|---|
| RMS ID | S_RMS_DATA |

SP POID:

| **Name** | **Value** |
|---|---|
| DOCID | SRMDOC04 90879147598374509860768097 34 |
| VERSION | 0 |
| VARIANT | 0 |

**POID Object at Runtime**

At runtime, the framework provides the POID object as an ABAP OO runtime object. Read and write access is available through the interface IF_SRM_POID. For SP dev elopment, it does not matter which class is involved, because the object is accessed only through the interface, and the instances are always provided by the framework.

A POID object can have the state *model POID* or *instance POID*.

A model POID is an elemen t that has not yet been identified, only the SPS ID and the RMS ID are known. Activities such as *Create* or *Find* are offered for model POIDs. If an element is uniquely identified (that is, if the SP POID is set), then the model POID becomes an instance POID. Activities such as *Change* or *Delete* are offered for instance POIDs.

# 2 SP Development: Methods for Implementation

This section describes the interface methods that you have to implement if you want to develop a new service provider. You develop a new servi ce provider if you want to integrate objects in a file that cannot be integrated using the service provider that is delivered as standard.

## 2.1 Overview of Service Provider Class Roles

A service provider must fulfill class roles specified by the framework. To fulfill a class role, you need to declare a class that inherits from a specified framework class and that implements the obligatory interfaces of the class role. Direct inheritance is not required. The class can also inherit from any other class that has originally inherited from the framework class.

The class roles are listed in the registry maintenance transaction (transaction SRMREGEDIT), under the *System Registry* node.

The following is a list of the class roles required for the service provider implementation. You must implement the obligatory interfaces of the obligatory class roles before your service provider can run in the framework.

- IS_SP_SYSTEM_CLASS
  - Implementation of the class role using the SP is obligatory.
  - Function: Publishes SP parameters
  - The SP class inherits directly or indirectly from the framework class CL_SRM.
  - The class role includes the following interfaces:
    - IF_SRM_SP_SYSTEM_PARA (obligatory)
    - IF_SRM_SP_INFO (optional)

- IS_SP_CONTENT_CONNECTION_CLASS
  - Implementation of the class role using the SP is obligatory.
  - Function: Connection to the SP repository
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_CONNECTION.
  - The class role includes the following interfaces:
    - IF_SRM_CONNECTION (obligatory)
    - IF_SRM_NON_VISUAL_INFO_SP (optional)
    - IF_SRM_CONNECTION_NEW (optional)
    - IF_SRM_CONTEXT_AUTOMATION (optional)
    - IF_SRM_SP_ENQUEUE (optional)
    - IF_SRM_SP_TRANSIENT_UPDATE (optional)
    - IF_SRM_SP_VERSION (optional)

- IS_SP_VISUAL_QUERY_WIN_CLASS
  - Implementation of the class role using the SP is optional (but is obligatory if using SAPGUI visualization)
  - Function: Visual search for WINGUI
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_CLIENT_OBJ.
  - The class role includes the interface:
    - IF_SRM_SP_VISUAL_QUERY_WIN (obligatory)

- IS_SP_VISUALIZATION_WIN_CLASS
  - Implementation of the class role using the SP is optional (but is obligatory if using SAPGUI visualization)
  - Function: WINGUI Visualization
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_CLIENT_OBJ.
  - The class role includes the following interfaces:
    - IF_SRM_SP_CLIENT_WIN (obligatory)
    - IF_SRM_SP_ACTIVITIES (obligatory)
    - IF_SRM_SP_AUTHORIZATION (obligatory)
    - IF_SRM_SP_CLIENT_OUTPLACE (optional)
    - IF_SRM_SP_MENUE (optional)

The graphic “Inheritance Hierarchy of SAP Classes“ shows an example of the inheritance hierarchy of the service provider classes to implement. The framework classes are displayed in color; their interfaces are already implemented. The service provider classes are not displayed in color; their interfaces must be implemented by the SP developer.

In the class diagram, one class is implemented to fulfill each class role. This class inherits directly from a framework class. This division of classes and their nomenclature is not prescribed. There are two rules for the division of class roles into classes:

1. A class role must be completely fulfilled by one class (it is not possible to distribute between multiple classes).
2. A class can contain any number of class roles, as long as this is possible according to the inheritance hierarchy of the basis classes.

For more information on the methods of the most important interfaces for these class roles, see sections 2.2 and 2.3.

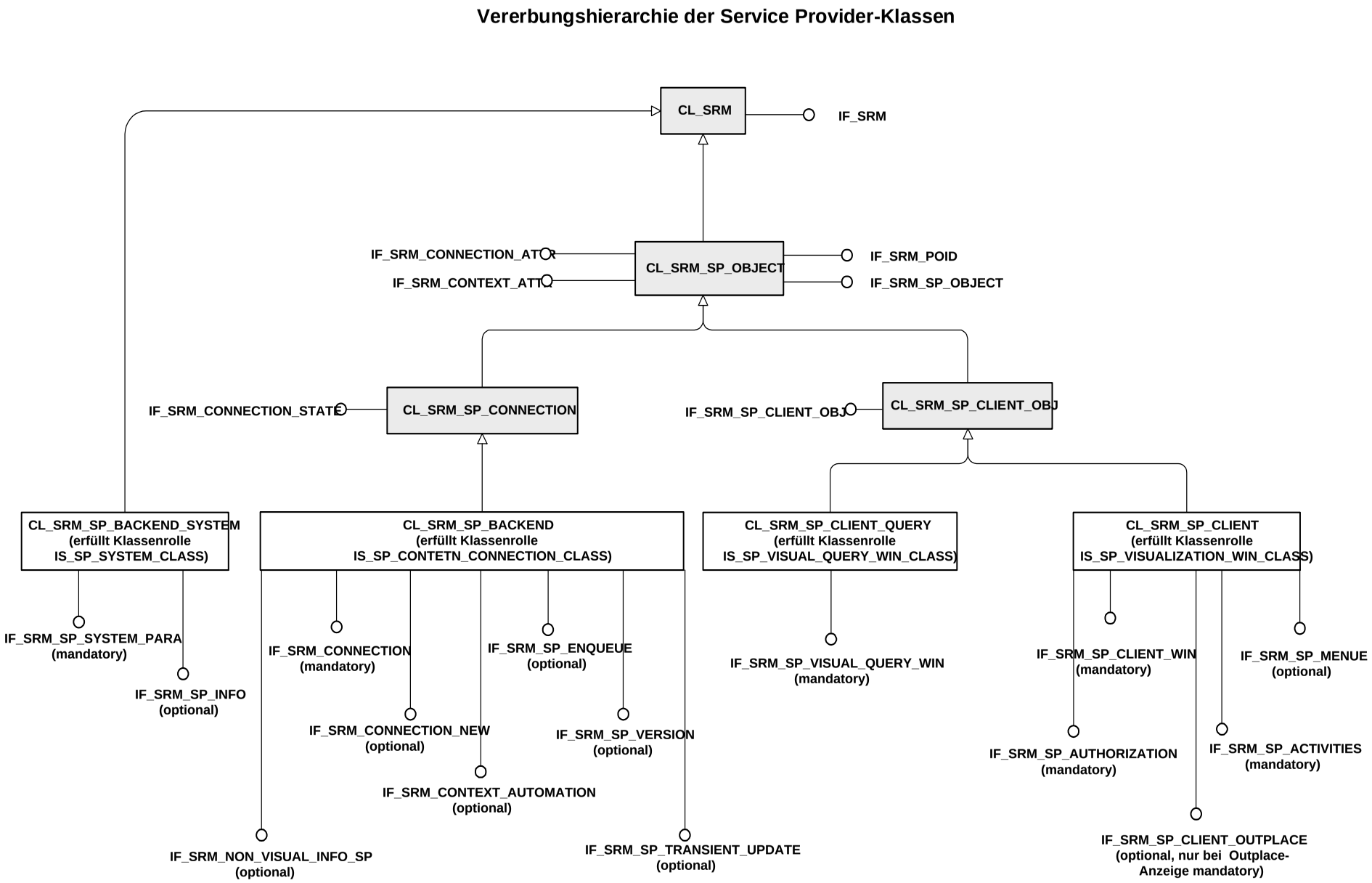



**Figure 2** Inheritance Hierarchy of the SP Classes

The following additional class roles can be fulfilled by an SP to enable use of optional framework services:

- IS_SP_FRONTEND_SAPGUI_PASV
  - Implementation of the class role using the SP is optional.
  - Function: Displaying the SP in passive mode
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_CLIENT_OBJ.
  - The class role includes the following interfaces:
    - IF_SRM_SP_ACTIVITIES (obligatory)
    - IF_SRM_SP_AUTHORIZATION (obligatory)
    - IF_SRM_SP_FRONTEND_SAPGUI_PASV (obligatory)

For more information, see *Calling an SP in a Passive Mode* on page 32.

- IS_SP_VALUE_HELP_WIN
  - Implementation of the class role using the SP is optional.
  - Function: SP input help
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_VALUE HELP.
  - The class role includes the interface:
    - IF_SRM_SP_VALUE_HELP_WIN (obligatory)

For more information, see *Input Help* on page 60.

- IS_SP_VALUE_CHECK
  - Implementation of the class role using the SP is optional.
  - Function: SP value check
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_VALUE_CHECK.
  - The class role includes the interface:
    - IF_SRM_SP_VALUE_CHECK_EXE (obligatory)

For more information, see *Value Check* on page 64.

- IS_SP_PROTOCOL_HANDLER
  - Implementation of the class role using the SP is optional.
  - Function: Logging activities
  - The SP class inherits directly or indirectly from the framework class CL_SRM_SP_PROTOCOL.
  - The class role includes the following interfaces (already implemented by CL_SRM_SP_PROTOCOL):
    - IF_SRM_SP_PROTOCOL_ENTRY
    - IF_SRM_SP_PROTOCOL_META
    - IF_SRM_SP_PROTOCOL_VIEWER

For more information, see *Logging* on page 67.

- IS_SP_PROP_REPOSITORY
  - Implementation of the class role using the SP is optional.
  - Function: Property unification: Operations in the attribute repository
  - The class role includes the interface:
    - IF_SRM_SP_PROP_REPOSITORY (obligatory)

For more information, see *Property Unification* on page 54.

- IS_SP_PROP_VIS_DEFINE
  - Implementation of the class role using the SP is optional.
  - Function: Property unification: Visualization description for the attributes
  - The class role includes the following interfaces:
    - IF_SRM_SP_PROP_VIS_DEFINE (obligatory)
    - IF_SRM_SP_PROP_VIS_LIST_DEF (optional)

For more information, see *Property Unification* on page 54.

- IS_SP_PROP_QUERY_DEFINE
  - Implementation of the class role using the SP is optional.
  - Function: Property unification: Search description for the attributes
  - The class role includes the interface:
    - IF_SRM_SP_PROP_QUERY_DEFINE (obligatory)

For more information, see *Property Unification* on page 54.

- IS_SP_PROP_VALUE
  - Implementation of the class role using the SP is optional.
  - Function: Property unification: Value handling for the attributes
  - The class role includes the interface:
    - IF_SRM_SP_PROP_VALUE (obligatory)

For more information, see *Property Unification* on page 54.

### 2.1.1 Registration of Service Provider Classes

When service provider classes have been completely implemented, they must be registered in the RM Framework Registry. To register classes , you use the *Registry Maintenance* transaction (transaction SRMREGEDIT). Service providers for Records Management must be registered within the area S_AREA_RMS.

To register a service provider, proceed as follows:

Position the cursor on the area S_AREA_RMS and choose *Create Service Provider* from the context menu. A dialog box is displayed. Enter an ID and a short description for the service provider. Choose the service provider type SRM_GENERAL. A dialog box is displayed with several tab pages: On the *Attributes* tab page, choose an icon for the elements (ICON: Instance) and an icon for the element types (ICON: Model) of the service provider. On the *Classes* tab page, enter the classes of the service provider and save your entries. The system fills the remaining tab pages automatically (for more information, see section 2.2).

## 2.2 Methods Called at Registration

The classes of a service provider must be registered in the RM Framework Registry. The following text describes the methods that are called at the time of registration.

### 2.2.1 IF_SRM_SP_SYSTEM_PARA

The interface IF_SRM_SP_SYSTEM_PARA groups together the methods for publishing parameters of the service provider.

#### IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_SP_POID

| | | | |
|---|---|---|---|
| RETURNING | re_desc | TYPE | srm_list_edit_attribute_desc |

This method returns a description of the SP POID parameters. The SP POID parameters supply the SP key information that is required for identification of the SP repository objects.

For each SP POID parameter, an attribute description object must be filled and added to the table of attribute description objects (returning parameter RE_DESC). If no SP POID parameters exist, the table remains empty.

**Notes for Programming**

To supply the table with data, one attribute description object must be requested and filled for each POID parameter in the implementation. An attribute description object is requested using a factory object. the method CREATE_ATTR_DESCR_SP_POID. The method returns an initial attribute description object for describing an SP POID parameter. For more information, see Procedures for Attribute Description Objects and Attribute Value Objects on page 51.

**Example Code with Commentary**

```
METHOD IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_SP_POID.

DATA: ** Factory Object
        lo_factory     TYPE REF TO if_srm_srm_object_factory,
      ** Attribute description object
        lo_ead         TYPE REF TO if_srm_edit_attribute_desc,
      ** Structure variable for general attribute description
        general_desc   TYPE srmadgen,
      ** Structure variable for data type-specific attribute description
        string_desc    TYPE srmadstr.

** Get factory object
  lo_factory = me->if_srm~get_srm_object_factory( ).
** Request an empty attribute description object from the factory object:
  lo_ead = lo_factory->create_attr_desc_sp_poid( ).

** Fill general attribute description
** (The values in the following example are values from the SP for URLs. The
values
** are different in every SP.)

** The ID of the POID parameter is stored in the constant srm_sp_poid_id_for_url
** from a type pool (you must create the type pool and
** include it in the class)
```

```
   general_desc-id    = srmur_sp_poid_id_for_guid. "URL
** The short text of the POID parameter is stored in text 001
   general_desc-text  = text-001. "GUID
** The POID parameter is of type string
   general_desc-type = IF_SRM_ATTRIBUTE_DESC=>STRING.
** It is not possible to assign multiple values to the POID parameter.
   general_desc-is_list = if_srm=>false.
** The POID parameter is obligatory
   general_desc-is_mand = if_srm=>true
** There is no input help for the POID parameter
   general_desc-is_help = IF_SRM=>false.
** There is no value check for the POID parameter
   general_desc_is-is_check = IF_SRM=>false.
** Transfer of the filled structure variables to the attribute description
** object:
   lo_ead->set_general_description( general_desc ).


** Fill data type-specific attribute description.
** (The value in the following example is a value from the SP for URLs. This
value is different in ** every service provider.)
** The string can be a maximum of 1000 characters long
   string_desc-max_length = 1000.
** Transfer of the filled structure variables to the attribute description
** object:
   lo_ead->set_general_description( string_desc ).


** Append the attribute description object to the table (returning-
** parameter)
   append lo_ead to re_desc.

ENDMETHOD.
```

---

**IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_CONNECTION**

| RETURNING | re_desc | TYPE | srm_list_edit_attribute_desc |
|---|---|---|---|

This method returns a description of the connection parameters. The SP connection parameters are used to define the element types.

For each connection parameter, an attribute description object must be filled and added to the table of att ribute description objects (returning parameter RE_DESC). If no connection parameters exist, the table remains empty.

**Notes for Programming**

To supply the table with data, one attribute description object must be requested and filled for each connection par ameter in the implementation. An attribute description object is requested using a factory object. The method CREATE_ATTR_DESCR_CONNECTION is used for this. The method returns an initial attribute description object for describing a connection parameter.

Programming is the same as for the publishing of POID parameters.

**IF_SRM_SP_SYSTEM_PARA~GET_ATTR_DESC_CONTEXT**

| RETURNING | re_desc | TYPE | srm_list_edit_attribute_desc |
|---|---|---|---|

This method returns a description of the context parameters. The context parameters can

optionally be used to exchange information between two SPs over the framework.

For each parameter, an attribute description object must be filled and added to the table of attribute description objects (returning parameter RE_DESC). If no context parameters exis t, the table remains empty.

**Notes for Programming**

To supply the table with data, one attribute description object must be requested and filled for each context parameter in the implementation. An attribute description object is requested using a factory ob ject. The method CREATE_ATTR_DESCR_CONTEXT is used for this. The method returns an initial attribute description object for describing a context parameter.

Programming is the same as for the publishing of POID parameters.

## 2.3 Methods Called at Runtime

The following text documents the methods that are called during the request at runtime.

A request starts when a user selects an activity for an element or an element type. The request ends when this activity has been executed. Example: In a record, a user selects a document to display. The request ends as soon as the document is displayed to the user. Two service providers are involved in this process. The SP for records (Records Browser) as the calling SP, and the SP for documents as the called SP.

The flow diagr am *Process Flow of a Request* (see page 18*)* is provided as an aid to understanding. In the flow diagram and in the following documentation, the calling service provider is called SP A, and the called service provider is called SP B. SP C is a (hypothetical) SP, which is covered by SP B in the display. The method calls to SP C are of no importance to the flow of request processing between SP A and SP B, they are only displayed for completeness.

The flow diagram displays both method s that are implemented by the framework, and methods that must be implemented by the SP developer. The methods that must be implemented by the SP developer are displayed in bold. Methods that begin with "MY" are internal methods of the relevant service provider. The name of this service provider is not specified.

An SP that does not start any requests itself, but instead is called by other SPs, only occurs in the role of SP B. To develop an SP of this sort, only the methods of the client object and back end object of SP B must be implemented (in a standard case).

Notes for reading the flow diagram:

Objects are represented by rectangles. The vertical dotted line is the lifetime of the object; the time runs from top to bottom. The dark -filled vertical bars rep resent periods of time during which an object is the focus of control for the whole sequence. Transparent vertical bars represent periods of time during which an object has delegated the process control. The arrows represent the method calls. The object to which the arrow is pointing has implemented the method.

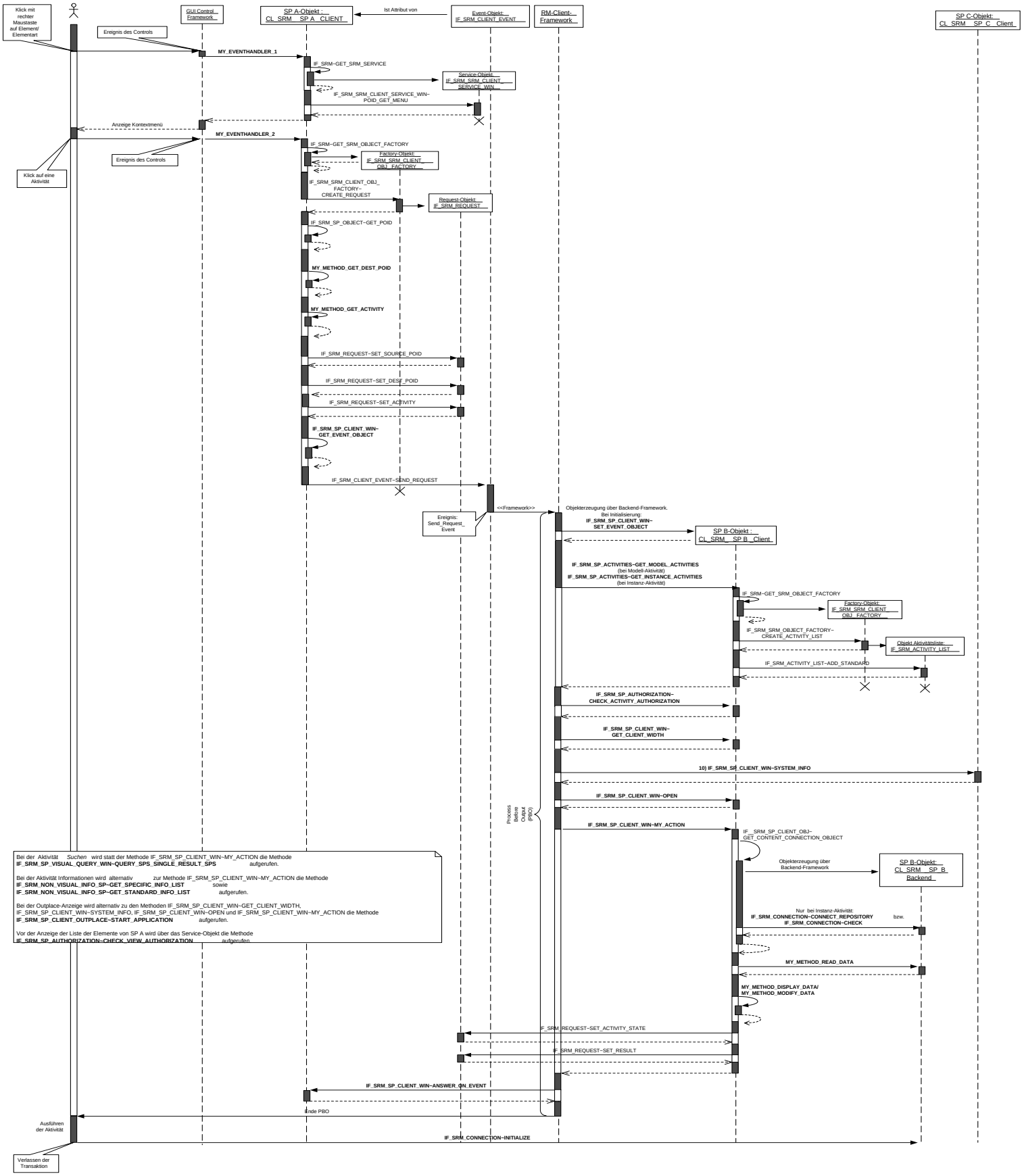



**Figure 1** Flow of a Request

### 2.3.1 IF_SRM_CONNECTION

This interface groups together methods that control the lifetime of the back-end object.

**IF_SRM_CONNECTION~CONNECT_REPOSITORY( )**

Parameters: none

This method is called when instance activities are executed. The method makes a connection to the SP repository object and can load data from the SP repository object into the runtime object. This is useful if you want to retrieve data only once (for example, the content of a document) and you want to buffer this data for performance reasons. The POID itself does not have to be saved, because it can be accessed at any time.

The SP repository object is identified using the POID object. The interface IF_SRM_POID is available for selecting the POID object attributes. The connection parameter values are selected using the interface IF_SRM_CONNECTION_ATTR.

The following steps must be executed:

1. Select SP POID parameter values : IF_SRM_POID~GET_SP_POID_VALUE_BY_ID for the object. If errors occur here, the exception CX_SRM_POID_SP_POID must be triggered.
2. Select connection parameter values: IF_SRM_CONNECTION_ATTR~-GET_STRING_VALUE for the object. If errors occur here, the exception CX_SRM_CONNEC_FAILED must be triggered.
3. Check whether the SP POID parameter values and connection parameter values identify an object in the SP repository on a one-to-one basis. If no object is found, the exception CX_SRM_CONNEC_FAILED must be triggered. If more than one object is found, either the connection parameters or the SP POID parameters are insufficient. The corresponding exception must be triggered.
4. If necessary, load information about the SP repository object into the runtime object (SP-specific).

**IF_SRM_CONNECTION~CHECK( )**

Parameters: none

This method is called when instance activities are executed, if these have already been executed once. It is called instead of the method IF_SRM_CONNECTION~CONNECT_REPOSITORY. This method checks whether the connection still exists between the runtime object and the SP repository object.

The following steps must be executed:

1. Check whether a connection has been made previously. To check this, the SP can either read the state object supplied by the framework, or check its own attributes for the object. If the result of this check is negative, the exception CX_SRM_CONNEC_STATE must be triggered.
2. Check whether the addressed SP repository object can be contacted as before. If the result of this check is negative, the exception CX_SRM_CONNEC_FAILED must be triggered.

**IF_SRM_CONNECTION~INITIALIZE( )**

Parameters: none

This method is called when the user exits Records Management. It initializes the SP back-end

object.

### 2.3.2 IF_SRM_NON_VISUAL_INFO_SP (optional)

This interface groups together methods that supply information about an element instance.

**IF_SRM_NON_VISUAL_INFO_SP~GET_STANDARD_INFO_LIST( )**

| | | | |
|---|---|---|---|
| EXPORTING | Ex_display_name | TYPE | String |
| EXPORTING | Ex_date_created | TYPE | Timestamp |
| EXPORTING | Ex_user_created | TYPE | String |
| EXPORTING | Ex_date_last_changed | TYPE | Timestamp |
| EXPORTING | Ex_user_last_changed | TYPE | String |
| EXPORTING | Ex_state_value | TYPE | String |
| EXPORTING | Ex_state_display_name | TYPE | String |
| EXPORTING | Ex_visual_info_sp_poid | TYPE | String |
| EXPORTING | Ex_semantic_type | TYPE | String |

This method is called as an alternative to the method IF_SRM_SP_CLIENT_WIN~ MY_ACTION, if the user selects the *Information* activity for an element type. The method returns a list of SP standard attribute values, which are displayed in a dialog bo x. The signature of the method shows which of the attributes are standard attributes.

The method is also called to determine the display name of an element. The parameter EX_DISPLAY_NAME should now contain an entry. All other entries are optional.

Note: No log can be generated in this method, since the method is called within the log call. If the method were to call the log again, an endless loop would come about.

**IF_SRM_NON_VISUAL_SP~GET_SPECIFIC_INFO_LIST( )**

| | | | |
|---|---|---|---|
| RETURNING | Re_value | TYPE | srm_list_attribute_value |

This method is also called as an alternative to the method IF_SRM_SP_CLIENT_WIN~ MY_ACTION, if the user selects the *Information* activity for an element type. The method returns a list of SP-specific attribute value objects. All entries are optional.

Note: No log can be generated in this method, since the method is called within the log call. If the method were to call the log again, an endless loop would come about.

### 2.3.3 IF_SRM_SP_INFO (optional)

This interface contains a method with which you can set the se mantic type for the elements of the service provider. The semantic type is an extension to the classification parameter TYPE, which can be set in Customizing. In contrast, the semantic type cannot be changed in Customizing and contains a keyword that characterizes the elements of the service provider.

If you have set the semantic type, another service provider can determine this using the service object method IF_SRM_SRM_SERVICE~GET_SP_SEMANTIC_TYPE.

**IF_SRM_SP_INFO~SEMANTIC_TYPE_GET( )**

| | | | |
|---|---|---|---|
| RETURNING | semantic_type | TYPE | String |

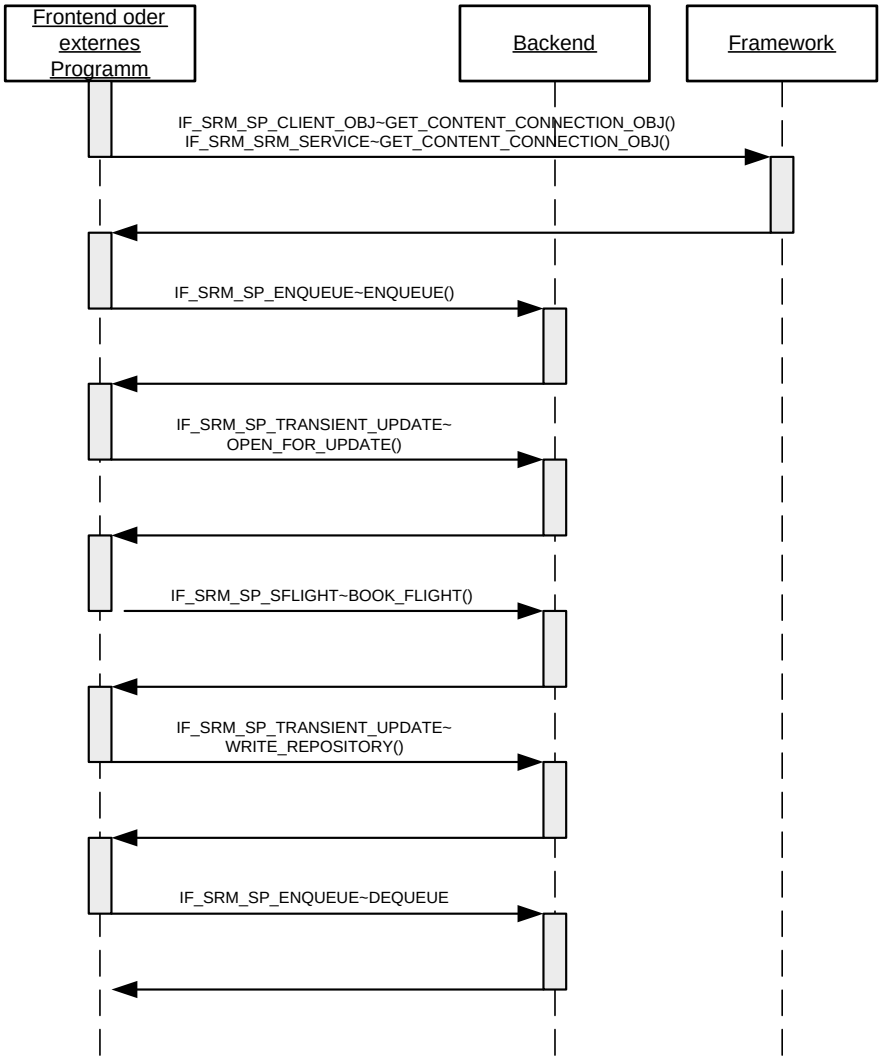
As a value for the returning parameter SEMANTIC_TYPE, you can use one of the constants C_SEMANTIC_* on the interface IF_SRM_NON_VISUAL_INFO_SP.

### 2.3.4 IF_SRM_SP_ENQUEUE (optional)

This interface, together with the interface IF_SRM_SP_TRANSIENT_UPD ATE (see below), are only relevant for service providers that manage their data in separate repository.

The interface can be used to lock and unlock elements of the service provider. For more details about the SAP locking concept, see the corresponding R/3 documentation.

Accessing the SP Backend must always be done roughly according to the following schema, irrespective of whether it is being done externally (a report, and so on) or through the SP client:



**IF_SRM_SP_ENQUEUE~ENQUEUE( )**

| | | | |
|---|---|---|---|
| IMPORTING | Im_mode | TYPE | String |
| IMPORTING | Im_scope | TYPE | String |

You call this method from the SP front end or from an external program, to lock an element.

Constants for lock type:

- IF_SRM_SP_ENQUEUE=>MODE_SHARED: Shared lock, read lock
- IF_SRM_SP_ENQUEUE=>MODE_EXCLUSIVE: Exclusive lock, write lock
- IF_SRM_SP_ENQUEUE=>MODE_EXTENDED: Extended lock; extended write lock

Constants for area of validity:

- IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG : Lock is automatically lifted when the transaction is ended
- IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG_AND_UPDATE_TASK: Lock belongs to dialog and update task
- IF_SRM_SP_ENQUEUE=>SCOPE_PERSISTENT: Persistent lock, can only be lifted by an explicit DEQUEUE
- IF_SRM_SP_ENQUEUE=>SCOPE_UPDATE_TASK: The update task inherits the lock (type 2)

Constant for implementation-dependent default value:

- IF_SRM_SP_ENQUEUE=>USE_IMPLEMENTATION_DEFAULT: Use implementation - dependent default value

Note that not all combinations of mode and scope have to be supported. If combinations that are not supported are required, an exception of type CX_SRM_SP_ENQUEUE must be raised (set text ID and attributes!)

The constant for implementation -dependent default values has a special role: it can be used instead of the normal constants for lock type and validity area; here, the implementation mu st use a default value. (Background: The sensible values are different for each lock object, meaning that no one default value can be specified. Nevertheless, a default value should be offered to the caller, to simplify the programming).

**Example Code**

```
DATA:  MY_BACKEND     TYPE REF TO IF_SRM_SP_BACKEND,
       SRM_SERVICE    TYPE REF TO IF_SRM_SRM_SERVICE,
       SRM            TYPE REF TO IF_SRM.

MY_BACKEND = … .
SRM_SERVICE = SRM->GET_SRM_SERVICE().
SRM_SERVICE->ENQUEUE_ELEMENT( IM_POID = POID
                              IM_MODE = MODE
                              IM_SCOPE = SCOPE ).
```

Note: The SRM service method checks for availability of the optional interface IF_SRM_SP_ENQUEUE. For this reason, using the direct call is preferable for this.

**IF_SRM_SP_ENQUEUE~DEQUEUE( )**

| | | | |
|---|---|---|---|
| IMPORTING | Im_mode | TYPE | string |
| IMPORTING | Im_scope | TYPE | string |

You call this method from the SP front end or from an exte rnal program, to unlock an object. For the meanings of the constants, see IF_SRM_SP_ENQUEUE~ENQUEUE( ) above.

### 2.3.5 IF_SRM_SP_TRANSIENT_UPDATE (optional)

This interface, together with the interface IF_SRM_SP_ENQUEUE (see above), are only relevant for service providers that manage their data in separate repository.

This interface is used to process operations on the back end, without the data being written into the repository immediately (known as transient processing).

**IF_SRM_SP_TRANSIENT_UPDATE~OPEN_FOR_UPDATE()**

Parameters: none

You call this method from the SP front end or from an external program, to load objects from the repository into the memory, or to create an empty object (depending on SP). After calling this method, you have the option of calling modifying methods for the specific back-end interface.

**IF_SRM_SP_TRANSIENT_UPDATE~WRITE_TO_REPOSITORY( )**

| | | | |
|---|---|---|---|
| IMPORTING | im_update_mode | TYPE | String |

You call this method from the SP front end or from an external program, to write the modified values that are in the me mory into the repository. After calling this method, the object is then in a saved state. The parameter IM_UPDATE_MODE can have the following values:

- `IF_SRM=>DB_UPDATE_AND_COMMIT`: COMMIT is executed immediately.
- `IF_SRM=>DB_UPDATE`: No COMMIT is executed (default setting). You have the execute the COMMIT yourself.
- `IF_SRM=>DB_UPDATE_TASK_AND_COMMIT`: The COMMIT is executed by the update task.
- `IF_SRM=>DB_UPDATE_TASK`: COMMIT is not executed by the update task.

Note: If changes are not saved, then open_for_u pdate() can be called again to return to the initial status.

**Example Code**

```
DATA:  OBJECT     TYPE REF TO IF_SRM_SP_BACKEND,
       TRANS_UPD  TYPE REF TO IF_SRM_SP_TRANSIENT_UPDATE.

OBJECT = ….

TRANS_UPD ?= OBJECT.
TRANS_UPD->OPEN_FOR_UPDATE( ).
OBJECT->DO_STUFF().
…
TRANS_UPD->WRITE_TO_REPOSITORY().
```

Note: Here, an indirect call with checking does not make sense, since the “internal info” of the class (which interface can you use to manipulate the data?) must be known.

## 2.3.6 IF_SRM_SP_ACTIVITIES

This interface groups together methods that publish the activities of the service provider. The model activities are displayed in th e context menu for an element type, the instance activities are displayed in the context menu for an element.

**IF_SRM_SP_ACTIVITIES~GET_MODEL_ACTIVITIES( )**

| RETURNING | re_activities | TYPE | if_srm_activity_list |
|---|---|---|---|

This method publishes the activities for element t ypes (model activities). The returning parameter RE_ACTIVITES should be filled with activities. The framework calls this method if the user has selected a model activity. The framework compares whether the current activity that has been passed to the request is contained in the list of activities for SP B.

The method is also called by the service object in the method IF_SRM_SRM_CLIENT_SERVICE_WIN~POID_GET_MENUE (see the flow diagram: SP A calls these service object methods in its method My_Eventhandler_1). If you have implemented an authorization check in the method IF_SRM_SP_AUTHORIZATION~CHECK_ACTIVITY_AUTHORIZATION (see below), it is useful to perform an authorization check. If the result of the authorization check is negative, the activity is not displayed.

**Example Code with Commentary**

```
METHOD if_srm_sp_activities~get_model_activities.

DATA:  o_factory  TYPE REF TO  if_srm_srm_object_factory.

** Generate the factory object:
   o_factory = me->if_srm~get_srm_object_factory( ).
** Request an activity list from the factory object:
   re_activities = o_factory->create_activity_list( ).
```

```
** Create Activity: Query the authorization and fill the activity list:
   IF me->if_srm_sp_authorization~check_activity_authorization(
     im_activity = if_srm_activity_list=>create ) = if_srm=>true.
   re_activities->add_standard( if_srm_activity_list=>create ).
   ENDIF.
** Find activity: Query authorization + fill activity list:
   IF me->if_srm_sp_authorization~check_activity_authorization(
     IM_ACTIVITY = IF_SRM_ACTIVITY_LIST=>QUERY ) = if_srm=>true.
   re_activities->add_standard( if_srm_activity_list=>query ).
   ENDIF.

ENDMETHOD.
```

---

**IF_SRM_SP_ACTIVITIES~GET_INSTANCE_ACTIVITIES( )**

| RETURNING | re_activities | TYPE | if_srm_activity_list |
|---|---|---|---|

This method publishes the activities for elements (instance activities). The returning parameter RE_ACTIVITES should be filled with the activities. The framework calls this method if the user selects an instance activity. The framework compares whether the current activity that has been passed to the request is contained in the list of activities for SP B.

The method is also called by the service object in the method IF_SRM_SRM_CLIENT_SERVICE_WIN~POID_GET_MENUE (see the flow diagram: SP A calls these service object methods in its method My_Eventhandler_1). If you have implemented an authorization check in the method IF_SRM_SP_AUTHORIZATION~CHECK_ACTIVITY_AUTHORIZATION (see below), it is useful to perform an authorization check. If the result of the authorization check is negative, the activity is not displayed.

**Example Code**

---

```
METHOD if_srm_sp_activities~get_instance_activites.

DATA:  o_factory  TYPE REF TO  if_srm_srm_object_factory.

O_factory = me->if_srm~get_srm_object_factory( ).
re_activities = o_factory->create_activity_list( ).

IF me->if_srm_sp_authorization~check_activity_authorization(
     im_activity = if_srm_activity_list=>display) = if_srm=>true.
  re_activities->add_standard( if_srm_activity_list=>display ).
  re_activities->set_default( if_srm_activity_list=>display ).
ENDIF.

IF me->if_srm_sp_authorization~check_activity_authorization(
     im_activity = if_srm_activity_list=>modify) = if_srm=>true.
  re_activities->add_standard( if_srm_activity_list=>modify ).
ENDIF.

IF me->if_srm_sp_authorization~check_activity_authorization(
     im_activity = if_srm_activity_list=>delete) = if_srm=>true.
   re_activities->add_standard( if_srm_activity_list=>delete ).
ENDIF.
```

```
IF me->if_srm_sp_authorization~check_activity_authorization(
     im_activity = if_srm_activity-list=>protocol) = if_srm=>true.
  re_activities->add_standard( if_srm_activity_list=>protocol ).
ENDIF.

ENDMETHOD.
```

## 2.3.7 IF_SRM_SP_AUTHORIZATION

This interface controls the authorization for executing the published activities, and the authorization for displaying elements as nodes in a list.

**IF_SRM_SP_AUTHORIZATION~CHECK_ACTIVITY_AUTHORIZATION( )**

| | | | |
|---|---|---|---|
| IMPORTING | im_activity | TYPE | srmif_sp_activity |
| RETURNING | re_authorized | TYPE | srmboolean |

This method checks the authorization for executing the selected activity. Before this method is called, the framework checks the a uthorization for executing the activity using the general Records Management authorization object (S_SRMSY_CL, with the fields *RMS*, *Element Type* and *Activity*). This method enables you to implement a more detailed authorization check. This is useful if you want more precise control of the separate authorizations within an element type (for example, access to attributes).

If the authorization check of the RM authorization object is sufficient, set the returning parameter to if_srm=>true. This means there are no authorization restrictions. If you set the value to *false*, the user does not have authorization to execute the current activity.

**Example Code**

```
METHOD if_srm_sp_authorization~check_activity_authorization.
    re_authorized = if_srm=>true.
ENDMETHOD.
```

**IF_SRM_SP_AUTHORIZATION~CHECK_VIEW_AUTHORIZATION( )**

| | | | |
|---|---|---|---|
| RETURNING | Re_authorized | TYPE | Srmboolean |

This method is called by th e service object before SP A displays its elements as nodes in a list. The method checks whether the user is permitted to view each node. In addition, SP A calls the method IF_SRM_SRM_CLIENT_SERVICE~
AUTH_CHECK_VIEW_BY_POID for the service object, and ente rs the POID of one element each time.

The service object method first checks the authorization for list display using the general Records Management authorization object (S_SRMSY_CL, with the fields RMS, element type, and activity). Secondly, the method ch ecks the authorization for list display using the method IF_SRM_SP_AUTHORIZATION~CHECK_VIEW_AUTHORIZATION. This method enables you to implement a very detailed authorization check, which is more precise than the authorization check using the RM authorization object.

If the authorization check of the general RM authorization object is sufficient, set the returning parameter to if_srm=>true. This means there are no authorization restrictions. If you set the value to *false*, the user is not authorized to display the element identified by the POID in a list.

**Example Code**

```
METHOD if_srm_sp_authorization~check_view_authorization.
    re_authorized = if_srm=>true.
ENDMETHOD.
```

## 2.3.8 IF_SRM_SP_CLIENT_WIN

This interface groups together methods for executing an activity.

**IF_SRM_SP_CLIENT_WIN~GET_EVENT_OBJECT( )**

| | | | |
|---|---|---|---|
| RETURNING | Event_object | TYPE | if_srm_client_event |

This method reads the event objec t at SP A. The event object is an attribute of the client class of each SP. The event object is a kind of “post box” to which the request is transferred.

The coding of this method is as follows:

```
METHOD if_srm_sp_client_win~get_event_object.
   event_object = me->if_srm_sp_client_win~event_object.
ENDMETHOD.
```

**IF_SRM_SP_CLIENT_WIN~SET_EVENT_OBJECT( )**

| | | | |
|---|---|---|---|
| IMPORTING | im_event_object | TYPE | if_srm_client_event |

This method sets the reference to the event object. The call is made when SP B is initialized. The event object is set for each SP, although it is only used if SP occurs in the role of SP A.

The coding of this method is as follows:

```
METHOD if_srm_sp_client_win~set_event_object.
   me->if_srm_sp_client_win~event_object = im_event_object
ENDMETHOD.
```

**IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH( )**

| | | | |
|---|---|---|---|
| IMPORTING | im_request | TYPE | if_srm_request |
| RETURNING | re_client_width | TYPE | I |

SP B uses this method to specify what percentage of the available width of the screen window it will take up. The method returns the desired width for an activity.

0 % = no display

100 % = normal display

**Example Code:**

```
METHOD if_srm_sp_client_win~get_client_width.

CASE im_request->get_activity( ).
  WHEN if_srm_activity_list=>display.
    re_client_width = 50.
  WHEN if_srm_activity_list=>delete.
    re_client_width = 0.
  ...
 ENDCASE.

ENDMETHOD.
```

**IF_SRM_SP_CLIENT_WIN~SYSTEM_INFO( )**

| IMPORTING | system_info | TYPE | srmif_sp_message |
|---|---|---|---|

By calling this method, the framework informs the SP that executed an activity in the previous request (SP C) of any existing changes of status in its own visualization. The client framework system messages are processed in this method.

The import parameter SYSTEM_INFO can have the following values:

SRMIF_SYSTEM_INFO_HIDE

In the next step, another service provider is displayed over the service provider; it must end all asynchronous activities immediately and ensure tha t all data is saved. Responses to asynchronous activities must be sent.

If a user displays an element from SP B, without saving the element from SP C that they previously edited, SP C opens a dialog box. For example, “Data has not been saved. Save, Do Not Save, or Cancel?” If the user wants to cancel, the exception CX_SRM_SP_USER_CANCEL must be triggered.

SRMIF_SYSTEM_INFO_SHOW

The service provider is displayed again in the next step.
IF_SRM_SP_CLIENT_WIN~MY_ACTION is then called (see below).

SRMIF_SYSTEM_INFO_DESTROY

Records Management is ended, all asynchronous methods must be ended and the state must be saved. Responses to asynchronous activities must be sent.

**Example Code with Commentary**

```
METHOD if_srm_sp_client_win~system_info

CASE system_info.
   WHEN srmif_system_info_show.
** ->Refresh screen again, if necessary.
   WHEN srmif_system_info_hide.
** ->Save if SP currently in unsaved state.
** If the user wants to cancel:
     IF user_cancel = if_srm=>true.
        raise exception type cx_srm_spcl_user_cancel.
```

```
    ENDIF.
WHEN srmif_system_info_destroy.
** -> Exit Records Management: Save.
ENDCASE.

ENDMETHOD.
```

---

**IF_SRM_SP_CLIENT_WIN~OPEN( )**

| IMPORTING | im_parent | TYPE | cl_gui_container |
|---|---|---|---|
| RETURNING | re_main_control | TYPE | cl_gui_container |

The method is called when an activity for an element (POID) is called in addition to MY_ACTION. This method initializes the runtime object and generates the contro ls that are required for the visualization.

**Example Code with Commentary**

---

```
METHOD if_srm_sp_client_win~open.

** → Coding for initialization
** → Generate and return main container

CREATE OBJECT my_main_splitter.
re_main_control = my_main_splitter.
** → Generate own controls
 ...
ENDMETHOD.
```

---

**IF_SRM_SP_CLIENT_WIN~MY_ACTION( )**

| IMPORTING | im_request | TYPE | if_srm_request |
|---|---|---|---|

By calling this method, the framework informs SP B that SP A is requesting an activity from it. The request determines which activity is to be executed and for which element.

This method is called for the execution of all published activities except for the activity *Find*. (There is a separate interface method for implementing the activity *Find*: IF_SRM_SP_VISUAL_QUERY_WIN~QUERY_SPS_SINGLE_RESULT). For executing model activities (for example *Create*), the SP POID must be set in the POID object, to enable conversion from model POID to instance POID. For instance activities, this step is omitted. The new POID and the status of the activity must be set in the request object.

Notes for asynchronous requests:

An asynchronous request is when the SP cannot return to a secure status immediately after executing the activity. Example: When a document is created, the empty document is displayed and edited by the user. As soon as the document is saved, the asynchronous activity is ended. Directly after the document is create d, the SP responsible for the document responds with IF_SRM_REQUEST=>ACTIVITY_ONGOING. After the document is saved, the asynchronous response is sent using IF_SRM_CLIENT_EVENT~SEND_ASYNC_ANSWER.

Note that the asynchronous response must not be “forgotten”, because otherwise the service providers further up in the hierarchy will not be able to return to a secure state (for example, record). This can lead to loss of data in the superordinate SP. Where overlapping occurs, the asynchronous response also needs to be sent. See IF_SRM_SP_CLIENT~SYSTEM_INFO.

**Example Code with Commentary**

```
METHOD if_srm_sp_client_win~my_action.

DATA: lo_my_backend TYPE REF TO cl_srm_sp_b_backend,
      lo_my_poid    TYPE REF TO if_srm_poid,
      lt_poid       TYPE srm_list_poid,
      my_data       TYPE    … .

** SP B calls its back end:
lo_my_backend ?= me->if_srm_sp_client_obj~get_content_connection_object( ).

** SP B  executes the request:
CASE im_request->get_activity( ).

WHEN if_srm_activity_list=>create.
** Request the own POID object from the request object (model POID)
   lo_my_poid = me->if_srm_sp_object~get_poid( ).
** Specify own SP POID
   lt_poid = ... .
** Set the SP POID
   lo_my_poid->set_sp_poid( lt_poid ).
** Set result in request object:
   im_request->set_activity_state( if_srm_request=>activity_ongoing ).
   im_request->set_result( lo_my_poid ).

WHEN if_srm_activity_list=>display.
** Request own POID object from the request object (instance POID)
   lo_my_poid = me->if_srm_sp_object~get_poid( ).
** Read data from back end
   my_data = lo_my_backend->my_method-read_data( ).
** Display data
   me->my_method_display-data( my_data ).
** Set result in request object:
   im_request->set_activity_state( if_srm_request=>activity_finished_with_ok ).
   im_request->set_result( lo_my_poid ).

WHEN if_srm-activity_list=>edit.
   lo_my_poid = me->if_srm_sp_object~get_poid( ).
   my_data = lo_my_backend->my_method_read_data( ).
   me->my_method_modify_data ( my_data ).
          ...
   im_request->set_activity_state( if_srm_request=>activity_ongoing ).
   im_request->set_result( lo_my_poid ).

ENDCASE.

…

ENDMETHOD.
```

**IF_SRM_SP_CLIENT_WIN~ANSWER_ON_EVENT( )**

| IMPORTING | im_request | TYPE | if_srm_request |
|---|---|---|---|

By calling this method, the framework SP A transfers the request object of a request that it has initiated (the request can be synchronous or asynchronous). The method must therefore only be

implemented by SPs that also act as SP A. SPs that act as SP B on ly, create the method as the method body.

The request can be identified using the request ID assigned by the framework when the request was generated. The activity status and the current POID can also be read using the request object.

**Example Code**

```
METHOD if_srm_sp_client_win~anwer_on_event.

DATA: l_request_id      TYPE         srmreqid,
      l_activity_state  TYPE         string,
      lo_result_poid    TYPE REF TO  if_srm_poid.

l_request_id = im_request->get_request_id( ).
l_activity_state = im_request->get_activity_state( ).
lo_result_poid = im_request->get_result( ).

** The data is processed here

ENDMETHOD.
```

## 2.3.9 IF_SRM_SP_CLIENT_OUTPLACE

This interface is only obligatory for service providers that always call their activities in a new session (outplace). The method includes an interface for executing the activities.

**IF_SRM_SP_CLIENT_OUTPLACE~START_APPLICATION( )**

| IMPORTING | Im_request | TYPE | if_srm_request |
|---|---|---|---|

This method is called instead of the methods IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH, IF_SRM_SP_CLIENT_WIN~SYSTEM_INFO, IF_SRM_SP_CLIENT_WIN~OPEN and IF_SRM_SP_CLIENT_WIN~MY_ACTION, if SP B displays its elements outplace. The method is only for implementation by service providers that use outplace display. The implementation is the same as for the method IF_SRM_SP_CLIENT_WIN~MY_ACTION.

Example: The service provider for transactions calls its element, a transaction. Because no element is integrated into the framework, there is no inplace display. Only the activity “Display in new session” (DISPLAY_OUT) is offered in the context menu. It is not possible to execute activities on other service providers. Only synchronous activities are permitted.

Note: Service providers that implement this method still have to implement the interface IF_SRM_SP_CLIENT_WIN. The methods of the interface IF_SRM_SP_CLIENT_WIN that are replaced by the method IF_SRM_SP_CLIENT_OUTPLACE~START_APPLICATION (see above), must always be created as the body of the methods.

## 2.3.10 IF_SRM_SP_VISUAL_QUERY_WIN

This interface includes the method for executing the activity *Find*.

**IF_SRM_SP_VISUAL_QUERY_WIN ~QUERY_SPS_SINGLE_RESULT SPS( )**

| | | | |
|---|---|---|---|
| IMPORTING | Request | TYPE | if_srm_request |

This method is used instead of the method IF_SRM_SP_CLIENT_WIN~MY_ACTION or IF_SRM_SP_CLIENT_OUTPLACE~START_APPLICATION, if the user has selected the activity *Find* for an element type. The search is aimed at exactly one element type.

A search dialog normall y consists of an input template, in which the search parameters can be entered, and a list, from which the user selects one of the elements found. Exactly one (SINGLE RESULT) SP object must be identified.

In the same way as for the activity *Create*, the SP POID must be set in the POID object to enable conversion from model POID to instance POID. The new POID object and the status of the activity must then be set in the request object (see the example code for the method IF_SRM_SP_CLIENT_WIN~MY_ACTION).

## 2.4 Calling a Service Provider in Passive Mode

If the RM base control or RM stacked control is embedded in a business context, it can be useful not to save changes to the RM element saved in the control individually, but instead at the same time as other data. To enable this, the SP must support the passive mode.

What does passive mode mean? In asynchronous activities that lead to a non -saved SP state (for example, IF_SRM_ACTIVITY_LIST=>MODIFY), the transition to the saved state is not permitted by the SP itself, but instead by the container in which the SP is running. The actual saving still occurs within the SP, it is only triggered from externally. In this case, the SP does not have its own Save button; instead the data is saved by using another Save button, for example, in the application toolbar on the screen.

SPs can support either one of the two visualization modes (active or passive), or they can support both at the same time. Passive calls are only possible if the SP element is called in the RM base control or RM stacked control.

The passive mode is implemented in the same way as the implementation of an SP in active mode, although instead of the interface IF_SRM_SP_CLIENT_WIN, the interface IF_SRM_SP_FRONTEND_SAPGUI_PASV (see page 1) must be implemented for the front -end class. (If both the passive and the active modes are supported, the interface IF_SRM_SP_FRONTEND_SAPGUI_PASV must be implemented in addition to the interface IF_SRM_SP_CLIENT_WIN.) Otherwise, the pushbuttons belo nging to those activities that are to be executed passively must be removed from the display in passive mode (for example, the Save button).

The front -end class must then be registered as normal using registry maintenance (transaction SRMREGEDIT). By imple menting IF_SRM_SP_FRONTEND_SAPGU, IF_SRM_SP_AUTHORIZATION, and IF_SRM_SP_ACTIVITIES, the service provider fulfills the class role IS_SP_FRONTEND_SAPGUI_PASV.

To change to passive mode, call the SP in the new class role IS_SP_FRONTEND_SAPGUI_PASV. The activ ities are actually executed in CL_SRM_REQUEST_PROCESSOR. The methods named in the following within the client framework and the client framework API have each been enhanced by the addition of an optional parameter [IM_]MODE_PASV, with the default IF_SRM=>F ALSE. If the parameter is set, the corresponding action is executed with a passive class role.

- IF_SRM_SRM_CLIENT_SERVICE~AUTH_CHECK_ACTIVITY( )
- IF_SRM_SRM_CLIENT_SERVICE~AUTH_CHECK_VIEW_BY_POID( )
- IF_SRM_SRM_CLIENT_SERVICE~POID_CHECK_ACTIVITY( )

- IF_SRM_SRM_CLIENT_SERVICE~POID_GET_ACTIVITIES( )
- IF_SRM_SRM_CLIENT_SERVICE~POID_GET_STANDARD_ACTIVITY( )

### 2.4.1 IF_SRM_SP_FRONTEND_SAPGUI_PASV

This interface can be implemented in addition to or instead of the interface IF_SRM_SP_CLIENT_WIN. The following methods are to be implemented on the interface IF_SRM_SP_CLIENT_WIN in the same way as the methods with the same names (see section 27).

- IF_SRM_SP_FRONTEND_SAPGUI~GET_EVENT_OBJECT( )
- IF_SRM_SP_FRONTEND_SAPGUI ~SET_EVENT_OBJECT( )
- IF_SRM_SP_FRONTEND_SAPGUI ~SYSTEM_INFO( )
- IF_SRM_SP_FRONTEND_SAPGUI~OPEN( )
- IF_SRM_SP_FRONTEND_SAPGUI ~MY_ACTION( )
- IF_SRM_SP_CLIENT_WIN~ANSWER_ON_EVENT( )

The following methods must be implemented specifically on the interface IF_SRM_SP_FRONTEND_SAPGUI :

**IF_SRM_SP_FRONTEND_SAPGUI~GET_REQUEST_VISIBLE( )**

| | | | |
|---|---|---|---|
| IMPORTING | IM_REQUEST | TYPE | IF_SRM_REQUEST |
| RETURNING | RE_REQUEST_VISIBLE | TYPE | SRMBOOLEAN |

In response to the request, this method returns information on whether the activity is to executed visually or non-visually (for example, Display -> visual, Delete -> non-visual)

The method is implemented in the same way as the method IF_SRM_SP_CLIENT_WIN~GET_CLIENT_WIDTH, except that you cannot assign the returning parameter RE_REQUEST_VISIBLE a number, but only TRUE or FALSE.

**IF_SRM_SP_FRONTEND_SAPGUI~FINISH_ASYNC( )**

| | | | |
|---|---|---|---|
| IMPORTING | IM_KEEP_STATE | TYPE | SRMBOOLEAN |

This method is executed for saving data. It is called within the method IF_SRM_BASE_CONTROL~FINISH_ASYNC (see above). It is called at the point of transition between an unsaved state and a saved state.

The import parameter IM_KEEP_STATE determines which mode the SP is to have after the data is saved (see the method IF_SRM_BASE_CONTROL~FINISH_ASYNC):

- IM_KEEP_STATE = IF_SRM=>TRUE: The SP remains in change mode
- IM_KEEP_STATE = IF_SRM=>FALSE: The SP changes to display mode

Note: When data is saved, you must take into account the update procedure specified in the request. To determine the update procedure for the request object, use IF_SRM_REQUEST~>GET_UPDATE_MODE. You receive one of the following constants as a returning parameter:

- IF_SRM=>DB_UPDATE: Changes are written; the SP cannot perform a commit.
- IF_SRM=>DB_UPDATE_AND_COMMIT: Changes are written; the SP then performs a COMMIT WORK.

- IF_SRM=>DB_UPDATE_TASK: Changes are written in the update task.
- IF_SRM=>DB_UPDATE_TASK_AND_COMMIT: Changes are written in the update task, and COMMIT_WORK is performed immediately afterwards.

If the SP does not control a particular update mode, the exception CX_SRM_UPDATE_MODE must be raised. For practical reasons, each SP should at least control the mode IF_SRM=>DB_UPDATE.

Important: The transition to display should not occur automatically when requests are sent. Otherwise the user of the control is burdened with additional work (the user would then have to block the messages). Instead, an internal call of the method IF_SRM_SP_FRONTEND_SAPGUI~MY_ACTION can be made.

# 3 SP Development: Methods Inherited for the Framework

The framework provides services. You can call these services using the inherited interface methods (see the diagram “Inheritance Hierarchy of the Service Provider Cla sses” on page 11). The following text documents the interfaces and their methods.

## 3.1 ME->IF_SRM

You can use the methods of this interface to request additional objects that offer additional services. For more information, see Applicable Framework Objects (see section 36).

## 3.2 ME->IF_SRM_POID

This interface groups together methods for access to its own POID object. For more information, see IF_SRM_POID: POID Object (see page 36).

## 3.3 ME->IF_SRM_CONNECTION_ATTR

This interface groups together methods for access to its own connection parameter values. It has the following methods:

- GET_VALUE: Returns a connection parameter value as an attribute value object (for more information, see page 51)
- GET_VALUES: Returns all connection parameter values (list of attribute value objects)
- GET_STRING_VALUE: Returns a connection parameter value for an attribute ID (the value is a list of strings, because the connection parameter can have multiple values assigned)

**Example Code: Reading Connection Parameter Values**
SP for infotypes, connection parameter RFC destination (variable “dest”)

```
DATA: lt_connection_par     TYPE srm_list_string,
      lwa_connection_par    TYPE srmliststr,
      dest                  TYPE srmavstr.
```

```
lt_connection_par = me->if_srm_connection_attr~get_string_value( 'RFC-
DESTINATION' ).

LOOP AT lt_connection_par INTO lwa_connection_par.
  dest = lwa_connection_par-value.
ENDLOOP.
```

## 3.4 ME->IF_SRM_CONTEXT_ATTR

This interface groups together methods for access to its own context parameter values. It has the methods named below.

Note: Depending on the definition of a context parameter, context parameters can have the data type String, Integer or Interface. To avoi d redundancy, XYZ is used in the method names below to represent STRING, INTEGER or INTERFACE. The names call the appropriate method according to the type of the context parameter.

- GET_XYZ_VALUE: Returns one or more values of a context parameter for an attribute ID

  **Example Code: Reading the Context Parameters**
  SP for infotypes, context parameter INFOTYPE (can have multiple values assigned, variable ex_infotype_tab)

```
DATA: ret_list_string        TYPE srm_list_string,
      ret_string             TYPE srmliststr,
      ex_infotype_tab        TYPE srmhr_itst_tab,
      wa_ex_infotype_tab     TYPE srmhritst.

   ret_list_string = me->if_srm_context_attr~get_string_value( 'INFOTYPE'
   ).
    LOOP AT ret_list_string INTO ret_string.
      CLEAR wa_ex_infotype_tab.
      MOVE ret_string-value TO wa_ex_infotype_tab-infotype.
      APPEND wa_ex_infotype_tab TO ex_infotype_tab.
    ENDLOOP.
```

- SET_XYZ_VALUE: Sets the value of its own context parameter
- GET_VALUES: Returns a list of all context parameters (list of attribute value objects, see page 44)
- EXECUTE_AUTOMATION_SET: Executes context automation. (context parameters of type Interface can only be used together with context automation).

## 3.5 ME->IF_SRM_SP_OBJECT

The interface IF_SRM_SP_OBJECT only has a single method: GET_POID. You use this method to read your own POID object. You receive an interface reference to IF_SRM_POID.

Note: The meth ods of the interface IF_SRM_POID (see page 26) are also available in your SP through the inherited class CL_SRM_SP_OBJECT. Caution: If you want to pass a POID object to an interface, you must obtain the POID object via IF_SRM_SP_OBJECT~GET_POID. Otherwise errors may occur.

## 3.6 ME->IF_SRM_SP_CLIENT_OBJ (only for SP front end)

This interface groups together services that you can call from your SP front end. It has the following methods:

- GET_VISUALIZATION_STATE_OBJECT: Returns a reference to an interface that pr ovides help with administering the state diagram
- GET_CONTENT_CONNECTION_OBECT: Returns a reference to its own back -end object. (It is not possible to obtain a reference from the back -end object to the client object, because the framework always calls an SP at the front end.)

  **Example Code: Get reference to own back-end object**

```
DATA: my_backend TYPE REF TO if_my_backend.
** The interface if_my_backend is an example that is defined by the SP.
my_backend ?= me->if_srm_sp_client_obj~get_content_connection_object( ).
```

  Note: It is recommended to group together all back-end methods using interfaces, so that you can exchange the back-end class if necessary.

## 3.7 ME->IF_SRM_CONNECTION_STATE (only for SP back end)

This interface groups together methods to read the state value of your back end -object. It has the following methods:

- GET_STATE: Returns the current state value
- GET_ALLOWED_SUCCESSORS: Returns permitted successor state
- SET_STATE: Sets a new state value. Possible state values are the following attributes of the interface IF_SRM_CONNECTION_STATE:
  - STATE_INITIAL
  - STATE_CONNECTED
  - STATE_NEW

# 4 Applicable Framework Objects

The objects described in the following are service objects for the service providers. The classes of the objects are already implemented using the framework. The objects are never instanced by an SP. Instances are always requested using IF_SRM, or by objects supplied by IF_SRM.

## 4.1 IF_SRM_POID: POID Object

This interface groups together methods for access to a POID object. To obtain your own POID object, use ME->IF_SRM_SP_OBJECT~GET_POID. To obtain foreign POID objects, you can use, for example, IF_SRM_SRM_SERVICE~GET_MODEL_POID or ~GET_INSTANCE_POID (see Service Object Page 40).

The interface IF_SRM_POID has the following methods:

- SET_SP_POID: Sets the SP POID. An SP must actively set the SP POID when a model

POID is changed to an instance POID, that is, in the methods that implement the *Create* and *Find* functions.

- `GET_SP_POID`: Returns the SP POID parameters as a list
- `GET_SP_POID_VALUE_BY_ID`: Returns a single SP POID parameter. If an element is identified by more than one SP POID parameter, you must call these methods once for every POID parameter.

**Example Code: Reading the SP POID Parameters**
Example: SP for URLs, POID parameter GUID (variable "poid_value")

```
DATA: poid_value TYPE string.
poid_value = me->if_srm_poid~get_sp_poid_value_by_id( 'GUID' ).
```

- `GET_AREA_POID`: Returns the AREA-POID as a list (in a Records Management context: the RMS ID)
- `GET_AREA_POID_VALUE_BY_ID`: Returns the current RMS ID. For the import parameter IM_ID, set the constant SRMIF_RMSID_ID from the type pool SRMIF. The value of this constant is `'RMS_ID'`.

**Example Code: Reading the Current RMS**

```
DATA: rms_id TYPE string.
TYPE-POOLS: srmif.
…
rms_id = me->if_srm_poid~get_area_poid_value_by_id( SRMIF_RMSOD_ID ).
```

- `GET_SRM_POID`: Framework internal use
- `GET_AREA_ID`: Returns the ID of the AREA that belongs to the POID
- `GET_SP_ID`: Returns the ID of the SP that belongs to the POID
- `GET_SPS_ID`: Returns the ID of the SPS that belongs to the POID
- `GET_POID_STATE`: Returns the current state of the POID. Possible state values are the following attributes (constants) of the interface IF_SRM_POID:
  - `STATE_INITIAL`: Framework internal use
  - `STATE_MODEL`: Set by the framework if the POID is a model POID
  - `STATE_INSTANCE`: Set by the framework if the POID is an instance POID
- `GET_POID_DIRECTORY_ID`: Returns the POID Directory ID of the POID (see page. 1).

The three methods listed below are shortcuts for reading the information about the POID object that is registered in the registry. The more detailed route leads via the Registry Object (see p. 1)

- `GET_SPS_REGISTRY:` Returns the interface reference to `IF_SRM_SPS_REGISTRY`, which you can use to read information on the relevant SPS.
- `GET_SP_REGISTRY:` Returns the interface reference to `IF_SRM_SP_REGISTRY`, which you can use to read information on the relevant service providers.
- `GET_AREA_REGISTRY:` Returns the interface reference to `IF_SRM_AREA_REGISTRY`, which you can use to read information on the relevant AREA.

## 4.2 IF_SRM_SRM_OBJECT_FACTORY: Factory Object

You can use the factory object to generate interface references to additional objects. To obtain the factory object, use `ME->IF_SRM~GET_SRM_SRM_OBJECT_FACTORY`. You use this method to obtain the interface reference to `IF_SRM_SRM_OBJECT_FACTORY`. At runtime, the system returns an object of the class `CL_SRM_SRM_CLIENT_OBJ_FACTORY`. You can therefore also reach the interface `IF_SRM_SRM_CLIENT_OBJ_FACTORY` using a cast.

The graphic below shows the inheritance hierarchy of the factory object:

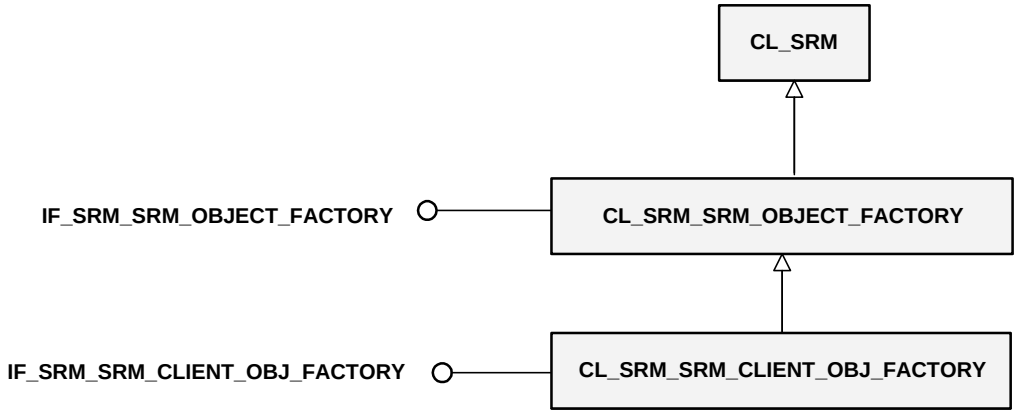



**Figure 4** Inheritance Hierarchy of the Factory Object

Note: You cannot program using class names. You should only program using interface references, because all classes are supposed to be exchangeable.

The graphic below provides an overview of the call hierarchy of the most important interface references. The arrows are to be read as follows: By calling the method in parentheses, you get the interface references to the object to which the arrow points.

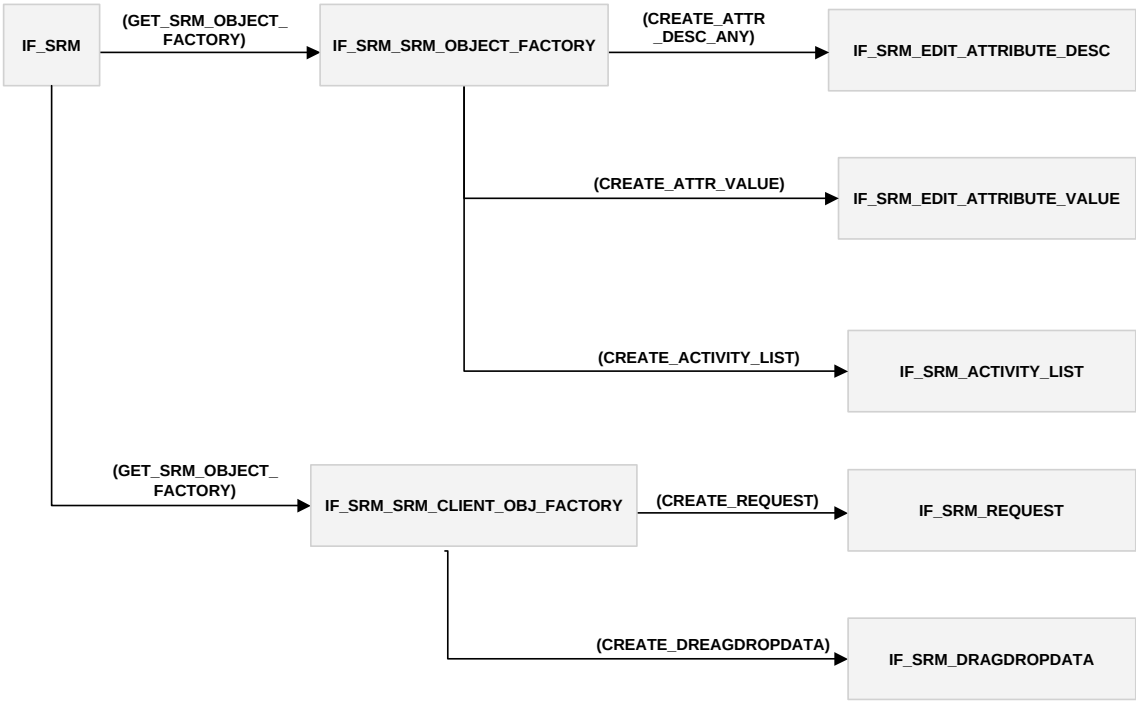



**Figure 5** Access to Interface References Using the Factory Object

### 4.2.1 IF_SRM_SRM_OBJECT_FACTORY

You can use this interface to generate attribute description objects and attri bute value objects, and an empty activity list. The methods are listed below.

- `CREATE_ATTR_DESC_XYZ:` Returns an interface reference to `IF_SRM_EDIT_ATTRIBUTE_DESC` (empty attribute description object). There are separate attribute description objects for SP POID parameters, for connection parameters, for context parameters, and for parameters that you want to display using the *Information* activity. There is also an attribute description object for each type of parameter. “XYZ” stands for SP POID, CONNECTION, CONTEXT, INFO and ANY. According to which parameter you are publishing, call the appropriate method (see page 12). For more information, see Procedure for Attribute Description Objects and Attribute Value Objects (see page 51).
- `CREATE_ATTRIBUTE_VALUE:` Returns an interface reference to `IF_SRM_EDIT_ATTRIBUTE_VALUE` (empty attribute value object). For more information, see Procedure for Attribute Description Objects and Attribute Value Objects (page 51).
- `CREATE_ACTIVITY_LIST:` Returns an interface reference to IF_SRM_ACTIVITY_LIST (empty activity list). You need an empty activity list if you want to offer a context menu with multiple columns, (a submenu). For more information, see page 25).

### 4.2.2 IF_SRM_SRM_CLIENT_OBJ_FACTORY

You use this interface to generate a request object and an object for Drag&Drop functions. The

interface has the following methods:

- `CREATE_REQUEST`: Returns an interface reference to `IF_SRM_REQUEST` (request object). For more information, see *Request Object* below.
- `CREATE DRAGDROPDATA`: Returns an interface reference to IF_SRM_DRAGDROPDATA.

## 4.3 IF_SRM_REQUEST: Request Object

At runtime, the request object contains all the information of a requests. The framework delivers the request object from the calling Service Provider to the called Service Provider, and then back again after the activity has been executed.

The calling Service Provider (referred to as SP A in the following) gets the request object through IF_SRM_SRM_CLIENT_OBJ_FACTORY~CREATE_REQUEST( ). It sets all the information for the request in the request object: the individual POID (SOURCE_POID), the POID of the element that is to execute the activity (DEST_POID), and the activity to be executed (ACTIVITY). This information can also include any parameters you require (PARAMETER). It then sends the request to the framework using IF_SRM_CLIENT_EVENT~SEND_REQUEST( ).

The called Service Provider (referred to as SP B in the following) receives the request object from the framework in the methods IF_SRM_SP_CLIENT_WIN~MY_ACTION( ) (or alternatively IF_SRM_SP_CLIENT_OUTPLACE~START_APPLICATION) and IF_SRM_SP_VISUAL_QUERY_WIN~QUERY_SPS_SINGLE_RESULT( ). It finishes reading the information and performs the activity correspondingly. It then sets the activity status (ACTIVITY_STATE) and the result POID (RESULT) for the request object.

SP A gets the request object returned from the framework in method IF_SRM_SP_CLIENT_WIN~ANSWER_ON_EVENT( ). It identifies the request using the request ID and reads the activity status and result POID.

Interface IF_SRM_REQUEST has the following attributes, which can be accessed using SET and GET methods.

REQUEST_ID: Number for the request. Is assigned by the framework when generating the request object and can be used to for identifying the request.

SOURCE_POID: POID of the element of SP A. Must be set by SP A before the activity is executed. It is read by the framework to be able to send the request back to SP A.

DEST_POID: POID of the element of SP B. Must be set by SP A before the activity is executed. It is read by the framework to be able to send the request to SP B.

ACTIVITY: Activity that is to be executed. Must be set by SP A before the activity is executed. It is read by SP B, to determine the activity that is to be executed.

PARAMETER: Any required parameter. Can be set by SP A before the activity is executed (optional). The ID of the parameter is set through IM_ID, and the value is set through IM_VALUE, in the forum of an attribute value object. Setting a parameter only makes sense if SP B also knows the parameter. It is read by SP B.

ACTIVITY_STATE: Activity status. Must be set by SP B after the activity is executed. Is read by SP A when it gets the request object again.

RESULT: Result POID. Must be set by SP B after the activity is executed. Is read by SP A when it gets the request object again.

UPDATE_MODE: Update mode, only relevant in passive mode (for more information about the update mode, see page 49). Must be set by SP A before the activity is executed. It is read by SP B. If SP B cannot execute the corresponding update mode, it must generate an error

message.

ERROR_MESSAGE: Error message, relevant in the BSP environment only. Must be set by SP B, if errors occur when executing the activity. Is read and output by SP A when it gets the activity again.

RESULT_IS_VISIBLE: Internal use only.

## 4.4 IF_SRM_SRM_SERVICE: Service Object

You can use the service object to request diverse framework services.

To obtain the service object, use `ME->IF_SRM->GET_SRM_SERVICE`. At runtime, the system returns either an object of the class `CL_SRM_SRM_CLIENT_SERVICE_WIN` or an object of the class `CL_SRM_SRM_CLIENT_SERVICE_BSP`. This depends on whether you are working with Windows (SAP GUI) or BSP. You can use a ca st to call all methods of the interfaces that are higher up in the inheritance hierarchy.

The graphic below shows the inheritance hierarchy of the service object:

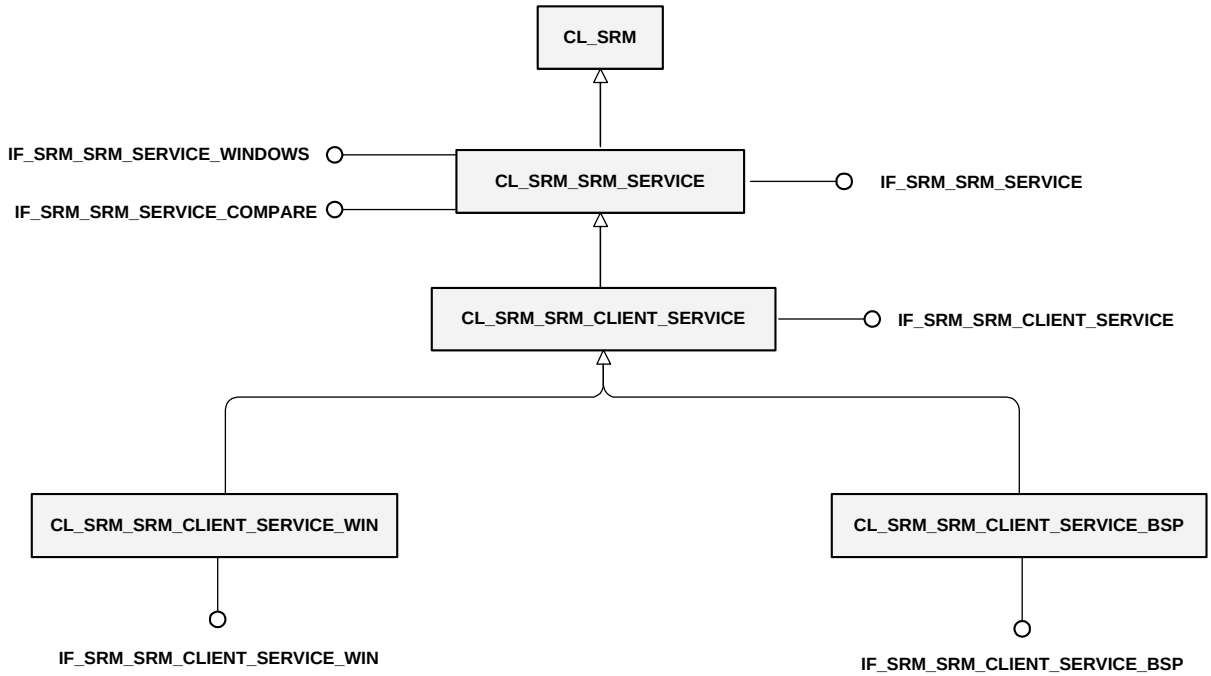



**Figure 6** Inheritance Hierarchy of the Service Object

Note: You cannot progra m using class names. You should only program using interface references, because all classes are supposed to be exchangeable.

### 4.4.1 IF_SRM_SRM_SERVICE

This interface provides general framework services. The services are platform-independent and can also be used by other components in addition to Records Management (the framework can be used by other components as well as by Records Management).

- GET_MODEL_POID: Returns the POID object for any model POID if the AREA POID and SPS POID are specified
- GET_INSTANCE_POID: Returns the POID object for any instance POID if the SPS ID,

AREA POID, and SP POID are specified

Note: You can also obtain a POID object using the interface IF_SRM_SRM_CLIENT_SERVICE (methods POID_GET_INSTANCE and POID_CREATE_MODEL). In contrast to GET_MODEL_POID and GET_INSTANCE_POID, you can enter the RMS ID as a string.

- CONVERT_STRING_TO_POID: Converts a POID that exists as an XML string to a POID object
- CONVERT_POID_TO_STRING: Converts a POID object to an XML string
- GET_POID_DIRECTORY: Returns the interface reference to IF_SRM_POID_DIRECTORY (POID Directory Object, page 1)
- GET_VISUAL_SERVICE_WINDOWS: Returns the interface reference to IF_SRM_SRM_SERVICE_WINDOWS (see below)
- GET_COMPARE: Returns the interface references to IF_SRM_SRM_COMPARE (see below)

The methods listed below request a POID object as an input parameter and return information about the element identified by this POID.

- GET_SP_CONNECTION: Returns the back-end object that fulfills the class role IS_SP_CONTENT_CONNECTION_CLASS
- CHECK_SP_CONNECTION: Checks the availability of the repository object
- GET_ACTIVITIES: Returns the activity list
- GET_DISPLAY_NAME: Returns the display name
- CONNECT_CONTEXT: Returns the context parameter values
- GET_INFO: Returns the information that is displayed in the standard activity “Information” (background list of attribute value objects)
- GET_SAP_ICON_ID: Returns the icon
- GET_VALUE_HELP: Returns an interface reference to IF_SRM_SRM_VALUE_HELP (for the input help)
- GET_VALUE_CHECK: Returns an interface reference to IF_SRM_SRM_VALUE_CHECK (for the value check)
- ENQUEUE_ELEMENT: Locks the element. The import parameter IM_MODE (constant for lock type) can have the following values:
  - IF_SRM_SP_ENQUEUE=>MODE_SHARED: Shared lock, read lock
  - IF_SRM_SP_ENQUEUE=>MODE_EXCLUSIVE: Exclusive lock, write lock
  - IF_SRM_SP_ENQUEUE=>MODE_EXTENDED: Extended lock; extended write lock

  The import parameter IM_SCOPE (constant for validity area) can have the following values:
  - IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG: Lock is automatically lifted when the transaction is ended
  - IF_SRM_SP_ENQUEUE=>SCOPE_DIALOG_AND_UPDATE_TASK: Lock belongs to dialog and update task
  - IF_SRM_SP_ENQUEUE=>SCOPE_PERSISTENT: Persistent lock, can only be lifted by an explicit DEQUEUE
  - IF_SRM_SP_ENQUEUE=>SCOPE_UPDATE_TASK: The update task inherits the lock (type 2)

  Note that not all combinations of mode and scope have to be supported. If combinations are required that are not supported, the called SP must raise an exception of type CX_SRM_SP_ENQUEUE.
- DEQUEUE_ELEMENT: Unlocks the element, parameters are the same as for ENQUEUE_ELEMENT.

### 4.4.2 IF_SRM_SRM_SERVICE_WINDOWS

This interface provides general framework services. The services can only be used if you are using the SAP GUI. These methods can also be used by other components apart from Records Management.

- DISPLAY_INFO: Displays the information of the standard activity *Information* in a popup
- GET_INFO: Returns an interface reference to IF_SRM_VISUAL_INFO_WIN (you can use this to display information in a control)
- DISPLAY_TEXT: Displays any text in a dialog box
- GET_VALUE_HELP: Returns an interface reference to IF_SRM_SRM_VALUE_HELP_WIN (interface for input help)
- GET_ACTIVITIES_CTMENU: Returns a context menu object
- GET_REGISTRY_BROWSER: Returns an interface reference to IF_SRM_REGISTRY_BROWSER (registry browser). The registry browser can be used to determine entities from the registry according to specific criteria, and display them in a dialog box for the user. (Note: The report SRM_DEMO_REGISTRY_BROWSER is available as an example implementation.
- GET_POID_BROWSER: Returns an interface reference to IF_SRM_POID_BROWSER (POID browser). The POID browser can be used to determine POIDs according to specific criteria, and display them in a dialog box for the user to select. The registry browser and the POID browser can be used, for example, for input helps.

### 4.4.3 IF_SRM_SRM_CLIENT_SERVICE

This interface provides services specifically for Records Management. The services are platform-independent.

The following methods return information about a POID:

- POID_CREATE_MODEL: Returns a POID object if the SPS ID and RMS ID are specified
- POID_GET_INSTANCE: Returns an instance POID if the SPS ID, RMS ID, and SP POID are specified

  Note: You can also obtain a POID object using the interface IF_SRM_SRM_ SERVICE (methods GET_MODEL_POID and GET_INSTANCE_POID). In this interface, you only enter the RMS ID as an AREA POID, and not as a string.
- POID_GET_STANDARD_ACTIVITY: Returns the standard activity.
- POID_GET_ACTIVITIES: Returns a list of all activities
- POID_CHECK_ACTIVITY: Checks whether an activity can be executed (for example, an instance activity cannot be called starting from a model POID)
- POID_GET_RMS_ID: Returns the RMS ID for a POID (in contrast to the method IF_SRM_POID~GET_AREA_POID, you receive a string value).
- GET_CUSTOM_SERVICE_MGR: Returns a reference to the Custom Service Manager. A custom service is defined as a service without elements, that is, there is only one instance of the current service (SP with elements without POID). The con nection parameter is the Custom Management Object. To define SPS for this SP, use the transaction srmcustsrv.

The following methods return functions for the authorization check:

- AUTH_CHECK_ACTIVITY: Checks the authorization for an activity
- AUTH_CHECK_VIEW_BY_RMSID: Checks the authorization for list display of an RMS
- AUTH_CHECK_VIEW_BY_SPSID: Checks the authorization for list display of an SPS
- AUTH_CHECK_VIEW_BY_POID: Checks the authorization for list display of a single

element

The following methods are abbreviations for access to the POID Directory and to the POID Relation Directory. The more detailed route leads via the POID Directory Object (see page 1)

- DIRECTORY_SET_POID: Enters a POID object in the POID Directory and returns the PDIR ID
- DIRECTORY_GET_POID: Returns a POID object if the PDIR ID is specified
- DIRECTORY_DEL_POID: Deletes a POID object
- DIRECTORY_SET_POID_RELA: Sets a relation between two POID objects (for example, CT for contains, where-used list)
- DIRECTORY_DEL_POID_RELA: Deletes a relation between two POID objects
- DIRECTORY_GET_POID1_RELA: Returns all the POID objects that are related to a specified POID object In the relations, a difference is noted between POID 1 and POID 2. The transferred POID object is viewed as POID 1.
- DIRECTORY_GET_POID2_RELA: Returns all the POID objects that are related to a specified POID object In the relations, a difference is noted between POID 1 and POID 2. The transferred POID object is viewed as POID 1.
- DIRECTORY_CHECK_POID1_RELA: Checks whether a relation exists for a POID object (POID 1)
- DIRECTORY_CHECK_POID2_RELA: Checks whether a relation exists for a POID object (POID2)

The following methods are shortcuts for accessing information in the registry. The more detailed route leads via the Registry Object (see page 1).

- REGISTRY_GET_RMS_LIST: Returns a list of all RMSs
- REGISTRY_GET_SPS_LIST: Returns a list of all SPS within an RMS
- REGISTRY_GET_SPS_CLASSI_PARA: Returns a classification parameter value for an SPS
- REGISTRY_CHECK_SPS_CLASSI_PARA: Checks whether a classification parameter value exists for an SPS.

### 4.4.4 IF_SRM_SRM_CLIENT_SERVICE_WIN

This interface provides services specifically for Records Management. The services can only be used if you are using a SAP GUI.

- POID_GET_ACTIVITY_VISUAL_STATE: Returns visualization options for an activity of a POID (non-visual, inplace, outplace)
- START_REQUEST_IN_NEW_MODE: Starts a request in a new session. The method uses an SP if it wants to open a new SAPGUI session for displaying the element.
- POID_GET_MENU: Returns a context menu with the activities for a POID
- GET_NEW_DRAGDROPDATA: Returns an interface reference to IF_SRM_DRAGDROPDATA. This method is a shortcut for the method IF_SRM_SRM_CLIENT_OBJ_FACTORY~CREATE_DREAGDROPDATA (see section 1)

### 4.4.5 IF_SRM_SRM_CLIENT_SERVICE_BSP

This interface is only available if you are using Business Server Pages (BSPs).

- REQUEST_GET_URL: Returns the URL for a request object

## 4.5 IF_SRM_POID_DIRECTORY: POID Directory Object

The POID Directory is a persistent store for POID objects.

Each persistent POID object has a POID Directory ID (PDIR ID). This refers uniquely to an entry in the POID Directory from which the POID object can be read if necessary. The PDIR ID is required as follows:

1. As a short key for some APIs for uniquely identifying an element.
2. For an element that writes a where-used list for other elements (for example: a record must be uniquely identified before a where-used list can be written)
3. For elements, for which a where-used list is written in the POID Relation Directory.

For POIDs that are not yet persistent, the PDIR ID is initial. A POID object can be actively entered in the POID Directory using the POID Directory object. You can also choose that every POID is written automatically in the POID Directory. For automatic entry, the framework offers POID Directory automation. You can activate this in registry maintenance: In registry maintenance, open the dialog box for the AREA, go to the “POID Directory” tab page and select the “POID Directory-Automation” indicator.

Note: If a where-used list is written for elements, the system always makes an entry in the POID Directory.

To obtain the POID Directory object, use the service object, method IF_SRM_SRM_SERVICE~GET_POID_DIRECTORY. Access to the POID Directory is performed using additional objects that you can obtain starting from the POID Directory object.

The graphic below provides an overview of th e call hierarchy of the most important interface references. The arrows are to be read as follows: By calling the method in parentheses, you get the interface references to the object to which the arrow points.

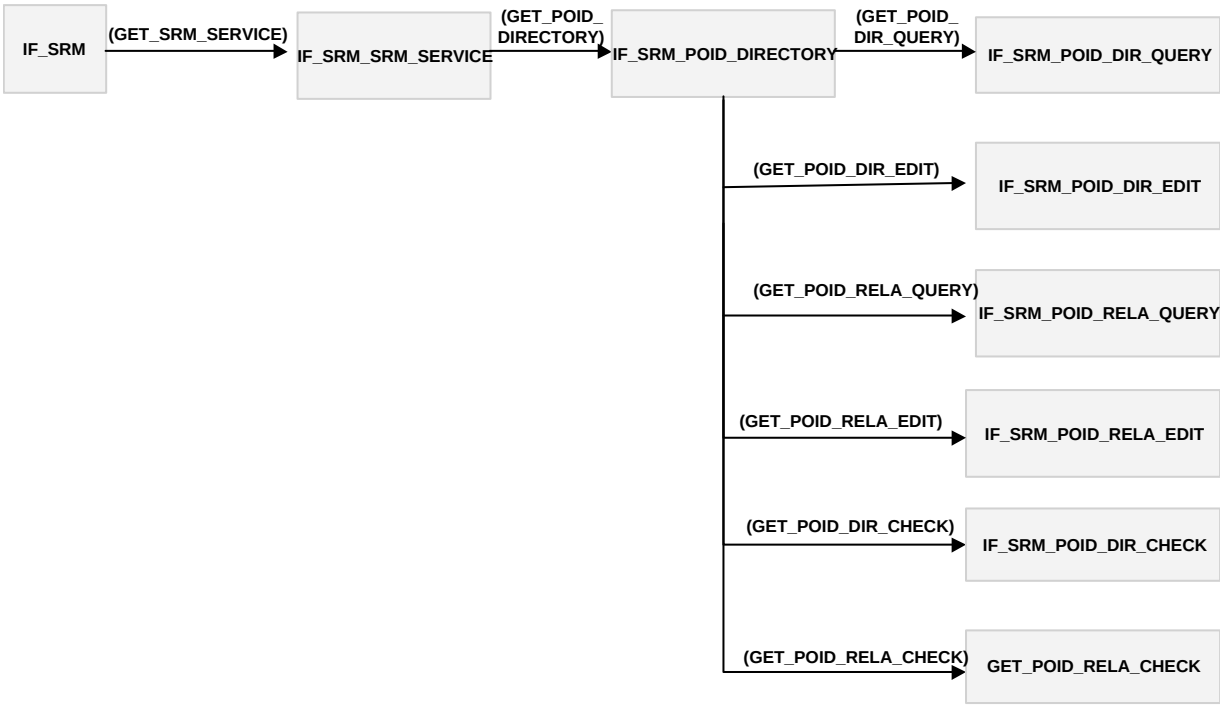



**Figure 7** Access to Interface References Using the POID Directory Object

### 4.5.1 IF_SRM_POID_DIR_QUERY

The method IF_SRM_POID_DIRECTORY~GET_POID_DIR_QUERY returns an interface reference to IF_SRM_POID_DIR_QUERY. This interface groups together methods for read access to the POID Dir ectory. You can read a complete POID object. The interface IF_SRM_POID_DIR_QUERY has the following methods:

- GET_ID_BY_SEARCH_KEY: Returns a list of POID Directory IDs for a search key
- GET_POID_BY_ID: Returns a POID object for a POID Directory ID
- GET_ID: Returns a POID Directory ID for a POID object

### 4.5.2 IF_SRM_POID_DIR_EDIT

The method IF_SRM_POID_DIRECTORY~GET_POID_DIR_EDIT returns an interface reference to IF_SRM_POID_DIR_EDIT . This interface groups together methods for write access to the POID Directory. It has the following methods:

- CREATE: Creates entry

  Note: This method corresponds to the method IF_SRM_SRM_CLIENT_SERVICE~ DIRECTORY_SET_POID.

**Example Code: Writing an Entry in the POID Directory**

```
DATA: service          TYPE REF TO if_srm_srm_service,
      poid_directory   TYPE REF TO if_srm_poid_directory,
      poid_dir_edit    TYPE REF TO if_srm_poid_dir_edit.

service = me->if_srm~get_srm_service( ).
poid_directory = service->get_poid_directory( ).
poid_dir_edit = poid_directory->get_poid_dir_edit( ).

pdir_id = poid_dir_edit->create( im_poid = my_poid
                         im_update_mode = if_srm=>DB_UPDATE ).
...
COMMIT WORK.
```

- DELETE_BY_ID: Deletes an entry if the PDIR ID and AREA ID are specified
- DELETE: Deletes an entry if the POID object to be deleted is specified

  Note: This method corresponds to the method IF_SRM_SRM_CLIENT_SERVICE~ DIRECTORY_DEL_POID.

### 4.5.3 IF_SRM_POID_RELA_QUERY

The method IF_SRM_POID_DIRECTORY~GET_POID_RELA_QUERY returns an interface reference to `IF_SRM_POID_RELA_QUERY`. This interface groups together methods for read access to the POID Relation Directory.

The POID Relation Directory stores the relationships between 2 POID objects. The possible relation types and their meaning are determined by the AREA. To maintain POID relations, use the IMG activity *Maintain Registry*, the dialog box for AREA, tab page *POID Directory*. In addition to the standard relation types (see below), you can also maintain further relations, which you later enter in the POID Relation Directory.

There are 2 relation types for Records Management:

- Relation type “CT” (=contains): Usage relationship
  (for example, record contains document)

  Note: The classification parameter USE only ever refers to CT!
- Relation type “IO” (=Instance of): Template relationship (for example, record is an instance of a record model)

The interface `IF_SRM_POID_RELA_QUERY` has the following methods:

- `GET_POID1_IDS_BY_POID2_ID`: Returns a list of POID 1 IDs for a POID 2 ID
- `GET_POID2_IDS_BY_POID1_ID`: Returns a list of POID 2 IDs for a POID 1 ID

Note: To access the where-used list, you are recommended to use the (simpler) call of the method CL_SRM_HELPER_1=>GET_WHEREUSED( ).

**Example Code: Reading a Relationship from the POID Relation Directory**

```
DATA: service         TYPE REF TO if_srm_srm_service,
      poid_directory  TYPE REF TO if_srm_poid_directory,
      poid_rela_query TYPE REF TO if_srm_poid_rela_query,
      lt_relations    TYPE        srm_list_poid_relations.

service = me->if_srm~get_srm_service( ).
poid_directory = service->get_poid_directory( ).
poid_rela_query = poid_directory->get_poid_rela_query( ).

lt_relations = poid_rela_query->get_poid2_ids_by_poid1_id (
                    im_area_id_poid1 =   'S_AREA_RMS'
                    im_pdir_id_poid2 =   pdir_id_poid_1
                    im_maximum_results = 100
                    im_relation_type =   'IO'
                    im_relation_scope =  if_srm_poid_rela_query
                                         =>relation_type_internal ).
```

### 4.5.4 IF_SRM_POID_RELA_EDIT

The method IF_SRM_POID_DIRECTORY~GET_POID_RELA_EDIT returns an interface reference to IF_SRM_POID_RELA_EDIT. This interface groups together methods for write access to the POID Relation Directory. It has the following methods:

- CREATE: Creates a relation between 2 POID objects (for example: where-used list)
- DELETE: Deletes the relation between 2 specified POID objects.
- DELETE_BY_ID: Deletes the relation between 2 POID objects if the PDIR IDs and AREA IDs of the objects are specified

Note: Writing a where-used list is only useful for service providers that call other service providers (in the graphic “Process Flow of a Request” this is SP A, for example, the SP for records, the Organizer etc., but it cannot be the SP for documents or SP for notes, and so on.)

Whether or not a where-used list is written depends on the Customizing settings for the element type of an element (values for the classification parameter USE: No value entered -> where-used list is written, if the value NOT_ACTIVATED is entered, the where-used list is not written).

SP A always writes a where-used list when it saves its contents. To write a where-used list, use the static method CL_SRM_HELPER_1=>SET_WHEREUSED. This method internally calls the method IF_SRM_POID_RELA_EDIT~CREATE, it then checks the classification parameter USE, and sets the relation type “CT”.

**Example Code: Writing a Where-Used List in the POID Relation Directory**

```
DATA: my_poid TYPE REF TO if_srm_poid.

my_poid = me->if_srm_sp_object~get_poid_object( ).

cl_srm_helper_1=>set_whereused(im_user = my_poid
                               im_used = lt_my_used_objects).

...

COMMIT WORK.
```

### 4.5.5 Commit Work

When an entry is made in the POID Directory or the POID Relation Directory, the database is changed. When methods are called that change the database, you need to enter one of the `IF_SRM=>DB_UPDATE_*` constants for execution of the COMMIT WORK. The constants have the following meanings:

- `IF_SRM=>DB_UPDATE_AND_COMMIT:` COMMIT is executed immediately.
- `IF_SRM=>DB_UPDATE:` No COMMIT is executed (default setting). You have the execute the COMMIT yourself.
- `IF_SRM=>DB_UPDATE_TASK_AND_COMMIT:` The COMMIT is executed by the update task.
- `IF_SRM=>DB_UPDATE_TASK:` COMMIT is not executed by the update task.

## 4.6 IF_SRM_SRM_REGISTRY: Registry Object

The registry object encapsulates the SRM registry. It is an API for the registry (read -access only). To obtain the registry object, use `ME->IF_SRM~GET_SRM_REGISTRY`. You can use the registry object to read all the information that is administered in the registry.

The following graphic provides an overview of the call heirarchy. The arrows are to be read as follows: By calling the method in parentheses, you get the interface references to the object to which the arrow points.

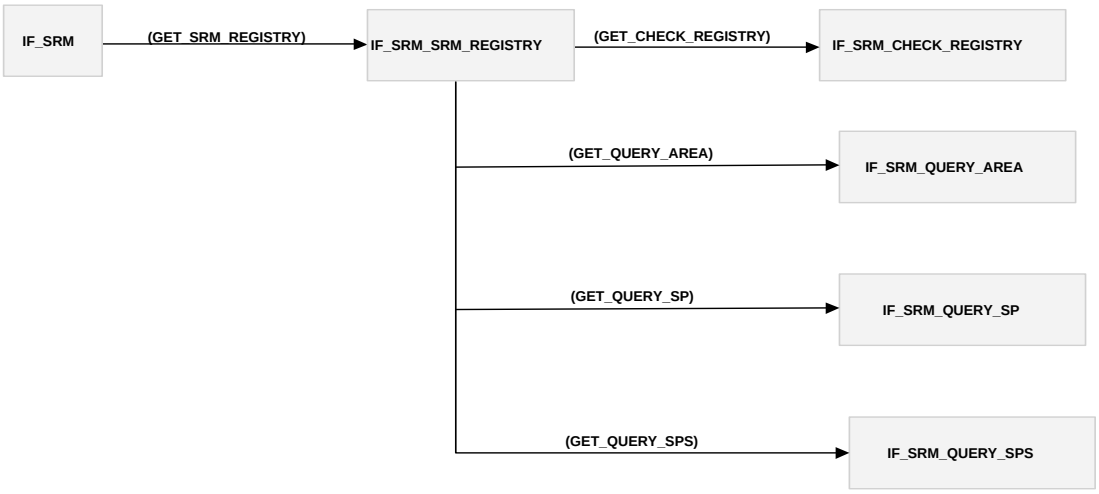



**Figure 8** Access to Interface References Using the Registry Object

### 4.6.1 IF_SRM_QUERY_SPS

You can use the method IF_SRM_SRM_REGISTRY~GET_QUERY_SPS to obtain an interface reference to IF_SRM_QUERY_SPS. You can use this to read information about all the SPSs registered in the registry.

To read information about a specific SPS, use the method IF_SRM_QUERY_SPS->GET_REG_BY_ID. By entering the SPS ID, you receive an interface reference to IF_SRM_SPS_REGISTRY. You can use methods of this interface to access all information on the SPS, for example, read the classification parameter values, read the connection parameter values, read the icons, determine the corresponding SP, determine the corresponding AREA.

For a list of all the SPSs registered in the registry, use the method GET_REG. You receive a new returning parameter of the type SRM_LIST_SPS_REGISTRY. This list contains references to IF_SRM_SPS_REGISTRY.

**Example Code: Reading a List of References to IF_SRM_SPS_REGISTRY**

```
DATA: lt_reg                TYPE        srm_list_sps_registry,
      lif_srm_sps_registry TYPE REF TO  if_srm_sps_registry,
      display_name         TYPE string.
LOOP AT lt_reg INTO lif_srm_sps_registry.
   display_name = lif_srm_sps_registry->get_display_name( ).
   ...
ENDLOOP.
```

### 4.6.2 IF_SRM_QUERY_SP

The method IF_SRM_SRM_REGISTRY~GET_QUERY_SP returns an interface reference to IF_SRM_QUERY_SP. You can use this interface to read information on all the service providers that are registered in the registry.

To read information about a specific service provider, use the method IF_SRM_QUERY_SP->GET_REG_BY_ID. By entering the SP ID, you receive an interface reference to IF_SRM_SP_REGISTRY. You can use the methods of this interface to access all information on the SP, for example, read the SP POID parameter definition, read the connection parameter definition, select the context parameter definition, determine the value check and input help attributes, determine all the class roles filled by the SP, determine the display name and the icon, determine the SP type.

### 4.6.3 IF_SRM_QUERY_AREA

The method IF_SRM_SRM_REGISTRY~GET_QUERY_AREA returns an interface reference to IF_SRM_QUERY_AREA. You can use this interface to read information on all the AREAs that are registered in the registry.

To read information about a specific AREA, use the method IF_SRM_QUERY_AREA->GET_REG_BY_ID. By entering the AREA ID, you receive an interface reference to IF_SRM_AREA_REGISTRY. You can use the methods of this interface to access all the information about the AREA, for example, check the classification parameters for an SPS, access

classification parameter definitions and values, determine the short text, read the POID Directory settings.

### 4.6.4 IF_SRM_CHECK_REGISTRY

The method IF_SRM_SRM_REGISTRY~GET_CHECK_REGISTRY retur ns an interface reference to `IF_SRM_CHECK_REGISTRY`. You can use this to check the existence of all entities registered in the registry.

# 5 Attribute Description Objects and Attribute Value Objects

An attribute description object is a definition for an attribu te. It is similar to a data element in that it specifies the data type, among other things. The following data types are supported: String, Integer, Interface and Date with DDIC reference.

An attribute value object defines concrete instances of an attribut e.

## 5.1 Writing

### 5.1.1 Attribute Description Object

The attribute description object for write access is represented through the interface IF_SRM_EDIT_ATTRIBUTE_DESC.

You get hold of the reference to this interface either through the Factory object (see page 1) or using a cast starting from IF_SRM_ATTRIBUTE_DESC (interface for read access to the attribute description object).

The following methods are available to you on the interface IF_SRM_EDIT_ATTRIBUTE_DESC for describing an attribute:

- SET_GENERAL_DESCRIPTION( ): Sets the general description of the attribute. Enter a structure of type SRMADGEN as an import parameter. This is composed of the following fields:
  - ID: ID of the attribute
  - TEXT: Text of the attribute
  - TYPE: Data type (String/In teger/Interface/DDIC Field) Set one of the constants IF_SRM_ATTRIBUTE_DESC~*
  - IS_LIST: true[1] -> The user can enter multiple attribute values.
  - IS_MAND: true -> Assigning an attribute value is mandatory.
  - IS_CHECK: true -> A value check is to be performed for the attribute.
  - IS_HELP: true -> A value help is to be offered for the attribute.

  The following checkboxes are only relevant as part of property unification. By default, the checkboxes are set to *false*.

[1] As a value for all checkboxes, you assign one of the constants IF_SRM=>TRUE or IF_SRM=>FALSE.

- IS_MAINTAIN: true -> A value can be assigned to the attribute. This can be done by using direct input, input help or can be done in the background. You can define whether the field is to be ready for input or not, in the visualization description.
- IS_ONCE_MAINTAIN: true -> The user may enter one attribute v alue only. The value can then no longer be changed.
- IS_LANG_SENS: true -> The attribute value is stored language-dependently.
- IS_GENERATE: true -> The a ttribute value must be able to be generated. (The user can start the generation by using the input help button.) Prerequisite: You must have already implemented the class role VALUE_HANDLER.
- IS_VISIBLE: true -> The a ttribute is visible in the attribute maintenance dialog. You also have to set the visualization description for the attribute maintenance dialog.
- IS_VIS_IN_LIST: true -> The a ttribute is visible in the hit list. You also have to set the visualization description for lists.
- IS_QUERY: true -> The a ttribute is visible in the search dialog. You also have to set the visualization description for the search dialog.
- Changes of the attribute value are logged. Prerequisite: You must have already implemented the class role IS_SP_PROTOCOL_HANDLER. For more information on this, see *Logging* on page 67.
- UNIT: Name of a mass unit. Only relevant for attributes that require a unit. Examples: km, t, l . In the attribute maintenance dialog the unit is added to the value.
- ALIAS-ID: Has no significance in the standard delivery.
- REPOS-ID: Has no significance in the standard delivery.
- SET_XYZ_DESCRIPTION( ): Sets the data type -specific description of the attribute ("XYZ" stands for STRING, INTEGER, INTERFACE or TABFIELD). Depending on the data type, enter a structure of type SRMADSTR, SRMADINT, SRMADIF or SRMADTAFIN for the import parameter.

The following methods are only relevant as part of property unification:

- SET_VISUAL_DESCRIPTION( ): Sets the visual description of the attribute for the attribute maintenance dialog. Enter a structure of type SRMADVIS as an import paramet er. This is composed of the following fields:
  - ROW_NO: Rows in which the attribute is to be displayed.
  - COLUMN_NO: Columns in which the attribute is to be displayed.
  - FIELD_TYPE: Form of appearance of the a ttribute (such as dropdown list box). Use one of the constants IF_SRM_ATTRIBUTE_DESC_VISUAL~*.
  - IS_EDIT: true -> The field for the a ttribute value is ready for input. Prerequisite: In the general attribute description, you have to have set the checkbox IS_MAINTAIN to true.
  - IS_EMPHASIZED: true -> The attribute is displayed in bold.
  - GROUP_NO: Number of a group , to which the attribute is to be assigned. In the standard delivery, the groups are not visualized.
  - GROUP_LABEL: Header for the group.
  - BUTTON_TAB: Button that is to be displayed in front of the attribute. If the attribute has multiple values, a button can be displayed before each new row.
- SET_QUERY_DESCRIPTION( ): Sets the visual description of the attribute for the search dialog. Enter a structure of type SRMADQUE as an import parameter. The fields of the structure correspond to those of the structure for the visualization description of the attribute maintenance dialog.

- SET_VIS_IN_LIST_DESCRIPTION( ): Sets the visual description of the attribute for a list, such as the hit list. Enter a structure of type SRMADVLS as an import parameter. This is composed of the following fields:
  - COLUMN_NO: Number of columns in which the attribute is to be displayed.
  - IS_EMPHASIZED: true -> The attribute is displayed in bold.
- UPDATE_GENERAL_*( ): Overwrites the value for the corresponding checkbox in the structure SRMADGEN. You can call this method at runtime.

### 5.1.2 Attribute Value Object

The attribute value object for write access is represented through the interface IF_SRM_EDIT_ATTRIBUTE_VALUE.

You get hold of the reference to t his interface either through the Factory object (see page 1) or using a cast starting from IF_SRM_ATTRIBUTE_VALUE (interface for read access to the attribute description object).

To set a value for an attribute value object , y ou must call 2 methods for the interface IF_SRM_EDIT_ATRIBUTE_VALUE:

1) SET_DESCRIPTION( ): Sets the attribute description. Enter the attribute description object that you just filled as the import parameter.
2) SET_XYZ_VALUE( ): XYZ stands for STRING, INTEGER o r INTERFACE. Sets the attribute value. As the import parameter, enter a table of type `SRM_LIST_STRING`, `SRM_LIST_INTEGER`, or `SRM_LIST_OBJECT`. For fields with DDIC references, use the method instead

   SET_DDIC_XYZ( ): XYZ stands for STRING, XSTRING, MEASURE or CURRENCY. Sets the attribute value with DDIC reference. Which method is read depends on the type of the attribute value in the DDIC (STRING = simple type, XSTRING = binary data, MEASURE = unit of measurement, CURRENCY = currency).

## 5.2 Reading

### 5.2.1 Attribute Description Object

The attribute description object for read access is represented through the interface IF_SRM_ATTRIBUTE_DESC.

When you call methods for reading attribute descriptions, a parameter of type SRM_LIST_ATTRIBUTE_DESC is returned. This is a table type with interface references to IF_SRM_ATTRIBUTE_DESC, that is, a list of attribute description objects.

The interface IF_SRM_ATTRIBUTE_DESC provides the following methods:

- GET_GENERAL_DESCRIPTION( ): Returns the general attribute description
- GET_XYZ_DESCRIPTION( ): XYZ stands for STRING, INTEGER, INTERFACE or TABFIELD. Returns the data-type-specific attribute description information.
- GET_VISUAL_DESCRIPTION( ): Returns the visualization description for the attribute maintenance dialog.
- GET_VIS_IN_LIST_DESCRIPTION( ): Returns the visualization description for lists.

- GET_QUERY_DESCRIPTION( ): Returns the visualization description for the search dialog.
- GET_ID( ): Returns the ID of the attribute.
- GET_STATE( ): Returns the state of the attribute. What is retu rned is one of the constants STATE_INITIAL (attribute description not yet set), STATE_GENERAL_SET (attribute description already set) or STATE_COMPLETE (attribute description and attribute value set, attribute description can no longer be changed).
- IS_OF_TYPE( ): Checks the data type.

### 5.2.2 Attribute Value Object

The attribute value object for read access is represented through the interface IF_SRM_ATTRIBUTE_VALUE.

When you call methods for reading attribute values, a parameter of type SRM_LIST_ATTRIBUTE_VALUE is returned. This is a table type with an interface reference to IF_SRM_ATTRIBUTE_VALUE, that is, a list of attribute value objects. One attribute value object always refers to exactly one attribute description object.

The interface IF_SRM_ATTRIBUTE_VALUE provides the following methods:

- GET_ID( ): Returns the ID of the attribute.
- GET_DESCRIPTION( ): Returns a reference to the corresponding attribute description object (IF_SRM_EDIT_ATTRIBUTE_DESC).
- GET_XYZ_VALUE, GET_DDIC_XYZ( ): Reads the attribute value (ana logous to writing the attribute value, see above).
- GET_STATE( ): Returns the state of the attribute. The system returns one of the constants STATE_INITIAL (attribute value not yet set), STATE_DESCRIPTION_SET (attribute description set) or STATE_COMPLETE (attribute description and attribute value set).

**Example Code: Reading a List of Attribute Value Objects of Type String**

```
DATA: lo_attr_val  TYPE REF TO if_srm_attribute_value,
      lt_attr_val  TYPE        srm_list_attribute_value,
      lt_strings   TYPE        srm_list_string,
      wa_string    TYPE        srmliststr.
…
LOOP AT lt_attr_val INTO lo_attr_val.
   lt_strings = lo_attr_val->get_string_value( ).
   loop at lt_strings into wa_string.
** The data is processed here
   endloop.
ENDLOOP.
```

# 6 Property Unification

Property unification is a service which a Service Provider can use to define attributes for its elements that ca n be displayed in the standard attribute maintenance dialog and the search

dialog. The attribute values can be read in background processing, set and printed.

Here, two different cases are possible:

a) The attributes of your Service Provider have their own attribute repository.

You have to implement the connection to the Property Unification, as well as the call for Property Unification Services.

b) The attributes of your Service Provider do not have their own attribute repository.

You only have to implement the call for Property Unification Services. You can use the default attribute repository.

You can use a standard implementation delivered by SAP for the connection to Property Unification. To define the attributes, all you have to do is perform the activities under Customizing Attribute in the IMG. Online documentation for these is available in the system.

## 6.1 Connecting to Property Unification

To connect your attribute repository to Property Unification, you have to implement the following class roles:

- IS_SP_PROP_REPOSITORY
- IS_SP_PROP_VIS_DEFINE
- IS_SP_PROP_QUERY_DEFINE
- IS_SP_PROP_VALUE (optional)

SAP delivers a default implementation for each class role. You can find these in the package SRM_PROPERTY. You can use the classes of the package as a reference implementation. If you want to, you can allow your classes to inherit from the default classes, redefining a few methods only. You can redefine the methods listed below.

Notes:

- You obtain and transfer the attributes and attribute values in the form of attribute description objects and attribute value objects. For more information about this, see Attribute description objects and attribute value objects on page 51.
- In all methods, you get the context object as import parameter. This channel can be used to pass on information. This parameter is not listed in the subsequent method descriptions.

Class that fulfills class role IS_SP_PROP_REPOSITORY:

- IF_SRM_SP_PROP_REPOS_META~GET( ): You determine the properties of the attributes in your repository. Returning PROPERTY_TAB: List of attribute value objects. For each attribute, you set the general attribute description in the relevant attribute description objects.
- IF_SRM_SP_PROP_REPOS_META~SINGLE_GET( ): Importing PROPERTY -ID: ID of an attribute. You determine the properties of this attribute in your repository. Returning PROPERTY: Attribute value object. You set the general attribute description in the relevant attribute description object.
- IF_SRM_SP_PROP_REPOS_DATA~GET( ): Importing PROPERTY_TAB: List of attribute value objects. For this, the general attribute description is already set. You determine the attribute values for the current POID and set these for the attribute value objects.
- IF_SRM_SP_PROP_REPOS_DATA~SET( ): Importing PROPERTY_TAB: List of attribute value objects that contain the attribute values entered by the user. You save these for the

current POID in your repository.

- IF_SRM_SP_PROP_REPOSITORY~QUERY( ): Importing QUERY_TAB: List of attributes valuated by user in the search dialog. You perform the search in your repository. Returning RESULT: Structure describing the results list. You are given two tables: PROPERTY_DESC_TAB: List of attribute description objects that defines which attributes are to be displayed in the columns of the results list. RESULT_TAB: Table containing an ID for each result row, as well as a list of attribute values of the result.
- IF_SRM_SP_PROP_REPOSITORY~QUERY_RESULT_DETAIL_GET( ): Importing ID: ID of the result row selected by the user. Returning RESULT_DETAIL: POID of the element of this result row.
- IF_SRM_SP_PROP_REPOSITORY~LOCK( ): Importing SHOW_LOCKED_POPUP: Flag for whether a dialog box is to be displayed, showing which user is causing the lock. You set an access lock in your repository and trigger the dialog box if required.
- IF_SRM_SP_PROP_REPOSITORY~UNLOCK( ): You remove the access lock.

Class that fulfills class role IS_SP_PROP_VIS_DEFINE:

- IF_SRM_PROP_VIS_DEFINE~GET( ): Importing PROPERTY_TAB: List of attribute value objects. For this, the general attribute description is already set in the relevant attribute description objects. You now set the visualization description for the attribute maintenance dialog.
- IF_SRM_PROP_VIS_LIST_DEF~GET( ): Changing PROPERTY_DESCRIPTION_TAB: List of attribute description objects. For this, you set the visualization description for lists (such as for results lists).

Class that fulfills class role IS_SP_PROP_QUERY_DEFINE:

- IF_SRM_SP_PROP_QUERY_DEFINE~GET( ): Changing PROPERTY_TAB: List of attribute value objects. For this, the general attribute description is already set in the relevant attribute description objects. You now set the visualization description for the search dialog.

Class that fulfills class role IS_SP_PROP_VALUE:

- IF_SRM_PROP_VALUE~CHECK_IS_INITIAL( ): Importing VALUE: Attribute value entered by the user. Importing PROPERTY_DESCRIPTION: Attribute description object for this value. Returning IS_INITIAL: Flag for whether this value is the initial value of the attribute. This information is required if the value input is mandatory.
- IF_SRM_PROP_VALUE~CONVERSION_OUT( ): Importing IN: Attribute value entered by the user. Importing PROPERTY_DESCRIPTION: Attribute description object for this value. You define a conversion which the attribute value has to run through before it is displayed. Returning OUT: Value in changed format.
- IF_SRM_PROP_VALUE~GET_TEXT( ): Importing VALUE: Current attribute value. Importing PROPERTY_DESCRIPTION: Related attribute description object. You define a text that is displayed to the right of the attribute input field. This may be the attribute value in another form, for example. Returning TEXT: Text to be displayed.

All of the methods described in the following have the changing parameter VAL_HDL_PROPERTY_TAB. This is a list of attribute value objects with additional checkboxes. The checkboxes have the following meanings:

- IS_REQUESTED: true[2] -> This attribute expects the action (flag set for import)

[2] As a value for all checkboxes, you assign one of the constants IF_SRM=>TRUE or

- REQUESTED_VALUE_IDX: Index of attribute value for multi -value attributes (set f or import)
- RESULT_OK: true -> Action was performed successfully or the user canceled (flag to be set by you).
- RESULT_ERR_TEXT: Error text (to be set by you if you have set RESULT_OK to FALSE)
- UPDATED: true -> User has selected a new value (flag to be set by you)
- HANDLER_FOUND: true -> The event handler has been found, according to the details in the attribute description object (flag to be set by you)

- IF_SRM_PROP_VALUE~EXECUTE_HELP( ): Only relevant if you have checked input help for at least one attribute in the general attribute description, and do not want to use the DDIC input help. Changing VAL_HDL_PROPERTY_TAB (see above). You implement input help for the attribute whose IS_REQUESTED flag has been set to TRUE. The values of all other attributes are available to you in the input help for the implementation.
- IF_SRM_PROP_VALUE~EXECUTE_CHECK( ): Only relevant if you have checked the value check for at least one attribut in the general attribute description. Changing VAL_HDL_PROPERTY_TAB (see above). You im plement the value check for the attribute whose IS_REQUESTED flag has been set to TRUE.
- GET_DROP_DOWN_VALUES( ): Only relevant of you have set a dropdown list box for at least one attribute in the visualization description. Importing VAL_HDL_PROPERTY_TAB (see above). You return the dropdown list box for the attribute whose IS_REQUESTED flag has been set to TRUE. Returning DROPDOWN_VALUES: List of texts.
- IF_SRM_PROP_VALUE~EXECUTE_GENERATE( ): Only relevant if you have checked the value generation for at le ast one attribut in the general attribute description. Changing VAL_HDL_PROPERTY_TAB (see above). You implement a generation mechanism for the value of the attribute whose IS_REQUESTED flag has been set to TRUE.
- IF_SRM_PROP_VALUE~EXECUTE_BUTTON_CLICK( ): Only relevant if you have defined for at least one attribute in the visualization description, that a button is to appear before the attribute. Importing BUTTON_ID. You implement event handling for the button. Returning VAL_HDL_PROPERTY_TAB (see above).

## 6.2 Calling Property Unification Services

To get used to this topic, we recommend that you study the example programs SRM_PROP_UNIFICATION_HOWTO and SRM_PROP_UNIFIC_QUERY_HOWTO. The following documentation refers to these programs. Note: You can execute these pr ograms only if the values of the constants exist in the system. If you want to execute the program, you must adjust the constants in the program.

Prerequisite for calling the services: you are in the Records Management context (the Framework has already b een instanced) and you know the instance POID of the element whose attributes you want to display or change. In the example program SRM_PROP_UNIFICATION_HOWTO, this information is first generated (see subprograms connectFramework and createRecordInstancePoid).

For all services, you require a reference to the Property Service object (IF_SRM_SRM_SERVICE_PROP). You can obtain this through the service object method IF_SRM_SRM_SERVICE~get_property_service( ). You also require a reference to the context object ( if_srm_prop_context). In the context object, you set the information that is relevant for each service. You can obtain the context object through the Property Service object method IF_SRM_SRM_SERVICE_PROP~get_context( ).

---

IF_SRM=>FALSE.

### 6.2.1 Calling the Standard Attribute Maintenance Dialog

You can call a synchronous control or an asynchronous control. The synchronous control saves the attributes directly when the user activates the green checkmark. The asynchronous control does not save directly when the green checkmark is act ivated. Instead, the changed attribute values are stored temporarily but not written to the database. They are saved in a second step.

**Calling the Synchronous Control:**

You obtain a reference to the synchronous control through IF_SRM_SRM_SERVICE_PROP~get_ctl_sync( ). In parameter IM_POID, you specify the instance POID of the element whose attributes are to be displayed or changed.

You set information for calling the control in the context object. You can set the following information:

- IF_SRM_PROP_CONTEXT_VIS~MODE_SET( ): You set the call mode of the control (display or change). Use one of the constants IF_SRM_PROP_CONTEXT_VIS~MODE_*. No default exists, calling this method is mandatory.
- IF_SRM_PROP_CONTEXT_VIS~PLACE_SET( ): You define whether the control is to be displayed inplace or outplace. Use one of the constants IF_SRM_PROP_CONTEXT_VIS~PLACE_*. If you select the inplace display, you also have to call the method IF_SRM_PROP_CONTEXT_VIS~PARENT_CONT_SET( ) and specify the parent container. The outplace display is the default.
- IF_SRM_PROP_CONTEXT_VIS~UI_SET( ): You define the user interface. Use one of the constants IF_SRM_PROP_CONTEXT_VIS~UI_*. The display in the SAP GUI is the default.
- IF_SRM_PROP_CONTEXT_VIS~USER_OBJECT_SET( ): This method gives you th e option of transmitting information using a user object. This only makes sense if you are specializing the control and evaluating the information.

You call the attribute maintenance dialog through the method of the synchronous control IF_SRM_SRM_PROP_CTL_SYNC->EXECUTE. You transfer the context object.

You can find an example implementation in program SRM_PROP_UNIFICATION_HOWTO, examples 0 and 1.

**Calling the Asynchronous Control:**

You obtain a reference to the asynchronous control through IF_SRM_SRM_SERVICE_PROP~get_ctl_Async( ). As was the case for synchronous control, you set the information in the context object. Method IF_SRM_SRM_PROP_CTL_ASYNC~EXECUTE( ) calls the control but does not perform any saving function. You can use IF_SRM_SRM_PROP_CTL_ASYNC ~PROPERTY_TAB_GET( ) to request a list of the changed attributes and their values.

To save the attributes, you first have call method IF_SRM_SRM_PROP_CTL_ASYNC~ FLUSH( ) and then RELEASE( ).

You can find an example implementation in program SRM_PROP_UNI FICATION_HOWTO, example 2.

### 6.2.2 Calling the Standard Search Dialog

You obtain a reference to the query control through IF_SRM_SRM_SERVICE_PROP~get_ctl_QUERY( ). In parameter IM_POID, you specify the model POID on which the search is to by performed.

You set th e information that is relevant for the search in the context object. You can set the following information:

- IF_SRM_PROP_CONTEXT_VIS~PLACE_SET( ): You define whether the control is to be displayed inplace or outplace. Use one of the constants IF_SRM_PROP_CONTEXT_VIS~PLACE_*. If you select the inplace display, you also have to call the method IF_SRM_PROP_CONTEXT_VIS~PARENT_CONT_SET and specify the parent container. The outplace display is the default.
- IF_SRM_PROP_CONTEXT_VIS~UI_SET( ): You define the user in terface. Use one of the constants IF_SRM_PROP_CONTEXT_VIS~UI_*. The display in the SAP GUI is the default.
- IF_SRM_PROP_CONTEXT_QUERY~MAX_HITS_SET( ): Maximum number of results to be displayed. The default is set to 200 results.
- IF_SRM_PROP_CONTEXT_QUERY~CASE_SENSITIVE_SET( ): Whether search is to be case-sensitive. The default is set to case-sensitive.
- IF_SRM_PROP_CONTEXT_QUERY~CURRENT_VERSION_ONLY_SET( ): Whether only the current versions of the elements in the results list are to be displayed. (Only a sen sible option if elements are versioned.) The default is set to current versions only.
- IF_SRM_PROP_CONTEXT_QUERY~FUZZY_SEARCH_SET( ): Whether a fuzzy search is to be used. (Can only be set if the attribute repository supports this feature.) The default is f or the search not to be performed as a fuzzy search.
- IF_SRM_PROP_CONTEXT_QUERY~NEAR_BY_SET( ): Whether a NEAR BY is to be used. (Can only be set if the attribute repository supports this feature.) The default is for the search not to be performed as a NEAR BY search.

You can use IF_SRM_PROP_CONTEXT_QUERY~ALL_SET( ) to transfer the information listed above summarized in a table.

You call the search dialog using the method of the query control IF_SRM_SRM_PROP_CTL_QUERY~execute( ). You transfer the context o bject and receive in return the result selected by the user.

For an example implementation, see the SRM_PROP_UNIFIC_QUERY_HOWTO program.

### 6.2.3 Attribute Operations in Background Processing

For operations in background procesisng, you require the repository hand ler. You obtain this through IF_SRM_SRM_SERVICE_PROP~get_repository( ). In the import parameter IM_POID, you specify the instance POID of the element whose attributes you want to perform operations on.

For write access, you have to set a lock in the repo sitory handler prior to performing any operation, using IF_SRM_SRM_PROP_REPOSITORY~lock( ). You can use IF_SRM_SRM_PROP_REPOSITORY~unlock( ) to remove this lock following the operation.

You can use IF_SRM_SRM_PROP_REPOS_META~get( ) to read the attributes and the values in the repository handler. You obtain a list of attribute value objects in return. You perform the

operations on these. For information about working with attribute value objects, see page 51.

You can find example implementations for changing an attribute value, changing multiple attribute values, reading an attribute value and reading all attribute values in program SRM_PROP_UNIFICATION_HOWTO, examples 3 to 6.

### 6.2.4 Search in Background Processing

For operations in b ackground procesisng, you require the repository handler. You obtain this through IF_SRM_SRM_SERVICE_PROP~get_repository( ). In import parameter IM_POID, you specify the model POID on which the search is to by performed.

You set information for the search in the context object. You can call all methods of the interface IF_SRM_PROP_CONTEXT_QUERY (see Calling the Standard Search Dialog).

You then fill a table of the type SRM_QUERY_EXPRESSION_TAB( ) with the actual search parameters. You use method IF_SRM_SR M_PROP_REPOSITORY->query( ) to execute the search. As parameters, you transfer the context object and the table of search parameters.

You can find an example implementation in program SRM_PROP_UNIFIC_QUERY_HOWTO, example 8.

### 6.2.5 Printing Attributes

You obtain the repository handler, from where you read the list the attributes (see above). You read the visualization description for these attributes.

In the context object you set print mode by using IF_SRM_PROP_CONTEXT_PRINT~MODE_SET( ). Use one of the constants IF_SRM_PROP_CONTEXT_PRINT~MODE_*. The constants have the following meanings: MODE_SEPARATE: The spool request is set immediately. MODE_EMBEDDED: The spool request is not set immediately. You can bundle multiple requests and then set one spool request for them all.

Through the Property Service object you obtain a reference to IF_SRM_SRM_PROP_PRINT and call method EXECUTE( ) in this object. You transfer the context object and the list of attributes.

You can find an example implementation in program SRM_PRO P_UNIFICATION_HOWTO, example 7.

# 7 Optional Framework Services

## 7.1 Input Help

The framework provides a service for implementing input help. Implementation of input help is divided into two areas:

- Server Integration

  Server integration of input help means that the server SP is called by a client. The server SP executes the input help and returns the value selected by the user to the calling client.

  Application example: When entering a value for a c onnection parameter in the registry, the

user is offered input help. The SP that published the connection parameter is the server, the registry is the client.

- Client integration

  Client integration of the input help is defined as follows: The client SP re quests a value, which was determined by the input help for a particular attribute, from a server SP. Input helps can only be requested for attributes of which the server SP has published the existence. This means the attributes must be context parameters, connection parameters, or POID parameters (see section 15).

  Application example: When entering an attribute value in the attribute maintenance dialog, the user is offered an input help (the values of this attribute are determine d by a different SP). The SP in which the attribute value is entered is the client SP. The SP that returns the values is the server SP.

The basis class CL_SRM_SP_VALUE_HELP is provided for input help. The following graphic provides an overview of the inher itance hierarchy of the basis class. The class displayed at the bottom is a class that you must implement for server integration:

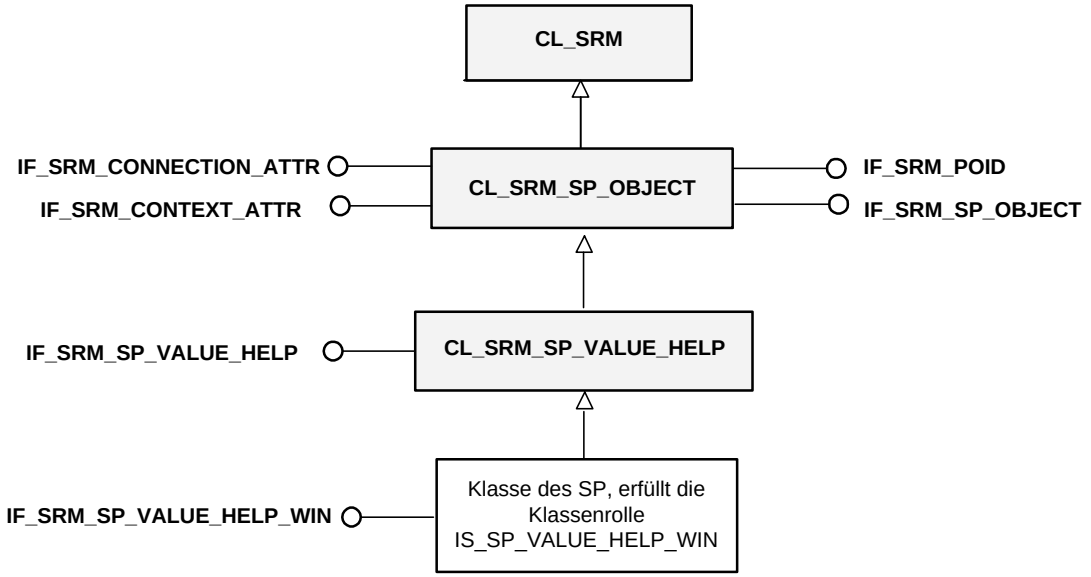



**Figure 9** Inheritance Hierarchy of the Basis Class for the Input Help

The interface IF_SRM_SP_VALUE_HELP of t he basis class CL_SRM_SP_VALUE_HELP provides methods for reading and writing parameters, which influence the input help. The server SP must read and process these values. The client SP must set the values.

| Method | Description |
|---|---|
| SET_ATTRIBUTE | Set: Attribute description for which input help is required. Must be set by the client. |
| SET_SELECTION_MAX_NUMBER | Set: Maximum number of values to be selected. Can be set: Default = 1 |
| SET_SELECTION_MIN_NUMBER | Set: Minimum number of values to be selected. Can be set by the client: Default = 1 |
| SET_LIMIT_INSTANCE | Set: Selection of possible values restricted to instance. If IF_SRM=>TRUE, the server must read the SP POID parameters and restrict the possible value selection to these values. Can be set by client: Default = |

| | |
|---|---|
| | IF_SRM=>FALSE |
| SET_LIMIT_SPS | Set: Selection of possible values restricted to SPS. . If IF_SRM=>TRUE, the server must read the connection parameter values and restrict the possible value selection to these values. Can be set by client: Default = IF_SRM=>FALSE |
| GET_ATTRIBUTES | Get: List of all attributes with input help |
| GET_SELECTION_MAX_NUMBER | Get: Maximum number of values to be selected. |
| GET_SELECTION_MIN_NUMBER | Get: Minimum number of values to be selected. |
| GET_LIMIT_INSTANCE | Get: Selection of possible values restricted to instance. |
| GET_LIMIT_SPS | Get: Selection of possible values restricted to SPS. |
| GET_ATTRIBUTE | Get: Attribute description for which input help is required. |

### 7.1.1 Server Integration

The following steps are required for server integration of the input help:

1) When publishing the POID parameter, connection parameter, or context parameter for which you want to offer an input help, enter the value IF_SRM=>TRUE in the general attribute description for the field IS_HELP.
2) Implement the class role of t he input help IS_SP_VISUAL_VALUE_HELP_WIN. To implement the class role, create a class that inherits from the basis class CL_SRM_SP_VALUE_HELP, and register this class in the registry for your SP.
3) Declare the interface IF_SRM_SP_VALUE_HELP_WIN for the c lass you have just created. The interface only has one method: IF_SRM_SP_VALUE_HELP_WIN~EXECUTE_SELECTION. You must implement this method.

**IF_SRM_SP_VALUE_HELP_WIN~EXECUTE_SELECTION( )**

| RETURNING | RE_ATTR_VALUES | TYPE | SRM_LIST_ATTRIBUTE_VALUE |
|---|---|---|---|

This method exe cutes the input help. The return parameter contains exactly one attribute value object with the value selected by the user. The following steps must be executed:

- Get values for the input help
- Call a dialog box for the value selection and transfer the values
- Get the attribute description object using the method IF_SRM_SP_VALUE_HELP~GET_ATTRIBUTE
- Set the attribute description object for the attribute value ob ject
- Set the value selected by the user for the attribute value object (a table of type srm_list_string is filled)
- Enter the attribute value object in the list (returning parameter)

**Example Code with Commentary**

METHOD if_srm_sp_value_help_win~execute_selection.

```
DATA:    lo_valuehelp    TYPE REF TO if_srm_sp_value_help,
         lo_attrdesc     TYPE REF TO if_srm_attribute_desc,
```

```
        lo_factory      TYPE REF TO if_srm_srm_object_factory,
        lo_attrvalue    TYPE REF TO if_srm_edit_attribute_value,
        lt_list_string  TYPE srm_list_string.

** -> Display input help selection box and return the value selected by the user
** in the field VALUE of the table lt_list_string.
   ...


** Request attribute description object
  lo_valuehelp ?= me.
  lo_attrdesc = lo_valuehelp->get_attribute( ).

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

** Request attribute value object
  lo_attrvalue = lo_factory->create_attr_value( ).


** Set attribute description object for the attribute value object
   lo_attrvalue->set_description( lo_attrdesc ).


** Set value selected by the user for the attribute value object
   lo_attrvalue->set_string_value( lt_list_string ).
   CLEAR lt_list_string.

** Enter the attribute value object in the list (returning parameter)
   APPEND lo_attrvalue TO re_attr_values.

ENDMETHOD. "IF_SRM_SP_VALUE_HELP_WIN~EXECUTE_SELECTION
```

### 7.1.2 Client integration

For client integration of the value help, proceed as follows:

Get a service object and use it to call the method IF_SRM_SRM_SERVICE_WINDOWS -> GET_VALUE_HELP. As the import parameter, enter the POID object of the server SP that delivers the input help. As a returning parameter, you receive an interface reference to IF_SRM_SRM_VALUE_HELP. This interface has the methods described below:

| Method | Description |
|---|---|
| GET_SETTINGS | Returns a reference to the interface IF_SRM_SP_VALUE_HELP, described above. |
| EXECUTE_SELECTION | Executes the input help |

**Example Code with Commentary**

```
Data:  lo_servicewindows TYPE REF TO if_srm_srm_service_windows,
       lo_srmvaluehelp   TYPE REF TO if_srm_srm_value_help_win,
       lo_poid           TYPE REF TO if_srm_poid,
       lo_spvaluehelp    TYPE REF TO if_srm_sp_value_help,
        lt_attrdesctab   TYPE        srm_list_attribute_desc,
        lo_attrdesc      TYPE REF TO if_srm_attribute_desc,
        lt_attrvalue     TYPE        srm_list_attribute_value.
```

```
** Request service object
   lo_servicewindows ?= me->get_srm_service( ).

** Request the input help object for a POID of the server SP
   lo_srmvaluehelp = lo_servicewindows->get_value_help( lo_poid ).

** Request the server SP-specific interface reference
   lo_spvaluehelp = lo_srmvaluehelp->get_settings( ).

** Request the attribute description object of which the server SP
** published the input help
   lt_attrdesctab = lo_spvaluehelp->get_attributes( ).

** Select the attribute description object, of which the parameters for
** the input help should be executed
   LOOP AT lt_attrdesctab INTO lo_attrdesc.
      IF lo_attrdesc->get_id( )
          = '<ID des Parameters>'.
        EXIT.
      ENDIF.
   ENDLOOP.

** Set the attribute description object
   lo_spvaluehelp->set_attribute( lo_attrdesc ).

** Execute the input help and enter the attribute value object with the value
** selected by the user in a single row table
   lt_attrvalue = lo_srmvaluehelp->execute_selection( ).
```

## 7.2 Value Check

In addition to the input help, the framework also provides a value check. This integration is divided into two areas:

- Server Integration

  The server SP checks whether the value entered by the user exists.

- Client integration

  The client SP checks the value entered by the user according to its own criteria.

The basis class CL_SRM_SP_VALUE_CHECK is provided for the value c heck. The following graphic provides an overview of the inheritance hierarchy of the basis class. The class displayed at the bottom is class that you must implement for server integration:

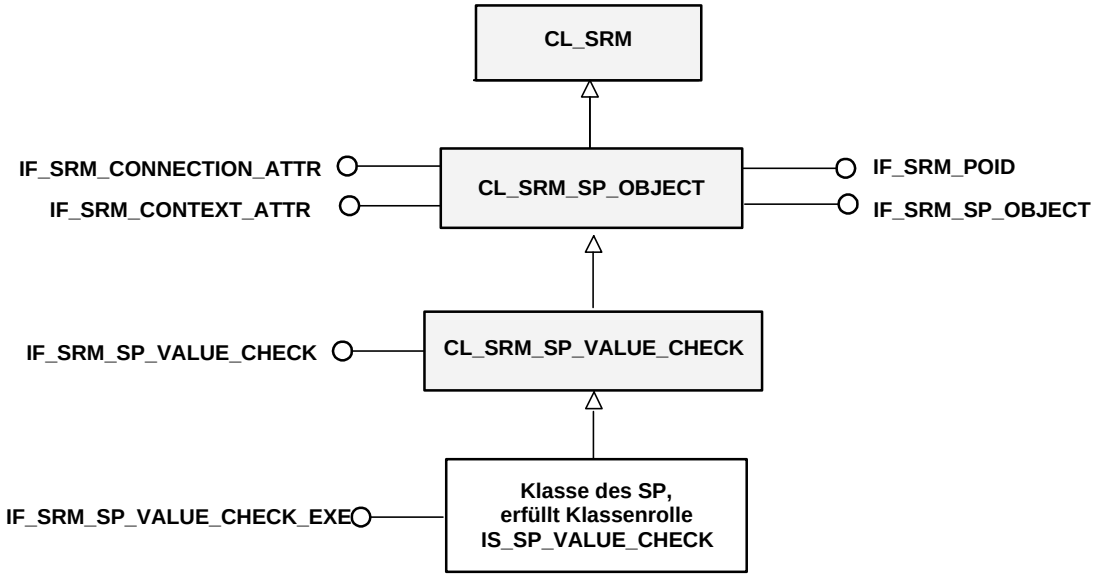



**Figure 2** Inheritance Hierarchy of the Basis Class for the Value Check

The interface IF_SRM_SP_VALUE_CHECK provides methods for reading and writing parameters, which influence the value check. The server SP must read and process these values. The client SP must set the values.

| Method | Description |
|---|---|
| GET_ATTRIBUTES | Get: List of all attributes with value check |
| SET_LIMIT_INSTANCE | Set: Selection of possible values restricted to instance. If IF_SRM=>TRUE, the server must read the SP POID parameters and restrict the possible value selection to these values. Can be set by client: Default = IF_SRM=>FALSE |
| GET_LIMIT_INSTANCE | See: SET_LIMIT_INSTANCE |
| SET_LIMIT_SPS | Set: Selection of possible values restricted to SPS. . If IF_SRM=>TRUE, the server must read the connection parameter values and restrict the possible value selection to these values. Can be set by client: Default = IF_SRM=>FALSE |
| GET_LIMIT_SPS | See: SET_LIMIT_SPS |
| SET_VALUES | Set: List of attribute values for which the check should be executed. Must be set by the client. |
| GET_VALUES | See: SET_VALUES |
| SET_UPDATE_VALUE_DESCRIPTION | Set: If IF_SRM=>TRUE, the server must update the description of the values for all attribute value objects (SET_ / GET_VALUES). Can be set by client: Default = IF_SRM=>FALSE |
| GET_UPDATE_VALUE_DESCRIPTION | See: SET_UPDATE_VALUE_DESCRIPTION |

### 7.2.1 Server Integration

The following steps are required for server integration of the value check:

1) When publishing the POID parameter, connection parameter, or context parameter for which you want to offer a value check, enter t he value IF_SRM=>TRUE in the general attribute

description for the field IS_CHECK.

2) Implement the class role of the input help IS_SP_VALUE_CHECK. To implement the class roles, create a class that inherits from the basis class CL_SRM_SP_VALUE_CHECK, and register this class in the registry for your SP.

3) Implement the interface IF_SRM_SP_VALUE_CHECK_EXE for the class you have just created. This interface only has one method: IF_SRM_SP_VALUE_CHECK_EXE~EXECUTE_CHECK. You must implement this method.

**IF_SRM_SP_VALUE_CHECK_EXE~EXECUTE_CHECK( )**

| RETURNING | re_check_result | TYPE | srmboolean |
|---|---|---|---|

This method executes the value check. The following steps must be executed:

- Use the method IF_SRM_SP_VALUE_CHECK ->GET_VALUES to get the value (or values) entered by the user. The returning parameter RE_ATTR_VALUES contains a list of attribute value objects
- Read the value (or values) from the attribute value object
- Execute the value check
- Set the returning parameter RE_CHECK_RESULT (flag whether or not the value exists)

**Example Code with Commentary**

---

METHOD if_srm_sp_value_check_exe~execute_check .

```
  DATA: lo_valuecheck TYPE REF TO if_srm_sp_value_check,
        lt_value      TYPE srm_list_attribute_value,
        lo_attrvalue  TYPE REF TO if_srm_attribute_value,
        lt_attrvalue  TYPE srm_list_string,
        lwa_attrvalue TYPE LINE OF srm_list_string,
        l_attrvalue   TYPE string,
        lwa_srmspsde  TYPE srmspsde.

** Get the object for the value check
  lo_valuecheck ?= me.

** Request the table with the attribute value object
  lt_value = lo_valuecheck->get_values( ).

** Read the value
   READ TABLE lt_value INTO lo_attrvalue INDEX 1.
    lt_attrvalue = lo_attrvalue->get_string_value( ).

   LOOP AT lt_attrvalue INTO lwa_attrvalue.
      l_attrvalue = lwa_attrvalue-value.
   ENDLOOP.

** -> Carry out a value check and set the indicator (returning parameter)
   ... .

ENDMETHOD. "IF_SRM_SP_VALUE_CHECK_EXE~EXECUTE_CHECK
```

---

### 7.2.2 Client integration

For client integration of the value check, proceed as follows:

Get a service object and use it to call the method IF_SRM_SRM_SERVICE ->GET_VALUE_CHECK. As the import parameter, enter the POID object of the server SP that executes the value check. As a returning parameter, you receive an interface reference to IF_SRM_SRM_VALUE_CHECK. This interface has the methods described below:

| Method | Description |
|---|---|
| GET_SETTINGS | Returns a reference to the interface IF_SRM_SP_VALUE_CHECK, described above. |
| EXECUTE_CHECK | Executes the value check |

## 7.3 Logging

The framework provides a service that you can use to log all the activities performed by a user on elements and subelements of the service provider. The logging function uses the classification parameters LOG_LEVEL and LOG_KEEP_DAYS.

The parameter LOG_LEVEL specifies the level of detail of the logging. When you create element types, you can assign the classification parameter LOG_LEVEL the values 1-4 (the standard setting is 3). The higher the value, the more activities are logged. When you implement the log function in your service provider, assign a log level to every activity that is to be logged. The log is only written if the log level assigned by the user is greater than or equal to the log level of the activity. If you want to ensure that a particular activity is always logged, assign a lower log level to this activity in the program.

The parameter LOG_KEEP_DAYS determines for how long the log entries are kept. You can enter a number of days as a value and assign the number of days to an element type (the standard setting is 365 days).

To use the logging service in your service provider, proceed as follows:

### 7.3.1 Determining Which Activities are Logged

Implement a class that inherits from the basis class CL_SRM_SP_PROTOCOL, and register this in the registry maintenance in your service provider. This fulfills the class role IS_SRM_PROTOCOL_HANDLER.

The class CL_SRM_SP_PROTOCOL implements the interface IF_SRM_PROTOCOL_META. This interface groups together methods for describing the activities to be logged. If necessary, you can redefine these methods:

**IF_SRM_PROTOCOL_META~GET_STANDARD_ACT_LIST( )**

| | | | |
|---|---|---|---|
| EXPORTING | EX_STANDARD_LIST | TYPE | SRMPT_PROTO_ACT_DESC_TAB |

This method returns a list of standard activities that are logged with a particular log level, display text, and DB update mode. If you want to log the standard activities using a different log level, you should redefine this method.

**IF_SRM_PROTOCOL_META~GET_SPECIAL_ACT_LIST( )**

| | | | |
|---|---|---|---|
| EXPORTING | EX_SPECIAL_LIST | TYPE | SRMPT_PROTO_ACT_DESC_TAB |

If you do not redefine this method, it remains empty. You redefine this method if you w ant to log service provider-specific activities. This method also returns a list of activities that are logged with a particular log level, display text, and DB update mode.

Note: The display text of the activity is determined using an OTR alias. To maint ain the OTR alias, open the transaction sotr_edit. Here you enter an ALIAS, the text, and the package. The OTR alias is composed as follows: < Package/technical abbreviation for the activity>.

### 7.3.2 Creating a Log Entry

Implement a method using your service pro vider client class. You call this method when want to write a log entry.

Example: To log the activities that are offered to the user in the context menu, call this method within the method IF_SRM_SP_CLIENT_WIN~MY_ACTION (or for outplace activities, IF_SRM_SP_CLIENT_OUTPLACE~START_APPLICATION).

For writing the log entry, use the method IF_SRM_PROTOCOL_ENTRY~WRITE that you have inherited from the class CL_SRM_SP_PROTOCOL.

**IF_SRM_PROTOCOL_ENTRY->WRITE( )**

| | | | |
|---|---|---|---|
| IMPORTING | IM_ACT_ID | TYPE | STRING |
| IMPORTING | IM_SUBOBJ_ID | TYPE | SRMPOIDID |
| IMPORTING | IM_ARG1 | TYPE | STRING |
| IMPORTING | IM_ARG2 | TYPE | STRING |
| IMPORTING | IM_ARG_STRING | TYPE | STRING |

This method enters a log entry for a POID. The values for the import parameters are displayed in the appropriate columns in the log. For th e import parameter IM_ACT_ID, enter the ID of the current activity (the information is displayed in the *Activity* column). For the import parameter IM_SUBOBJ_ID, enter the POID Directory ID of the element for which the logged activity will be executed (the information is displayed in the log in the *Object ID* column). The parameters IM_ARG1 and IM_ARG2 refer to a value that has been changed by the user (for example, an attribute value). For the parameter IM _ARG1, enter the old value (displayed in the log in t he column *Value 1*), for the parameter IM_ARG2, enter the new value (displayed in the log in the column *Value 2*). For the parameter IM_ARG_STRING, you can enter any value (displayed in the log in the *Comment* column).

We recommend that you buffer the object of type IF_SRM_PROTCOL_ENTRY as an attribute for your service provider client class, so that you do not need to request it from the factory object for every call.

**Example Code with Commentary**

```
DATA: root        TYPE REF TO if_srm_root,
      factory     TYPE REF TO if_srm_sps_factory,
      o_proto     TYPE REF TO if_srm_sp_protocol_entry,
      o_poid      TYPE REF TO if_srm_poid,
      l_pdir_id_c TYPE srmpoidid,
      l_pdir_id_s TYPE string.

** Get own POID object
  TRY.
      o_poid ?= me->if_srm_sp_object~get_poid( ).
```

```
** Get object of type IF_SRM_PROTOCOL_ENTRY
      IF me->my_proto_handler IS INITIAL.
        root = me->if_srm~get_root( ).
        factory = root->get_sps_factory_by_poid( o_poid ).
        o_proto ?= factory->connect_object( im_class_role =
                            'IS_SP_PROTOCOL_HANDLER' im_poid = o_poid ).
        me->my_proto_handler = o_proto.
      ENDIF.

** Determine current POID Directory ID
      IF NOT o_poid IS INITIAL.
        l_pdir_id_s = o_poid->get_poid_directory_id( ).
        l_pdir_id_c = l_pdir_id_s.
      ENDIF.

** Write log entry
      IF me->my_proto_handler IS NOT INITIAL.
        CALL METHOD me->my_proto_handler->write
          EXPORTING
            im_act_id     = im_activity
*            im_arg1       = im_arg1
*            im_arg2       = im_arg2
*            im_arg_string = im_arg_str.
            im_subobj_id   = l_pdir_id_c.
*         commit work.
      ENDIF.
```

### 7.3.3 Display Log

Implement a method using your service provider client class that displays the log. You call this method if the user selects the function for displaying the log. (You need to make this function available within the service provider, either as a separate activity or as a pushbutton.)

To display the log entry, you can use the function module SRM_PROTOCOL_POPUP . This function module internally calls the method IF_SRM_PROTOCOL_VIEWER~DISPLAY, which you have inherited from the class CL_SRM_SP_PROTOCOL. You therefore need to enter an object of type IF_SRM_PROTOCOL_VIEWER in the function module.

**Example Code**

```
DATA: o_proto_viewer TYPE REF TO if_srm_sp_protocol_viewer.


  o_proto_viewer ?= me->my_proto_handler.

  CALL FUNCTION 'SRM_PROTOCOL_POPUP'
    EXPORTIgNG
      protocol_viewer = o_proto_viewer
    EXCEPTIONS
      internal_error  = 1
      OTHERS          = 2.
```