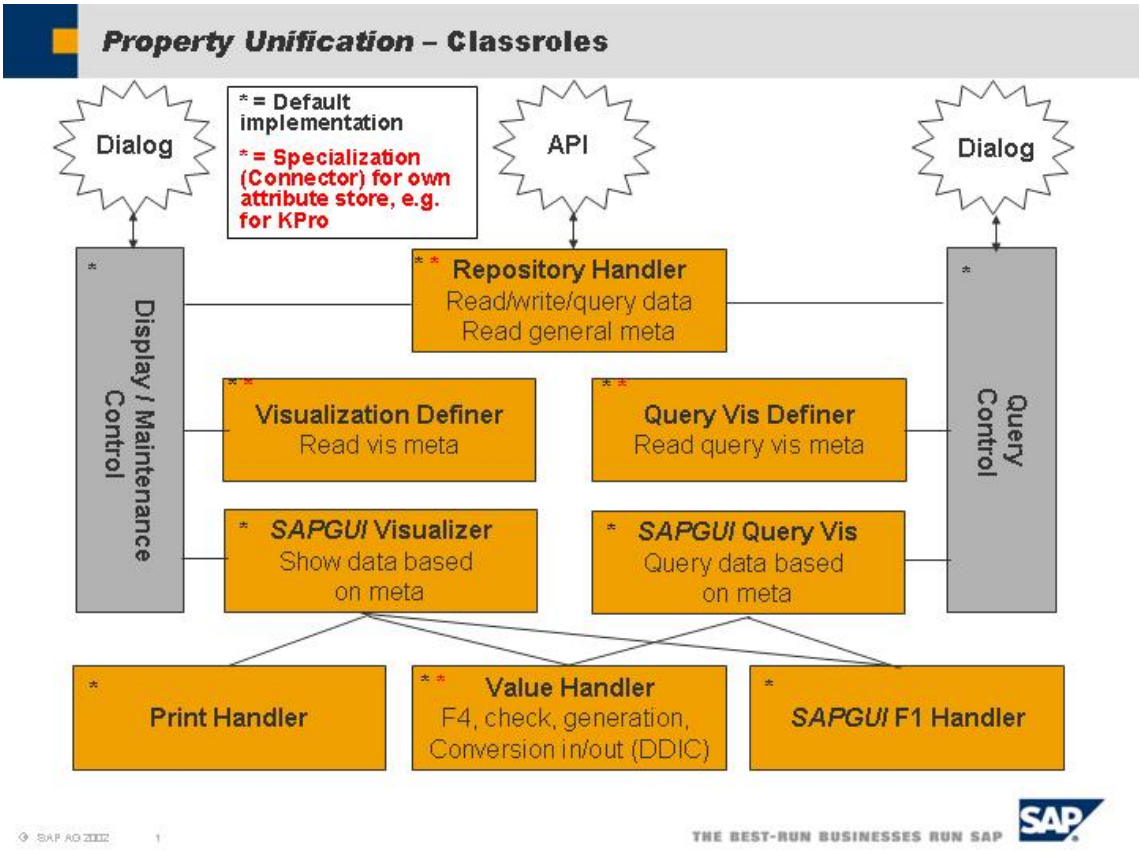
SAP®

Using Default Property Repository via PU – Step by step 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. This document focuses on last part, the re-use.



Basis RCM delivers with release WAS 7.00 (NW 2005) a new massdata enabled property repository (so-called default property repository). The new property repository comes with an own API and an own configuration/customizing.

The new property repository (the API and the customizing) is connected to PU classroles/interfaces. This PU “connector” represents the default PU implementation for classrole *repository* and can be re-used by any SP.

The following chapters describe step by step, how a SP developer can use (integrate) this new default property repository.

Covered functionality

The new property repository is able to handle massdata and is able to deal with multi language and multi value properties (depends on configuration). It has an own API, which can be used independent from PU classroles/interfaces.

The PU “connector” and the PU customizing views offer the normal PU functionality and have no restrictions.

Therefore all visual (controls) and non-visual PU services for reading/writing/searching data and metadata can be used.

Configuration / Customizing

Step 1: Creating table(s) and include(s) for property data storage

First of all you’ve to think about the requirements for your properties: is a property single or multi valued, is a property language depended or not. Depending on this decision, you’ve to create a set of tables and includes.

Single valued language independent properties can be clustered in one table and single valued language dependent properties can be clustered in one table too, but for a multi valued property you need an own table. Therefore you’ll find 4 table types LIS, LDS, LIM and LDM. Explanation of abbreviations: L(anguage), D(ependent), I(independent), S(ingle valued) and M(ulti valued).

Each table type has to have two fixed includes in front, followed by the specific data include. The first include is SRM_PROP_PERS_KEY containing the fields for the client (MANDT) and for identifying an attribute record (REC_ID). The second include is SRM_PROP_PERS_ADMIN. It yields the fields for administrative data.

E.g. to introduce 2 language independent and single valued properties (SEMANTIC_ID and DESCRIPTION) i created the following table ZHHAATTRLIS and data include ZHHAATTRLIS_I…

.INCLUDE	SRM_PROP_PERS_KEY		
.INCLUDE	SRM_PROP_PERS_ADMIN		
.INCLUDE	ZHHAATTRLIS_I	STRU	0
SEMANTIC_ID	SRMCRQYSEMID	CHAR	64
DESCRIPTION	SRMCRQYDESCR	CHAR	64

And to introduce 1 language independent and multi valued property (KEYWORD) i created the following table ZHHAATTRLIM and data include ZHHAATTRLIM_I…

MANDT	MANDT	CLNT	3
GUID	SCMG_CASE_GUID	CHAR	32
SEQNO	INT4	INT4	10
.INCLUDE	ZHHAATTRLIM_I	STRU	0
KEYWORD	SRMGSKEYWD	CHAR	64

Step 2: Register table(s) and include(s) for property model

After creation of table and include set for property data storage, you've to register them in customizing tables, so that the default repository recognizes them.

(TODO: change this, if ready) Up to now you've to register table and include set by hand in the following customizing tables.

All tables/includes for single valued properties have to be registered in customizing table SRM_ATTR_MDL. E.g. for the example above a entry could look like following…

MANDT	000
MDL ID	HHA_AL
TABNAME LIS	ZHHAATTRLIS
STRUNAME LIS	ZHHAATTRLIS_I
TABNAME LDS	
STRUNAME LDS	
PROP PERS API	

This means that i introduced a new property model with ID HHA_AL and that I registered a table and a include for language independent single valued properties. A attribute model is a set of attribute IDs, which can be assigned to one or more elementtypes.

Language dependent and multi valued tables/includes have to be registered in customizing table SRM_ATTR_MDL_LDM.

Language independent and multi valued tables/includes have to be registered in customizing table SRM_ATTR_MDL_LIM. E.g. for the example above a entry could look like following…

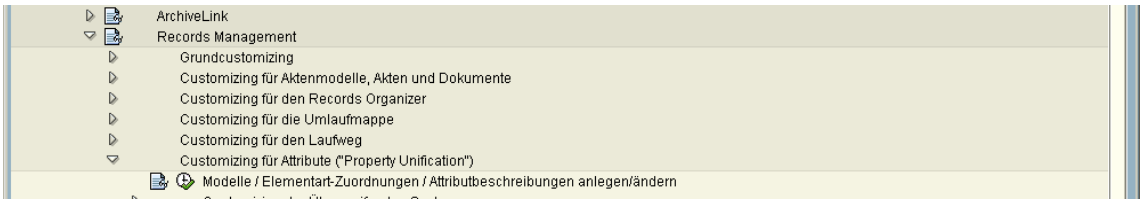
MANDT	000
MDL ID	HHA_AL
TABNAME	ZHHAATTRLIM
STRUNAME	ZHHAATTRLIM_I

Don't forget to transport these customizing entries, if they're not local!

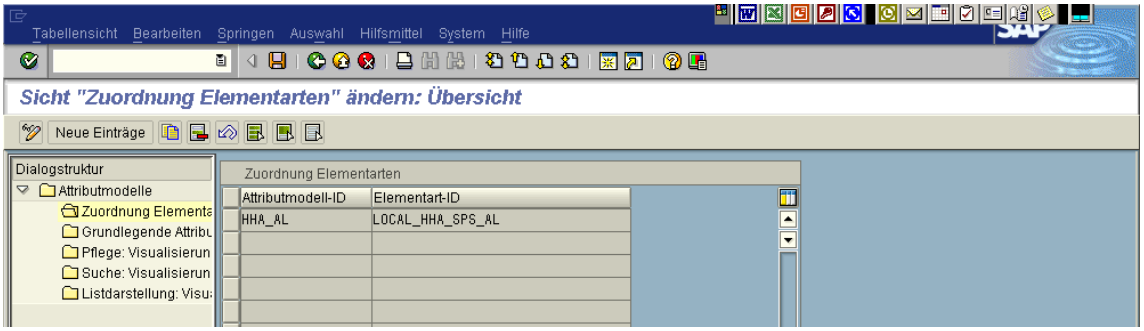
Step 3: Adjusting metadata for each property ID

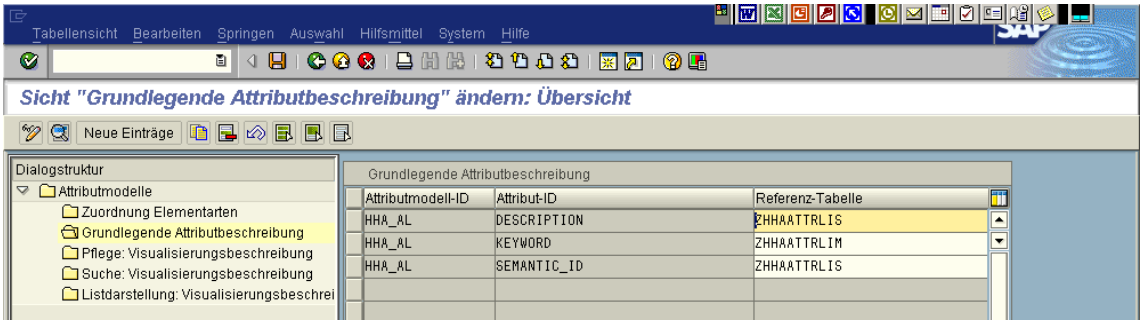
With the registration of the data tables/includes the entries in the metadata customizing tables are made automatically, but you've to adjust these entries.

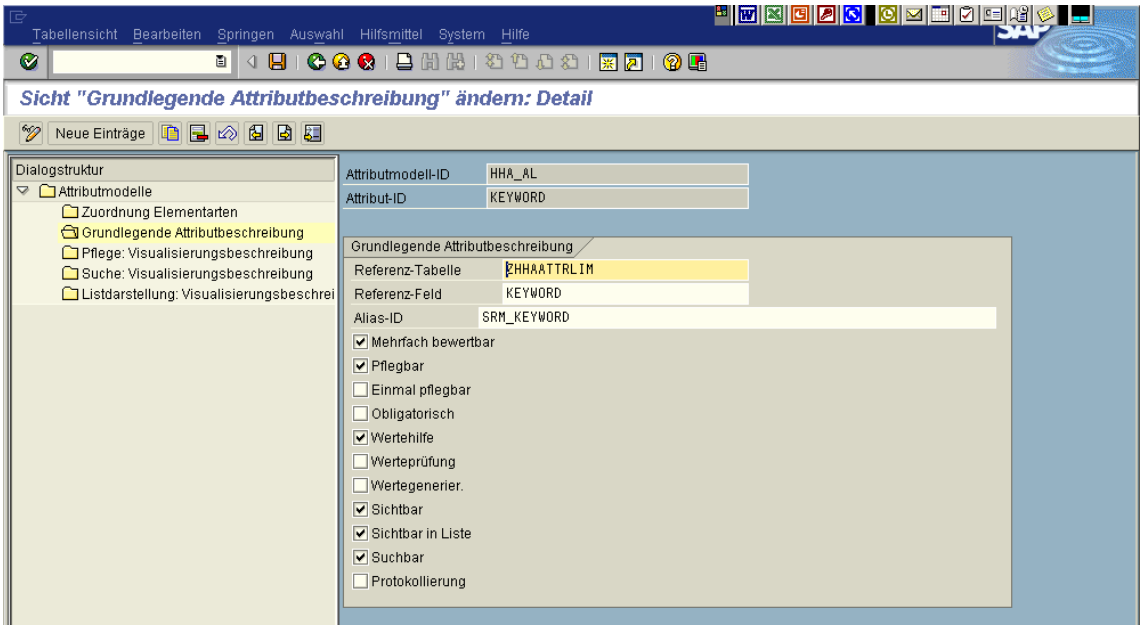
You can do so in customizing transaction SPRO at the following node…

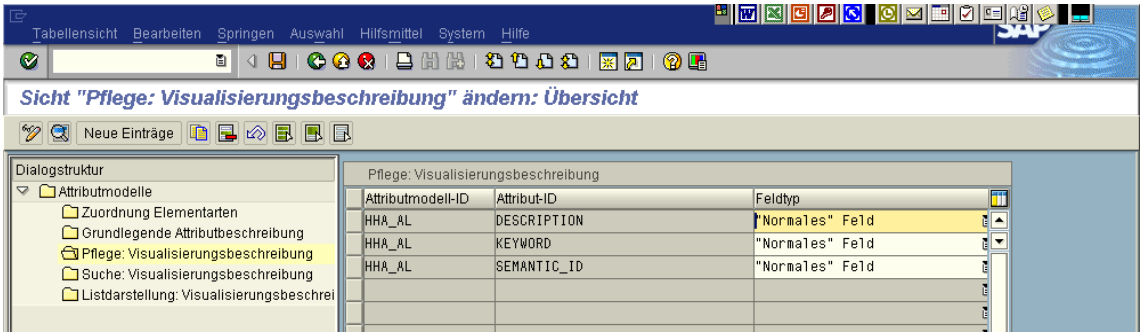


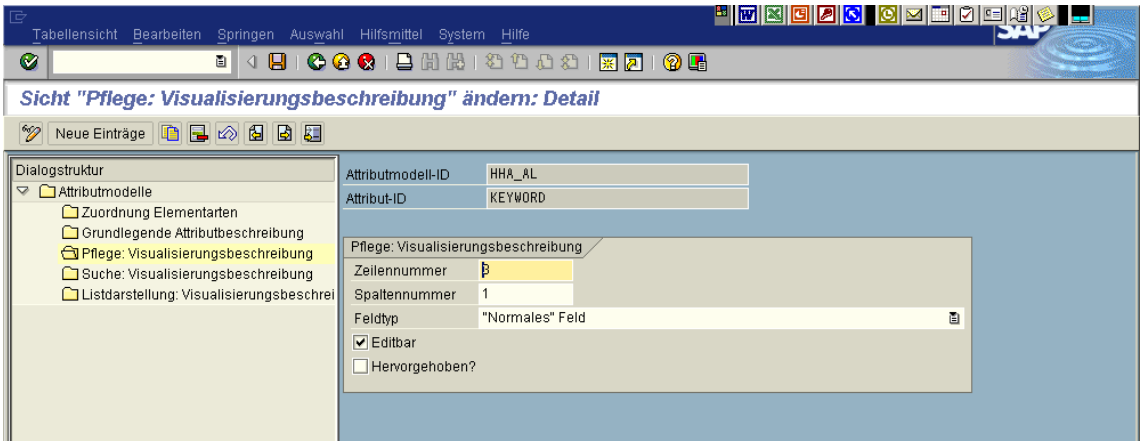
Here you'll find a view cluster which covers the complete metadata customizing for your attribute model (a attribute model is a set of attribute IDs, which can be assigned to one or more elementtypes), please adjust the entries for your needs…











Don’t forget to transport these customizing entries, if they’re not local!

Coding

After configuration/customizing 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 (optional): Working with an own attribute set key (instead of POID-Dir-ID)

If you don't want to work with generic POID-Directory-ID as key for a value set in property repository, you have the chance to specialize this.

Please create a class that inherits from CL_SRM_SP_PROP_REPOSITORY (the default PU implementation) and overwrite methods of interface IF_SRM_SP_PROP_REPOS_KEY.

After activation of this class, you've to register it in transaction SRMREGEDIT at your SP for repository classrole.

Step 2: 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 3: 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…

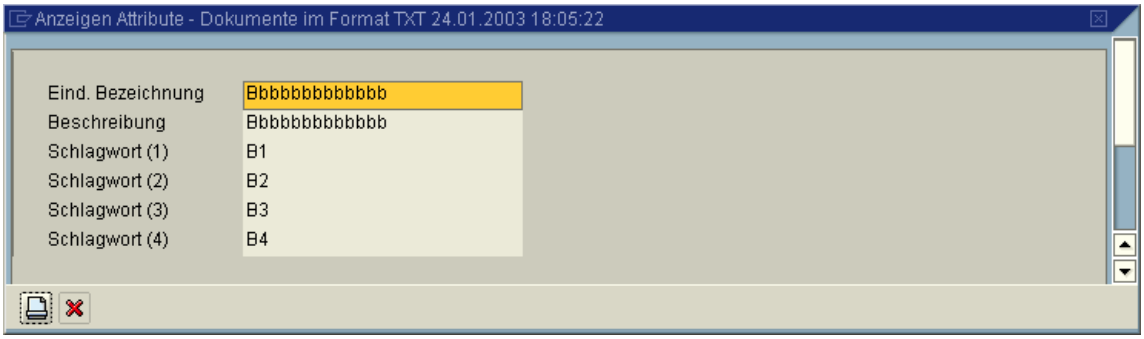
```
method IF_SRM_SP_ACTIVITIES~GET_INSTANCE_ACTIVITIES.

  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.
```

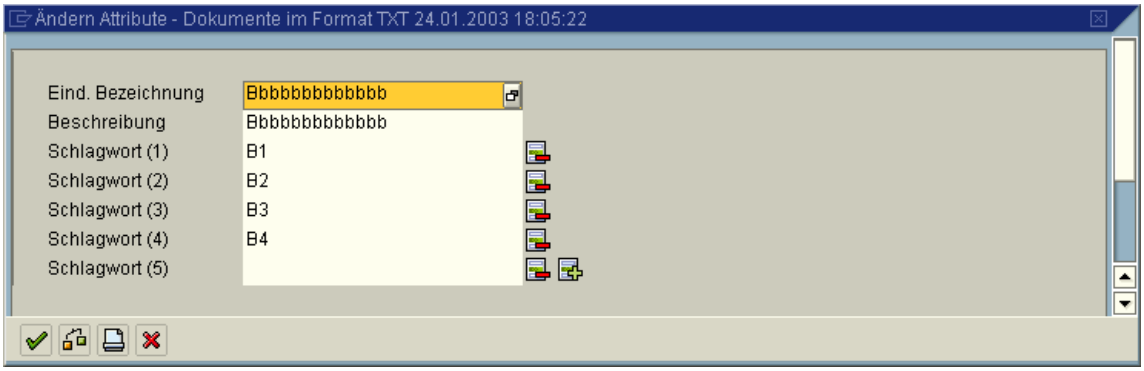
Now you've to handle the new instance activities PROP_* in your IF_SRM_SP_CLIENT_WIN~MY_ACTION implementation…

```
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 3 (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.