SAP®

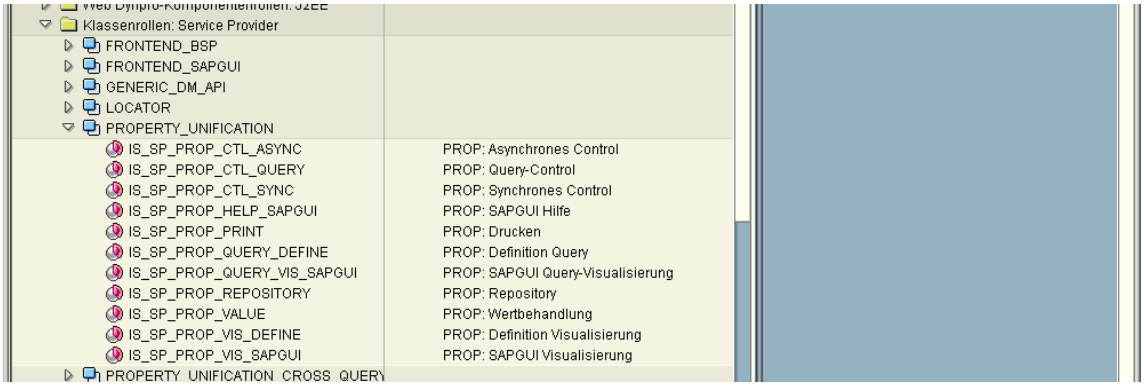
Writing a “connector” to Property Unification – How-to guide

Contents

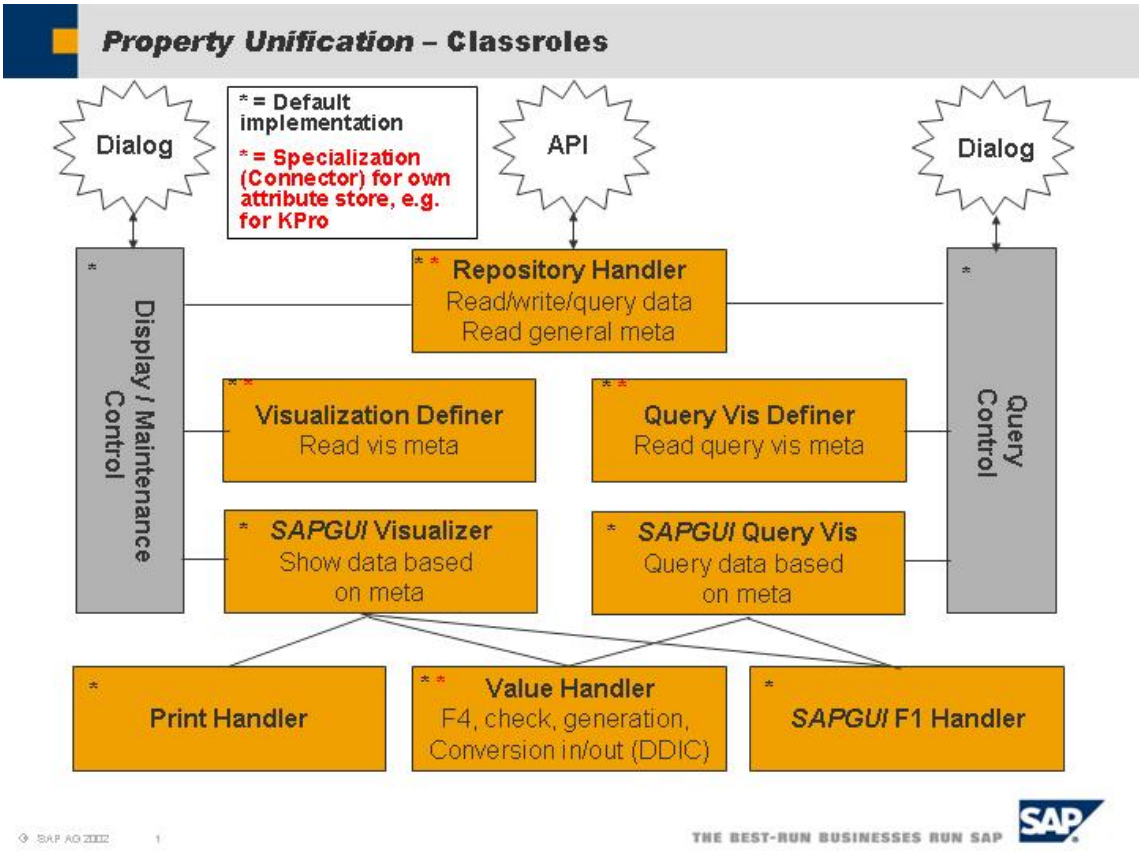
Introduction

The Property Unification (short PU) defines a set of generic (SP independent) classroles and interfaces for attribute data and metadata access. PU forces a clean separation between backend, frontend and controller logic and PU forces a clean separation between data and metadata handling. Each SP is able to connect to PU by implementing mandatory PU classroles (interfaces) or by re-using the default implementations.

PU consists of the following classroles…



Basis RCM delivers for all PU classroles default implementations. These default implementations can be re-used by a SP without modification as long as this fits to SP needs. But sometimes there’s a need for specialization or re-writing a classrole implementation from scratch, e.g. when a own attribute repository has to be connected or when a special value help/check (which can not be realized via DDIC) is needed.



Normally a SP, which wants to use PU and which comes with an own attribute repository, has to implement 3 to 4 PU classroles: IS_SP_PROP_REPOSITORY, IS_SP_PROP_VIS_DEFINE, IS_SP_PROP_QUERY_DEFINE and (depending on the need for special value help/check) IS_SP_PROP_VALUE. All other classroles (IS_SP_PROP_CTL_ASYNC etc.) are normally re-used without modification: the default implementations a very “generic”.

On the other hand: SPs, which want to use PU and which don’t have to support an own attribute repository (means that they can re-use our default repository), can re-use all PU default implementations without modification, they only have to do some customizing regarding the default repository (please compare to document “PU_HOWTO_using_default_repository.doc”).

This document focuses on the first type of SPs, which have to support an own attribute repository. These SP need a PU connector, means the implementation for the classroles IS_SP_PROP_REPOSITORY, IS_SP_PROP_VIS_DEFINE and IS_SP_PROP_QUERY_DEFINE.

Basis RCM delivers PU connectors for default repository and for KPro attributes (via GSP). These connectors are located in the packages SRM_PROPERTY and SRM_GSP_PROPERTY.

The PU connector for the default repository is the “reference implementation”. Feel free to look at the existing classrole implementations in package SRM_PROPERTY!

Attribute value and description objects

Before implementing PU classroles, you should be (at least a little bit) familiar with the framework built-in attribute value and description object concept, since all attribute data and metadata exchange is done via these runtime objects.

The attribute value and description objects are (of course) runtime objects. The value object holds the current data (single and multi values are possible) and the description object the current metadata of a single attribute. The description object is divided in a general description part, a visualization description part (-> maintenance view), a query visualization description part (-> query view) and a list visualization description part (-> (hit) list view).

The work for setting up a attribute value and attribute description object (for one attribute ID) could be like following: Creating attribute value and description object via framework factory and setting a link from value object to description object…

```
    data: srm_object_factory type ref to if_srm_srm_object_factory,
          edit_attr_value type ref to if_srm_edit_attribute_value,
          edit_attr_description type ref to IF_SRM_EDIT_ATTRIBUTE_DESC,
          attr_description type ref to IF_SRM_ATTRIBUTE_DESC,
          general_description type SRMADGEN,
          tabfield_description type SRMADTAFIN,
          visual_description type SRMADVIS,
          query_description type SRMADQUE,
          vis_in_list_description type SRMADVLS,
          string_tab type SRM_LIST_STRING,
          string_wa type line of SRM_LIST_STRING.

    srm_object_factory = me->if_srm~GET_SRM_OBJECT_FACTORY( ).


**********************************************************************
*** setting up attribute DESCRIPTION for one attribute ID…
**********************************************************************

    edit_attr_description ?= srm_object_factory->CREATE_ATTR_DESC_PROPERTY( ).

    general_description-id = 'FOO'. "Constant would be better…
    general_description-alias_id = 'SRM_FOO'. "Optional: Alias ID is important for
       "cross SP search, please have a look at the document
       "PU_HOWTO_cross_SP_search.doc" for futher information…
    general_description-repos_id = ''. "Optional: Could be helpful for connector
       "itself, if connector has to serve more than one repository…
    general_description-type = IF_SRM_ATTRIBUTE_DESC=>TABFIELD. "DDIC reference,
       "normally the best choice!
    general_description-is_list = IF_SRM=>FALSE. "single or multi valued…
    general_description-… = …
    edit_attr_description->SET_GENERAL_DESCRIPTION( general_description ).

*   Optional: Only to fill, if general_description-type = TABFIELD…
    tabfield_description-tabname = 'FOO'. "Table or structure name…
    tabfield_description-fieldname = 'BAR'.
    edit_attr_description->SET_TABFIELD_DESCRIPTION( tabfield_description ).

*   Optional: Only to fill, if general_description-IS_VISIBLE = TRUE…
*    visual_description-group_no = . NOT SUPPORTED AT THE MOMENT!
*    visual_description-group_label = . NOT SUPPORTED AT THE MOMENT!
    visual_description-row_no = 1.
    visual_description-column_no = 1.
    visual_description-… = …
    edit_attr_description->SET_VISUAL_DESCRIPTION( visual_description ).

*   Optional: Only to fill, if general_description-IS_QUERY = TRUE…
*    query_description-group_no = . NOT SUPPORTED AT THE MOMENT!
```

```
*    query_description-group_label = . NOT SUPPORTED AT THE MOMENT!
     query_description-row_no = 1.
     query_description-column_no = 1.
     query_description-… = …
     edit_attr_description->SET_QUERY_DESCRIPTION( query_description ).

*    Optional: Only to fill, if general_description-IS_VIS_IN_LIST = TRUE…
     vis_in_list_description-column_no = 1.
     vis_in_list_description-emphasized = if_srm=>false.
     edit_attr_description->SET_VIS_IN_LIST_DESCRIPTION( vis_in_list_description ).


**********************************************************************
*** setting up attribute VALUE for one attribute ID…
**********************************************************************

     edit_attr_value ?= srm_object_factory->CREATE_ATTR_VALUE( ).

*    >>>>> Store link from value to description object… <<<<<
     attr_description ?= edit_attr_description.
     edit_attr_value->set_description( attr_description ).

*    Here single valued, since general_description-is_list = FALSE…
     string_wa-id = '1'. "not so important in this case, has to serve uniqueness…
     string_wa-value = 'FOO_IN'. "mandatory: internal: after input conversion…
     string_wa-value_out = 'FOO_OUT'. "optional: external: after output conversion…
     string_wa-text = 'This is my FOO longtext. "-> longtext in maintenance view…
     insert string_wa into table string_tab.
     edit_attr_value->set_ddic_string( string_tab ).
```

There're analogous interfaces (IF_SRM_ATTRIBUTE_DESC, IF_SRM_ATTRIBUTE_VALUE) to read out attribute value and description objects.

The attribute value and description objects are nomally stored in internal tables of type SRM_LIST_ATTRIBUTE_VALUE and SRM_LIST_ATTRIBUTE_DESC.

All description structures (like SRMADGEN) have a field called USER_OBJECTS: This may be used for tunneling SP specific metadata through PU interfaces. E.g. this is used to attach metadata IS_LOIO at attributes which are stored in KPro (-> KPro PU connector).

PU context object

Another point to know in advance is the PU context object. Many PU classrole implementations rely on a context object, especially the control implementations rely on them.

The PU context object bundles the needed "additional" information, e.g. in a context object is specified, where a control should be started: inplace or outplace.

The caller of a PU service (here synchronous control) creates and fills the context…

```
     myPropertyService = myService->get_property_service( ).

*    Create context object...
     myPropertyContext = myPropertyService->get_context( ).

*    Fill context object...
     myPropertyContext->if_srm_prop_context_vis~ui_set(
           if_srm_prop_context_vis=>ui_sapgui ).
```

```
myPropertyContext->if_srm_prop_context_vis~mode_set(
      if_srm_prop_context_vis=>mode_create ).
myPropertyContext->if_srm_prop_context_vis~place_set(
      if_srm_prop_context_vis=>place_out ).

* Call control...
myCancelled = mySynchronousPropertyControl->execute( myPropertyContext ).
```

The implementer of a PU classrole consumes (reads out) the context…

```
case context->if_srm_prop_context_vis~mode_get( ).
 when if_srm_prop_context_vis=>mode_modify
   or if_srm_prop_context_vis=>mode_create.
  …
 when if_srm_prop_context_vis=>mode_display
  …
 when others.
endcase.
```

The context object itself (class CL_SRM_PROP_CONTEXT) holds the context data in attributes and implements at lot of getter and setter methods. The setter and getter methods are divided into interfaces of “purpose”: most methods are needed for maintenance visualization and query visualization. The leading interface IF_SRM_PROP_CONTEXT includes the following subsequent interfaces…

- IF_SRM_PROP_CONTEXT_CROSS_QY (For PU cross SP search service)
- IF_SRM_PROP_CONTEXT_PRINT (For PU printing service)
- IF_SRM_PROP_CONTEXT_QUERY (For PU query visualization)
- IF_SRM_PROP_CONTEXT_REG (very special for record number generator!)
- IF_SRM_PROP_CONTEXT_VIS (For PU visualization)

The main interfaces IF_SRM_PROP_CONTEXT and IF_SRM_PROP_CONTEXT_VIS have a setter and a getter method for USER_OBJECTS: This may be used for tunneling SP specific metadata through PU interfaces.

You’ll find appropriate constants at context interfaces for each purpose, e.g…

```
if_srm_prop_context_vis=>mode_modify
if_srm_prop_context_vis=>mode_create
if_srm_prop_context_vis=>mode_display
if_srm_prop_context_vis=>mode_query
```

Implementing PU classrole IS_SP_PROP_REPOSITORY

The “repository” classrole serves the connection to the attribute data and metadata store. It realizes read, write and search access to attribute data and it realizes read access to attribute metadata.

The “repository” classrole consists of the leading interface IF_SRM_SP_PROP_REPOSITORY and the following included subsequent interfaces…

- IF_SRM_SP_PROP_REPOS_DATA (Access to attribute data)
- IF_SRM_SP_PROP_REPOS_META (Access to attribute metadata)

- IF_SRM_SP_PROP_REPOS_KEY (>= 7.00, special key handling)
- IF_SRM_SP_PROP_REPOS_CONN (>= 7.00, “declaring PU participation”)

Be aware that there’re aliases defined for most included interface methods at IF_SRM_SP_PROP_REPOSITORY: e.g. EXISTENCE_CHECK for IF_SRM_SP_PROP_REPOS_DATA~EXISTENCE_CHECK.

The “repository” classrole is called from PU controls for maintenance and query and it can be called directly in a dark way. Please have a look at howto reports SRM_PROP_UNIFIC_CONTROL_HOWTO and SRM_PROP_UNIFIC_DARK_HOWTO.

The sequence of calls to repository classrole interface methods is normally like following, but you shouldn’t rely and program on this assumption (!): a *SET might follow a *GET or another *GET or vice-versa. Ok, a LOCK should be ahead of a UNLOCK and a QUERY ahead of a QUERY_RESULT_DETAIL_GET. Please have a look at howto report SRM_PROP_UNIFIC_DARK_HOWTO or at “reference” implementation of control classrole, e.g. at implementation of synchronous control CL_SRM_SP_PROP_CTL_SYNC->IF_SRM_SP_PROP_CTL_SYNC~EXECUTE, to see some call sequences. Here’re 2 examples…

- Writing of attribute data: methods called directly or from PU control (e.g. synchronous)…

```
repository->lock( ).
repository->*get( … ). “maybe omitted for direct call scenario…
<…PU control: show the maintenance view…>
repository->*set( … ).
repository->unlock( ).
```

- Search on attribute values: methods called directly or from PU query control

```
repository->query( ).
repository->query_result_detail_get( … ).
```

Take in mind for implementation of the following interface methods, that you’re implementing classrole interfaces, which have a built-in access to a current poid (me->if_srm_sp_object~get_poid( )). Therefore you’ve always access to a complete key (instance POID) or to a part of a key (model POID) and therefore there’s no further key specification in signature needed.

Feel free to look at the existing classrole “reference” implementations in package SRM_PROPERTY. The “reference” class for repository classrole is CL_SRM_SP_PROP_REPOSITORY.

Implementing the leading interface IF_SRM_SP_PROP_REPOSITORY…

- ~GET:
  This method gets a context object and has to return data and metadata for the complete attribute set, that is assigned to current instance (!) POID.

  The main work is normally very easy to do, since it can be done via delegation of calls to subsequent interfaces…

```
property_tab = me->IF_SRM_SP_PROP_REPOS_META~get( context ).
me->IF_SRM_SP_PROP_REPOS_DATA~get( property_tab = property_tab
       context = context ).
```

```
<…do some additional work…>
```

In addition to the delegation calls you might (?) have to do some other work too.

The result is a internal table PROPERTY_TAB, which is a list of attribute value objects and with this a list of attribute description objects too (each attribute value object has a link to a description object, please compare to chapter “Attribute value and description objects”).

- ~LOCK:
  This method gets a context object and a boolean flag, if a (already locked or failed lock) popup should be shown. It has to realize the lock for the write access to the set of attributes, that is assigned to current instance (!) POID.

  This is normally done by calling a enqueue function.

  This is an optional method, since the caller (direct or via PU control) has to decide (context->if_srm_prop_context_vis~do_lock_set/get), if there’s a separate lock needed for attribute write access (maybe existing lock for entity itself, e.g. document content, is enough).

- ~UNLOCK:
  This method has to realize the release of the lock for the set of attributes, that is assigned to current instance (!) POID.

  This is normally done by calling a enqueue function.

  This is an optional method, since the caller (direct or via PU control) has to decide (context->if_srm_prop_context_vis~do_lock_set/get), if there’s a separate lock needed for attribute write access (maybe existing lock for entity itself, e.g. document content, is enough).

- ~QUERY:
  This method gets a query expression table, a context object and a (optional) result description table.

  The query method has to realize the search regarding the query expression table and the context object, and it has to assemble the search result in a format based on a result description table. The query method is – of course – called at a model POID.

  The query expression table has a format that is derived from SAPs select options (ranges).
  The context object includes some information, which is important before starting the query, e.g. the following 2 infos…

```
context->IF_SRM_PROP_CONTEXT_QUERY~CASE_SENSITIVE_GET( )
context->IF_SRM_PROP_CONTEXT_QUERY~MAX_HITS_GET( )
```

  You should throw an exception, if you’re not able to support a special query context, e.g. like following (BTW: the PU query control doesn’t support fuzzy search too)…

```
if context->IF_SRM_PROP_CONTEXT_QUERY~FUZZY_SEARCH_GET( )
       = if_srm=>true.
       RAISE EXCEPTION TYPE cx_srm_sp_prop_repository
              EXPORTING TEXTID =
                cx_srm_sp_prop_repository=>ET_NOT_SUPPORTED_CONTEXT.
endif.
```

If the optional result description tab (table of attribute description objects) is empty, you've to call the PU visualization define classrole, which delivers the visualization metadata, in this case the hitlist visualization metadata…

```
myPropertyDescTab = me->IF_SRM_SP_PROP_REPOS_META~raw_get(
  filter = IF_SRM_SRM_PROP_REPOS_META=>FILTER_VISIBLE_IN_LIST
  context = context ).
myPropertyVisListDefine ?= myPropService->GET_VIS_DEFINE( myPoid ).
myPropertyVisListDefine->get(
  exporting context = context  do_column_sort = if_srm=>true
  changing property_desc_tab = myPropertyDescTab ).
```

After successful search with given query expression tab (select option tab), you've to build up the result structure, which consists of the actual result description table (imported or build up like described before) and the result lines. The result lines are (due to performance reasons!) not instance POIDs, they're attribute values, which can be shown in a hitlist. The corresponding method ~QUERY_RESULT_DETAIL_GET returns the instance POID for exactly one hit in the hitlist.

- ~QUERY_RESULT_DETAIL_GET:
  This method gets a query ID and a context object and has to deliver the query detail (the instance POID) for the given ID.

  This method is – of course – called after a call of ~QUERY. It has to be able to deal with the query ID, which is the key of the query result table. This ID might be a POID directory ID or something else.

Implementing the interface IF_SRM_SP_PROP_REPOS_DATA…

- ~DELETE:
  This method gets a context object and has to delete the data for the complete attribute set, that is assigned to current instance (!) POID.

- ~EXISTENCE_CHECK:
  This method has to check, if any data exists for the attribute set, that is assigned to current instance (!) POID.

- ~GET:
  This method gets a table of attribute value objects and a context object. The list of attribute value objects might be the complete set of attributes, that is assigned to current instance (!) POID, but it doesn't have to, it could be a subset.

  The list of attribute value objects has empty value objects, but – of course – filled description objects.

The GET method has to read the attribute data from its repository and has to fill up the list of empty value objects. The fill-up is normally done via "EDIT_ATTRIBUTE_VALUE->SET_DDIC_STRING(STRING_TAB)." for each value object.

- ~SET:
  This method gets a table of attribute value objects and a context object. The list of attribute value objects might be the complete set of attributes, that is assigned to current instance (!) POID, but it doesn't have to, it could be a subset.

  The list of attribute value objects has filled value and description objects.

  The SET method has to read the data from the attribute value objects (normally via "STRING_TAB = ATTRIBUTE_VALUE->GET_DDIC_STRING( )."") and has to write the attribute data to its repository, this means inserts or updates of values.

Implementing the interface IF_SRM_SP_PROP_REPOS_META…

- ~GET:
  This method gets a context object and has to return the metadata for the complete attribute set, that is assigned to current model or instance POID.

  The metadata has to be returned as table of attribute value objects, where – of course - the value objects are empty and the description objects are filled.

- ~RAW_GET:
  This method gets a (optional) filter and a context object and has to return the metadata for the complete attribute set, that is assigned to current model or instance POID.

  The (optional) filter can have the following constant values…
  - IF_SRM_SRM_PROP_REPOS_META=>FILTER_VISIBLE
  - IF_SRM_SRM_PROP_REPOS_META=>FILTER_VISIBLE_IN_LIST
  - IF_SRM_SRM_PROP_REPOS_META=>FILTER_QUERY

  …which means, that if FILTER_VISIBLE_IN_LIST is specified, this method should only return attributes, which are (potentially) shown in hitlist.

  The metadata has to be returned as table of attribute description objects.

- ~SINGLE_GET:
  This method gets a attribute ID and a context object and has to return the metadata for one attribute, that is assigned to current model or instance POID.

  The metadata has to be returned as attribute value object, where – of course - the value object itself is empty and the description object is filled.

Implementing the interface IF_SRM_SP_PROP_REPOS_KEY…

Implementation for this interface is only needed, if you want to re-use the default repository. Please have a look at document "PU_HOWTO_using_default_repository.doc", section "Coding", section "Step 1 (optional): Working with an own attribute set key (instead of POID-Dir-ID)".

Implementing the interface IF_SRM_SP_PROP_REPOS_CONN…

- ~SPS_PARTICIPATION:
  Since there're default implementations for all PU classroles, a "higher" caller of PU services (e.g. clientframework or organizer) cannot find out, if a SP or elementtype really (actively) supports PU.

  Therefore a SP has to declare the participation to PU via this method. A SP might do this for all elementtypes (SPS) or only for selected ones.

Implementing PU classrole IS_SP_PROP_VIS_DEFINE

The "visualization define" classrole serves the connection to the attribute metadata (here metadata for maintenance and (hit-)list view) store. It realizes read access to attribute visualization metadata (e.g. row number).

The "visualization define" classrole consists of the interface IF_SRM_SP_PROP_VIS_DEFINE (for maintenance view) and the interface IF_SRM_SP_PROP_VIS_LIST_DEF (for (hit-)list view).

The "visualization define" classrole is called from PU controls for maintenance and query and it can be called directly in a dark way (but the second is not very realistic). Please have a look at howto reports SRM_PROP_UNIFIC_CONTROL_HOWTO and SRM_PROP_UNIFIC_DARK_HOWTO.

Take in mind for implementation of the following interface methods, that you're implementing classrole interfaces, which have a built-in access to a current poid (me->if_srm_sp_object~get_poid( )). Therefore you've always access to a complete key (instance POID) or to a part of a key (model POID) and therefore there's no further key specification in signature needed.

Feel free to look at the existing classrole "reference" implementations in package SRM_PROPERTY. The "reference" class for repository classrole is CL_SRM_SP_PROP_VIS_DEFINE.

Implementing the interface IF_SRM_SP_PROP_VIS_DEFINE…

- ~GET:
  This method gets a table of attribute value objects and a context object and has to add the visualization metadata (-> maintenance view), that is assigned to current model or instance POID.

  The metadata has to be added to the table of attribute value objects, means to the description objects, which are linked to the value objects. This is done with "EDIT_ATTRIBUTE_DESCRIPTION->SET_VISUAL_DESCRIPTION(VIS_DESCRIPTION)."

Implementing the interface IF_SRM_SP_PROP_VIS_LIST_DEF…

- ~GET:
  This method gets a table of attribute value objects and a context object and has to add the visualization metadata (-> (hit-)list view), that is assigned to current model or instance POID.

  The metadata has to be added to the table of attribute value objects, means to the description objects, which are linked to the value objects. This is done with "EDIT_ATTRIBUTE_DESCRIPTION->SET_VIS_IN_LIST_DESCRIPTION(VIS_IN_LIST_DESCRIPTION)."

Implementing PU classrole IS_SP_PROP_QUERY_DEFINE

The "query define" classrole serves the connection to the attribute metadata (here metadata for query view) store. It realizes read access to attribute query visualization metadata (e.g. row number).

The "query define" classrole consists of the interface IF_SRM_SP_PROP_VIS_QUERY.

The "query define" classrole is called from PU query control and it can be called directly in a dark way (but the second is not very realistic). Please have a look at howto reports SRM_PROP_UNIFIC_CONTROL_HOWTO, SRM_PROP_UNIFIC_DARK_HOWTO and SRM_PROP_UNIFIC_QUERY_HOWTO.

Take in mind for implementation of the following interface methods, that you're implementing classrole interfaces, which have a built-in access to a current poid (me->if_srm_sp_object~get_poid( )). Therefore you've always access to a complete key (instance POID) or to a part of a key (model POID) and therefore there's no further key specification in signature needed.

Feel free to look at the existing classrole "reference" implementations in package SRM_PROPERTY. The "reference" class for repository classrole is CL_SRM_SP_PROP_QUERY_DEFINE.

Implementing the interface IF_SRM_SP_PROP_QUERY_DEFINE…

- ~GET:
  This method gets a table of attribute value objects and a context object and has to add the visualization metadata (-> query view), that is assigned to current model POID.

  The metadata has to be added to the table of attribute value objects, means to the description objects, which are linked to the value objects. This is done with "EDIT_ATTRIBUTE_DESCRIPTION->SET_QUERY_DESCRIPTION(QUERY_DESCRIPTION)."

Optional: implementing PU classrole IS_SP_PROP_VALUE

The “value” classrole bundles all requests (from visual controls) for value help, value check, value generation, conversion etc. for a single attribute.

There’s a default implementation for this classrole which realizes already the main work for attributes of type DDIC: value help, value check, in/out conversion. But apart from this you might (?) want to add a very special value help, which cannot be fired via DDIC or something similar. In this case, you can do so by implementing “value” classrole.

The “value” classrole consists of the interface IF_SRM_SP_PROP_VALUE.

The “value” classrole is called from PU maintenance and query visualization (e.g. for SAPGUI).

Take in mind for implementation of the following interface methods, that you’re implementing classrole interfaces, which have a built-in access to a current poid (me->if_srm_sp_object~get_poid( )). Therefore you’ve always access to a complete key (instance POID) or to a part of a key (model POID) and therefore there’s no further key specification in signature needed.

The EXECUTE_* methods work with a table of structure (VAL_HDL)…

- IS_REQUESTED (*comes in*: marks the attribute, for which the value request was called)
- REQUESTED_VALUE_IDX (*comes in*: field index in (string) value table (only for multivalued attributes), for which the value request was called)
- RESULT_OK (*gos out:* requested or other attributes)
- RESULT_ERR_TEXT (*gos out:* requested or other attributes)
- UPDATED (*gos out*: requested or other attributes: markes changed ATTRIBUTE_VALUE)
- HANDLER_FOUND (*gos out*: requested attribute)
- ATTRIBUTE_VALUE (*comes in and gos out*)

This table and (complicate) structure is needed, since in some cases there’s a need for update of some more attributes, even when the value request is always only requested (IS_REQUESTED) for exactly one attribute. Each call to a EXECUTE_* method gets always the complete and current list of attribute values!

Feel free to look at the existing classrole “reference” implementations in package SRM_PROPERTY. The “reference” class for repository classrole is CL_SRM_SP_PROP_VALUE.

Implementing the interface IF_SRM_SP_PROP_VALUE…

- ~EXECUTE_CHECK:
  This method gets a table of structure VAL_HDL (see above) and a context object and has to check the requested attribute for consistency. After this the “result” has to be stored in the table of structure VAL_HDL (see above).

- ~EXECUTE_HELP:
  This method gets a table of structure VAL_HDL (see above) and a context object and has to serve a value help (popup or dark) for the requested attribute. After this the “result” has to be stored in the table of structure VAL_HDL (see above).

- ~EXECUTE_GENERATE:
  This method gets a table of structure VAL_HDL (see above) and a context object and has to serve a value generation (popup or dark) for the requested attribute. After this the “result” has to be stored in the table of structure VAL_HDL (see above).

- ~EXECUTE_BUTTON_CLICK:
  This method gets a button ID, a table of structure VAL_HDL (see above) and a context object and has to react on a propietary button click (button is shown before attribute field, only in maintenance view) for the requested attribute. After this the “result” has to be stored in the table of structure VAL_HDL (see above).

- ~GET_DROPDOWN_VALUES:
  This method gets a table of structure VAL_HDL (see above) and a context object and has to serve all dropdown values for the requested attribute. After this the “result” has to be stored in the table of structure VAL_HDL (see above).

- ~CONVERSION_IN:
  This method gets a OUT string, a property description object and a context object and has to deliver the IN string (output to input conversion).

- ~CONVERSION_OUT:
  This method gets a IN string, a property description object and a context object and has to deliver the OUT string (input to output conversion).

- ~CHECK_IS_INITIAL:
  This method gets a VALUE string, a property description object and a context object and has to say, if this string contains the “semantical” initial value (normally checked via “is initial”).

- ~GET_TEXT:
  This method gets a value, a property description object and a context object and has to deliver the description text (or long text) for the value.

Integrating PU services / testing functionality

After writing the PU connector the SP developer has to integrate the built-in PU services at his frontend class, otherwise he won’t benefit from PU.

In general the built-in PU services can be devided in visual services (controls) and in “dark” services for reading/writing/searching properties. This document focuses on the usage of PU controls. You may have a look at the delivered reports…

- SRM_PROP_UNIFIC_CONTROL_HOWTO
- SRM_PROP_UNIFIC_DARK_HOWTO
- SRM_PROP_UNIFIC_QUERY_HOWTO

…for a deeper understanding of offered PU services and how to call them.

The integration of visual PU services for the maintance of properties can be done outplace (means popup) or inplace in a splitter container, the SP developers has to decide.

If the SP developer chooses the outplace way, he also has to decide, if he want's to use the synchronous or the asynchronous PU maintance control: synchronous means direct save after pressing the OK button.

If the SP developer chooses the inplace way, he has to use the asynchronous PU maintance control: asynchronous means that the save of cached property values has to be triggered from outside by a separate method call.

The visual PU query control can (at the moment) only be called outplace (means popup).

The following coding examples are based on a specialized SP AL frontend class (inherits from standard SP AL frontend class). You'll find this local class (CL_HHA_SP_DOCVIEW_AL) only in BCE.

Step 1: Calling the PU query control

First of all you've to add a new activity PROP_QUERY (or something similar) to your model activity declaration in your frontend class. E.g. it could look like following…

```
method IF_SRM_SP_ACTIVITIES~GET_MODEL_ACTIVITIES.

  RE_ACTIVITIES = SUPER->IF_SRM_SP_ACTIVITIES~GET_MODEL_ACTIVITIES( ).

  RE_ACTIVITIES->add_separator( ).

  data myActivityDescription TYPE srmactta.

  myActivityDescription-id = 'PROP_QUERY'.
  myActivityDescription-text = text-005.
  myActivityDescription-changing = if_srm=>false.
  RE_ACTIVITIES->add_activity( myActivityDescription ).

endmethod.
```

Now you've to handle the new model activity PROP_QUERY in your IF_SRM_SP_CLIENT_WIN~MY_ACTION implementation…

```
method IF_SRM_SP_CLIENT_WIN~MY_ACTION.

  case im_request->get_activity( ).
    when …
    when 'PROP_QUERY'.
      me->prop_model_action( im_request ).
    when others.
      SUPER->IF_SRM_SP_CLIENT_WIN~MY_ACTION( im_request ).
  endcase.

endmethod.


method PROP_MODEL_ACTION .
  data: myPoid type ref to if_srm_poid,
        myService type ref to IF_SRM_SRM_SERVICE,
        myPropertyService type ref to IF_SRM_SRM_SERVICE_PROP,
```

```
        myPropertyContext type ref to if_srm_prop_context,
        myPropertyQueryControl type ref to IF_SRM_SRM_PROP_CTL_QUERY,
        myCancelled type srmboolean,
        myQueryResult type srm_prop_query_result_detail,
        myCx type ref to cx_srm.

  try.
    myPoid = me->if_srm_sp_object~get_poid( ).
    myService = me->if_srm~get_srm_service( ).
    myPropertyService = myService->get_property_service( ).
    myPropertyContext = myPropertyService->get_context( ).
    myPropertyQueryControl = myPropertyService->get_ctl_query( myPoid ).

    myPropertyContext->if_srm_prop_context_vis~ui_set(
       if_srm_prop_context_vis=>ui_sapgui ).
    myPropertyContext->if_srm_prop_context_vis~place_set(
       if_srm_prop_context_vis=>place_out ).
    myPropertyContext->if_srm_prop_context_query~case_sensitive_set(
       if_srm=>true ).
    myPropertyContext->if_srm_prop_context_query~current_version_only_set(
       if_srm=>true ).
    myPropertyContext->if_srm_prop_context_query~max_hits_set( 200 ).

    myPropertyQueryControl->execute( exporting context = myPropertyContext
      importing result = myQueryResult  cancelled = myCancelled ).

….

endmethod.
```

That's all. This results in a new model activity "Search via attributes" and the following search popup…

Step 2: Calling the PU maintance control outplace

First of all you've to add some new activities PROP_DISPLAY, PROP_MODIFY, … (or something similar) to your instance activity declaration in your frontend class. E.g. it could look like following…

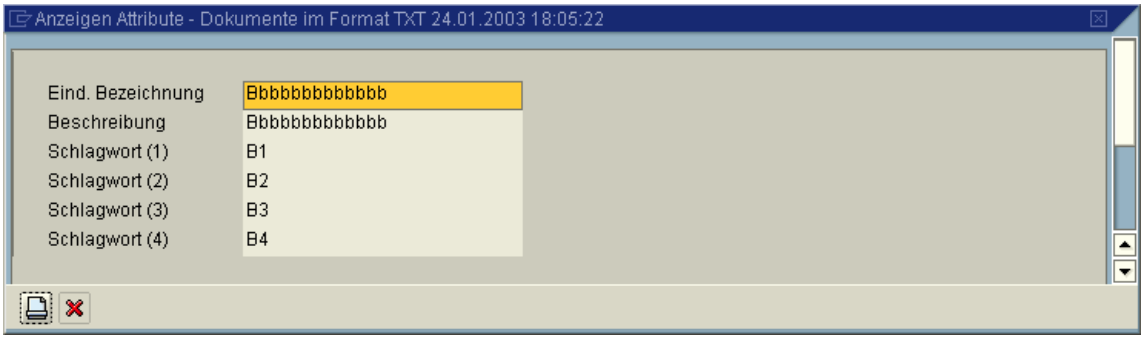
```
method IF_SRM_SP_ACTIVITIES~GET_INSTANCE_ACTIVITIES.

  RE_ACTIVITIES = SUPER->IF_SRM_SP_ACTIVITIES~GET_INSTANCE_ACTIVITIES( ).

  RE_ACTIVITIES->add_separator( ).

  data myActivityDescription TYPE srmactta.

  if me->prop_existence_check( ) = if_srm=>true.
    myActivityDescription-id = 'PROP_DISPLAY'.
    myActivityDescription-text = text-004.
    myActivityDescription-changing = if_srm=>true.
    RE_ACTIVITIES->add_activity( myActivityDescription ).
    myActivityDescription-id = 'PROP_MODIFY'.
    myActivityDescription-text = text-002.
    myActivityDescription-changing = if_srm=>true.
    RE_ACTIVITIES->add_activity( myActivityDescription ).
    myActivityDescription-id = 'PROP_DELETE'.
    myActivityDescription-text = text-003.
    myActivityDescription-changing = if_srm=>true.
    RE_ACTIVITIES->add_activity( myActivityDescription ).
  else.
    myActivityDescription-id = 'PROP_CREATE'.
    myActivityDescription-text = text-001.
    myActivityDescription-changing = if_srm=>true.
    RE_ACTIVITIES->add_activity( myActivityDescription ).
  endif.

endmethod.
```

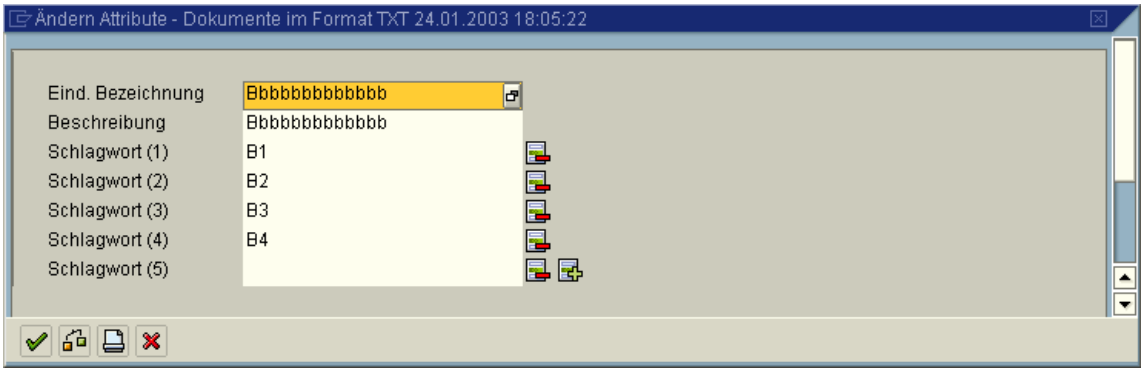
Now you've to handle the new instance activities PROP_* in your IF_SRM_SP_CLIENT_WIN~MY_ACTION implementation…

```
method IF_SRM_SP_CLIENT_WIN~MY_ACTION.

  case im_request->get_activity( ).
    when 'PROP_CREATE'
      or 'PROP_MODIFY'
      or 'PROP_DISPLAY'
      or 'PROP_DELETE'.
      me->prop_instance_action( im_request ).
    when 'PROP_QUERY'.
      me->prop_model_action( im_request ).
    when others.
      SUPER->IF_SRM_SP_CLIENT_WIN~MY_ACTION( im_request ).
  endcase.

endmethod.

method PROP_INSTANCE_ACTION.
  data: myPoid type ref to if_srm_poid,
        myService type ref to IF_SRM_SRM_SERVICE,
        myPropertyService type ref to IF_SRM_SRM_SERVICE_PROP,
        myPropertyContext type ref to if_srm_prop_context,
        mySynchronousPropertyControl type ref to IF_SRM_SRM_PROP_CTL_SYNC,
        myPropertyRepository type ref to IF_SRM_SRM_PROP_REPOSITORY,
        myCancelled type srmboolean,
        myCx type ref to cx_srm.

  try.
    myPoid = me->if_srm_sp_object~get_poid( ).
    myService = me->if_srm~get_srm_service( ).
    myPropertyService = myService->get_property_service( ).

    case im_request->get_activity( ).
      when 'PROP_CREATE'
        or 'PROP_MODIFY'
        or 'PROP_DISPLAY'.
        myPropertyContext = myPropertyService->get_context( ).
```

```
          myPropertyContext->if_srm_prop_context_vis~ui_set(
            if_srm_prop_context_vis=>ui_sapgui ).
          myPropertyContext->if_srm_prop_context_vis~place_set(
            if_srm_prop_context_vis=>place_out ).
          case im_request->get_activity( ).
            when 'PROP_CREATE'.
              myPropertyContext->if_srm_prop_context_vis~mode_set(
                if_srm_prop_context_vis=>mode_create ).
            when 'PROP_MODIFY'.
              myPropertyContext->if_srm_prop_context_vis~mode_set(
                if_srm_prop_context_vis=>mode_modify ).
            when 'PROP_DISPLAY'.
              myPropertyContext->if_srm_prop_context_vis~mode_set(
                if_srm_prop_context_vis=>mode_display ).
          endcase.
          myPropertyContext->if_srm_prop_context_vis~do_lock_set(
               if_srm=>true ).

          mySynchronousPropertyControl = myPropertyService->get_ctl_sync(
               myPoid ).
          myCancelled = mySynchronousPropertyControl->execute(
               myPropertyContext ).

          if myCancelled = if_srm=>true.
            im_request->set_activity_state(
               if_srm_request=>activity_canceled_by_user ).
            return. "!!!!!
          endif.

        when 'PROP_DELETE'.
*         TODO - of course - popup 'really delete?'...
          myPropertyRepository = myPropertyService->get_repository( myPoid ).
          myPropertyRepository->lock( show_locked_popup = if_srm=>true ).
          myPropertyRepository->IF_SRM_SRM_PROP_REPOS_DATA~delete( ).
          myPropertyRepository->unlock( ).
      endcase.

      ….
      endmethod.
```

That's all. This results in some new instance activities "Display attributes", "Modify attributes", … and the following popups…





When you want to use the asynchronous mechanismn, e.g. when you want to save properties at the same point of time when you save the content, you've to call the asynchronous control and have to call at least 2 methods later on…

```
myAsyncPropertyControl->flush( ). "writing property values to repository
myAsyncPropertyControl->release( ). "release resources: lock, cache, …
```

If you want to read out the property value cache (e.g. before it's flushed) you can do this at asynchronous control with the following method…

```
myCurrentPropertyValueTab = myAsyncPropertyControl->property_tab_get( ).
```

Step 2 (alternative): Calling the PU maintance control inplace

For inplace visualization you've to create a container and pass it to asynchronous control, the rest is nearly the same than in outplace visualization…

```
myPropertyContext->if_srm_prop_context_vis~place_set(
      if_srm_prop_context_vis=>place_in ).
myPropertyContext->if_srm_prop_context_vis~parent_cont_set(
      GLOB_PROP_CONTAINER ).

myAsyncPropertyControl->execute( myPropertyContext ).
```

If you want to be informed when a field value in property control has been changed by a user (e.g. to set a traffic light), you can register to event IF_SRM_SRM_PROP_CTL_ASYNC~DATA_CHANGED.

If you want to trigger a value check for all fields in inplace property control, you can do so with following method call…

```
myAsyncPropertyControl->inplace_check( ).
```

For a better understanding of calls to asynchronous control you may have a look at SP record frontend class CL_SRM_REC, which uses the inplace PU control.