How to Implement Custom BAdIs for XStep Valuation and Generation

Applies to:

XStep-based PI Sheets (ERP PP-PI) or Electronic Work Instructions (ERP PP). For more information, visit the Manufacturing homepage.

Summary

When you use XSteps to define PI Sheets or Electronic Work Instructions the automatic valuation (e.g. material number) and the generation of elements (e.g. material component list) are important features. To provide even more flexibility new custom-defined valuation symbols or generation scopes can be included. This article provides a simple guide on how to set up the BAdIs needed for this.

Author: Dr. Arne Manthey

Company: SAP AG

Created on: 9 July 2008

Authors Bio

Dr. Arne Manthey was born in Stuttgart, Germany. His background includes a German 'Diplom' (Master) in chemical engineering in 1995 and a Ph. D. in Chemical Engineering (Aerosol science) in 2000. He has been a Consultant for Manufacturing in chemical and pharmaceutical industries at SAP Germany from 2000 - 2007. There he focused on PI sheets and OPC connectivity. Since April 2007 he is working as solution manager for application solution management manufacturing at SAP AG.

Table of Contents

Introduction

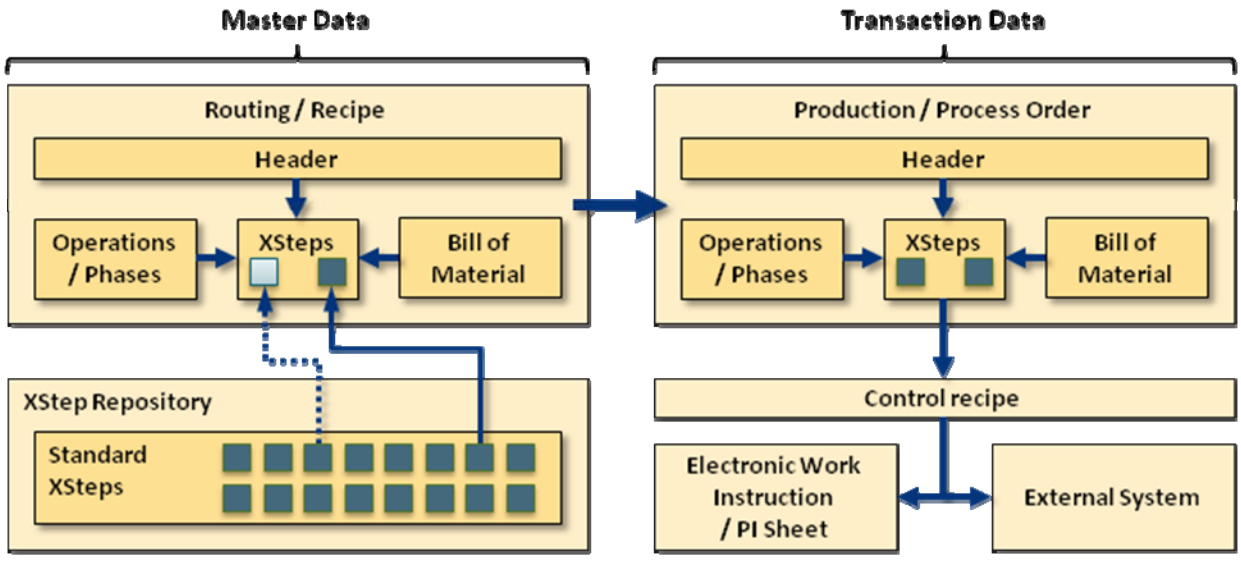
XSteps are the state-of-the-art technology to define PI Sheets (ERP PP-PI, R/3 Enterprise Ext. 2.0 and later) or Electronic Work Instructions (ERP PP, ERP 6.0 EhP3 / ECC 6.03). PI Sheets or Electronic Work Instructions (EWIs) are easy-to-use work sheets where all the manufacturing relevant data can be displayed or entered. Operators can work on these sheets as a one-stop-shop with no need to start multiple transactions in the ERP system.

In order to allow this flexibility all relevant data must be provided inside the XSteps. This is done with automatically valuated parameters and generation scopes for repeated elements (e.g. material components). There are numerous valuation symbols and generation scopes available in SAP standard. However there is always the need to provide additional information or to filter a generation scope by custom criteria.

This guide explains how to set up these BAdIs.

XStep Background

There are three locations for XSteps:



- Standard XStep (SXS) repository
  - Definition of generic building blocks that can be re-used inside the repository or in recipes/routings
  - Independent of specific recipes/routings or orders
- Master recipe (PP-PI) or standard routing (PP)
  - Standard XSteps can be included as **reference** (changes in the repository will also change the reference in the recipe/routing, dotted line in picture) or **copy** (decoupled from repository)
  - XSteps can be assigned to specific operations/phases
- Process order (PP-PI) or production order (PP)
  - All XSteps and SXS are copied from the routing/recipe. References will be exploded with the valid version of the order start date
  - On control recipe generation execution of all **generation scopes** (e.g. for all components) and **symbol valuations** (e.g. material number). This is the part where all order-relevant data flows into the XStep elements where you have defined it.
- Control recipe: Contains all information from the XSteps and either
  - Builds the instructions in the Electronic Work Instruction / PI Sheet
  - Is sent to an external system where the information is processed

BAdIs for XSteps

Beside the standard valuation symbols and generations scopes every customer can create own methods for valuation and generation. This is done with BAdI implementations. Every standard symbol and generation can be re-used in these BAdIs.

The XStep locations mentioned before are also distinguished during valuation of symbols and execution of generation scopes. Dependent on the location there are several restrictions:

- Standard XStep repository (Application XSV):
  No information of specific recipes, orders or phases/operations available, e.g.:
  - No Material number
  - No Component data
  - No Scheduling dates
- Master recipe or standard routing (Application MRC)
  No specific order information available, e.g.:
  - No Scheduling dates
  - No batch data
  - No variant configuration
- Process or production order (Application MOR):
  - No restrictions

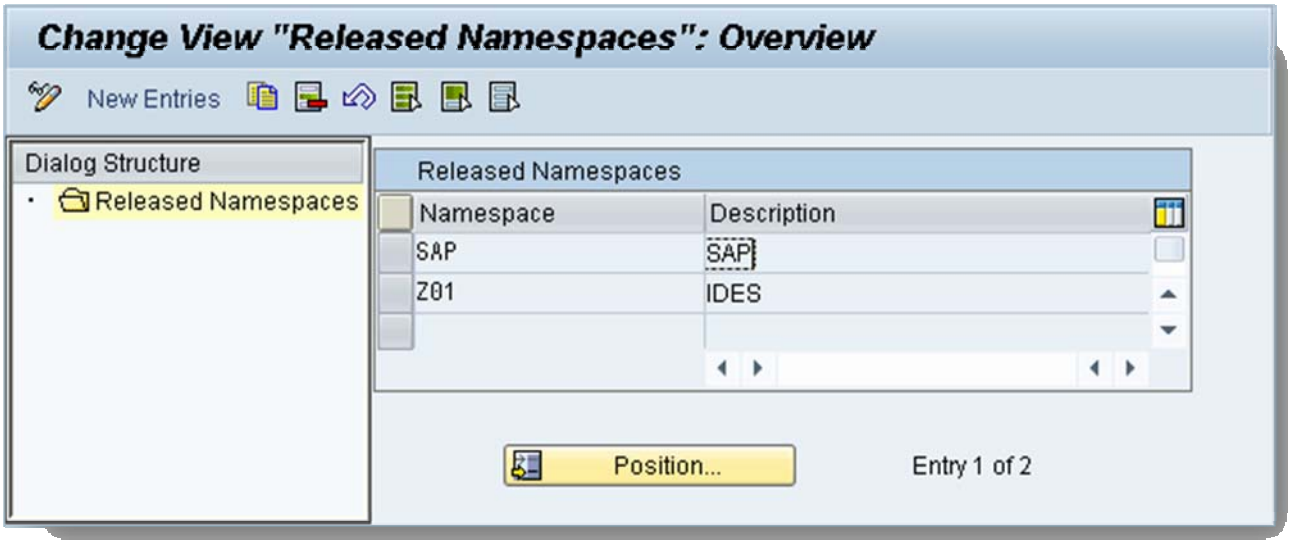
This is an important aspect that you need to consider when implementing the BAdIs. Since there is the possibility of simulation in all three applications also the generation and symbol valuation can be executed in these simulations. If you want some meaningful information to be displayed you need to check the calling application in the BAdI and set some dummy information. You can do this either inside the BAdI method (using a case statement) or by using different BAdI implementations which are controlled by the BAdI filter (as explained later)

Since this is of course more effort than just providing the data in the order application (MOR) you can as well leave the result blank for the other applications (XSV and MRC) and live with the results. As a matter of fact also many of the SAP standard objects do not offer meaningful (or dummy) content for every case. (For example, if you simulate a material component generation scope in the SXS repository there will be no information displayed)

Customizing

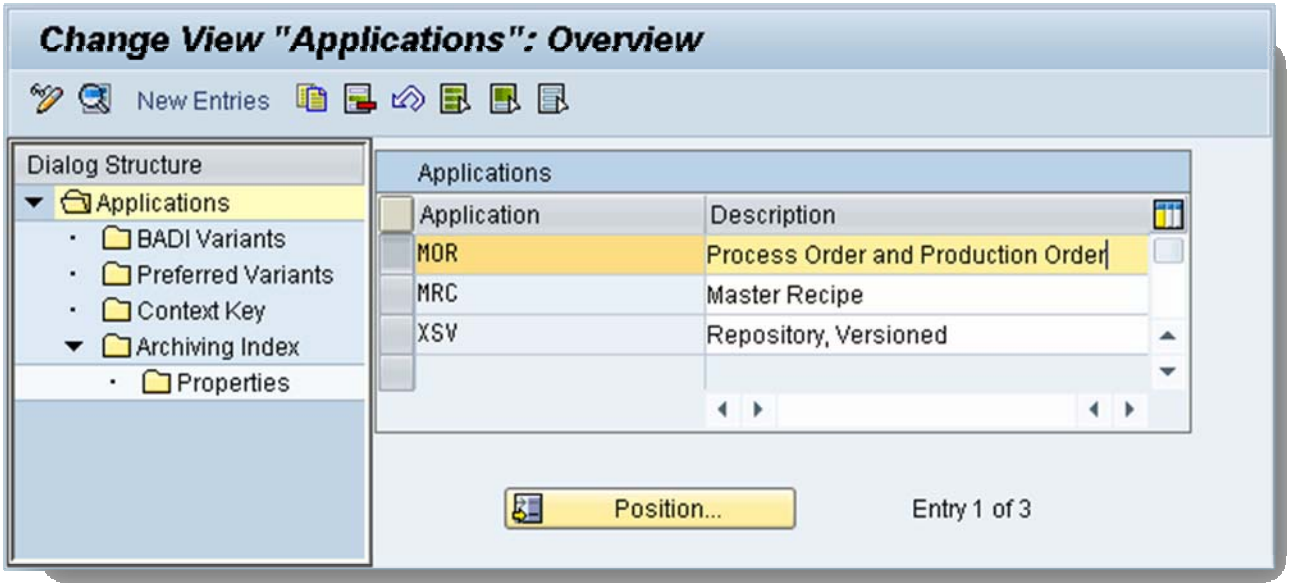
Release Namespaces [CMX04]

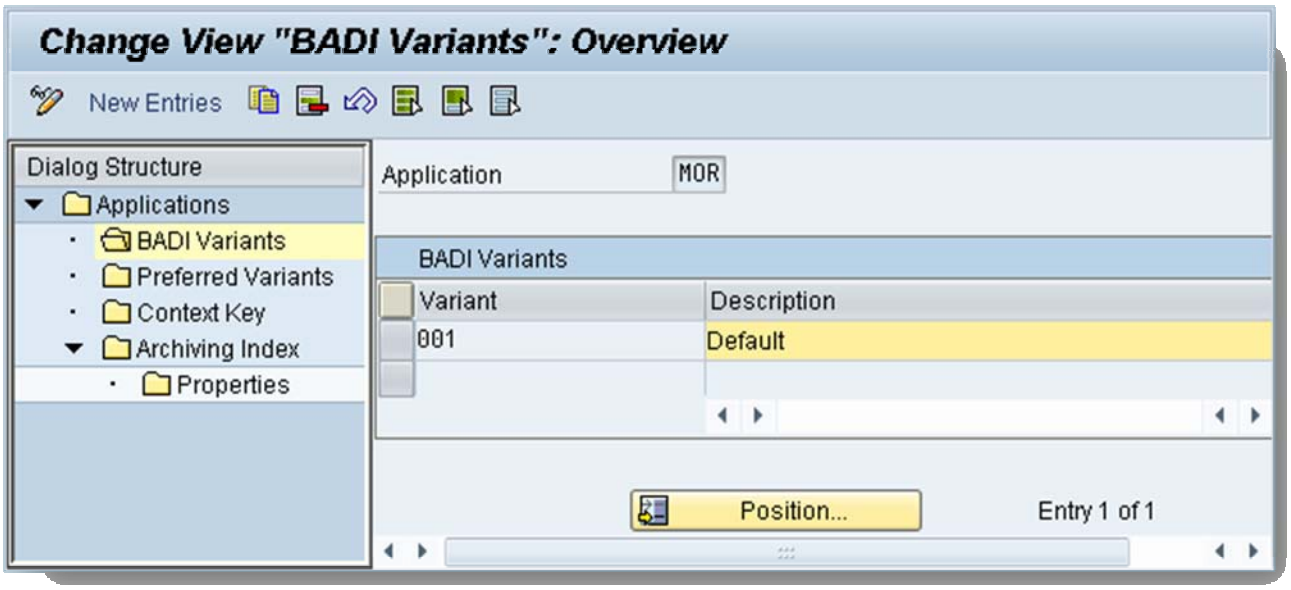
Here you have to release the namespaces defined by transaction CMX02



Applications & Variants [CMX01] - Optional

There might be a requirement to have different variants of BADIs active (e.g. one variant is active in summer the other in winter). For this purpose several variants can be maintained for the different applications.

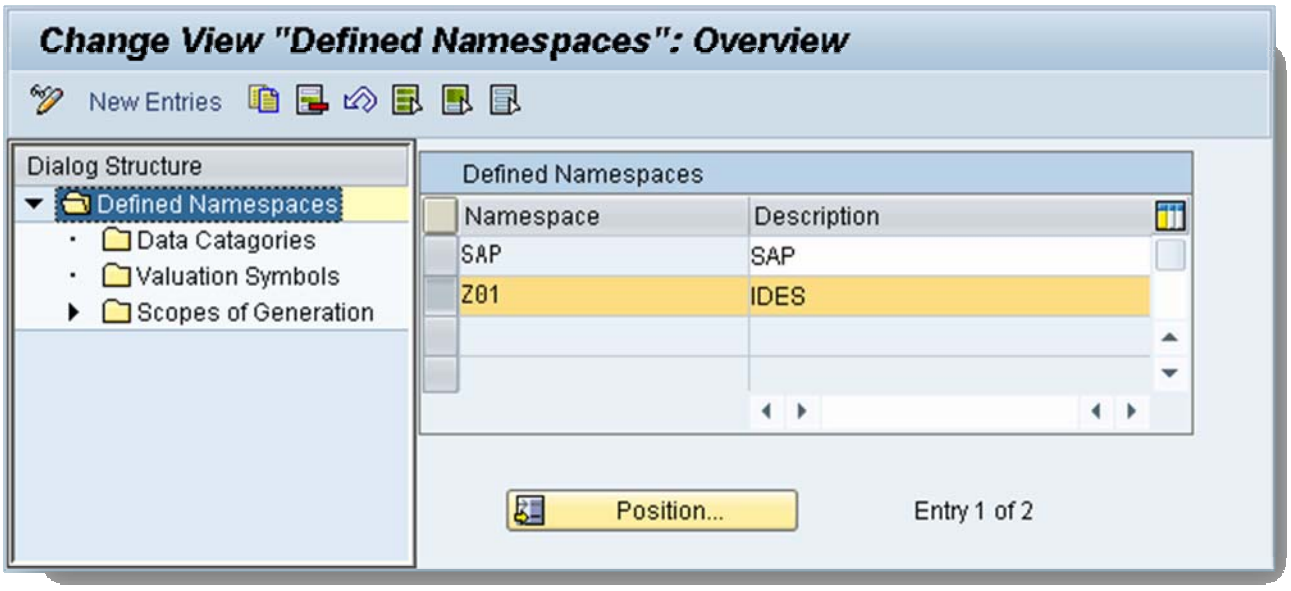




Scopes of Generation and Valuation Symbols [CMX02]

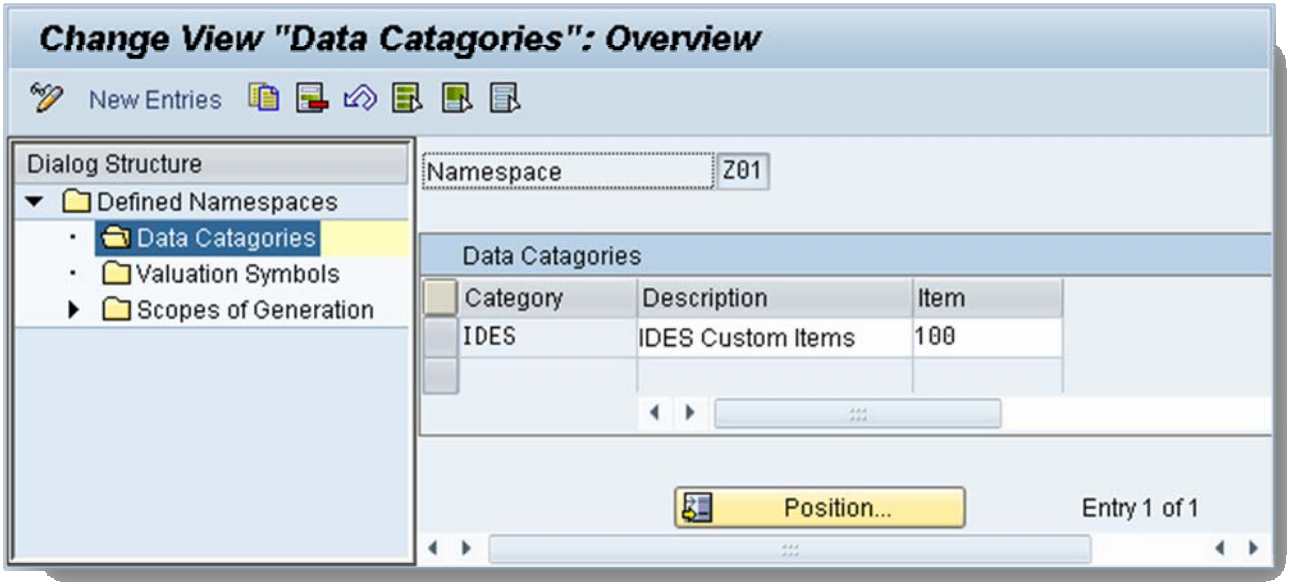
Here you define the structure of generation scopes and symbols in your namespace (including the predefined SAP namespace).

First you have to define one or more namespaces to better organize the BADIs. The standard namespace 'SAP' contains all the standard valuation symbols and generation scopes. When you create new namespaces you need to release those with transaction CMX04.



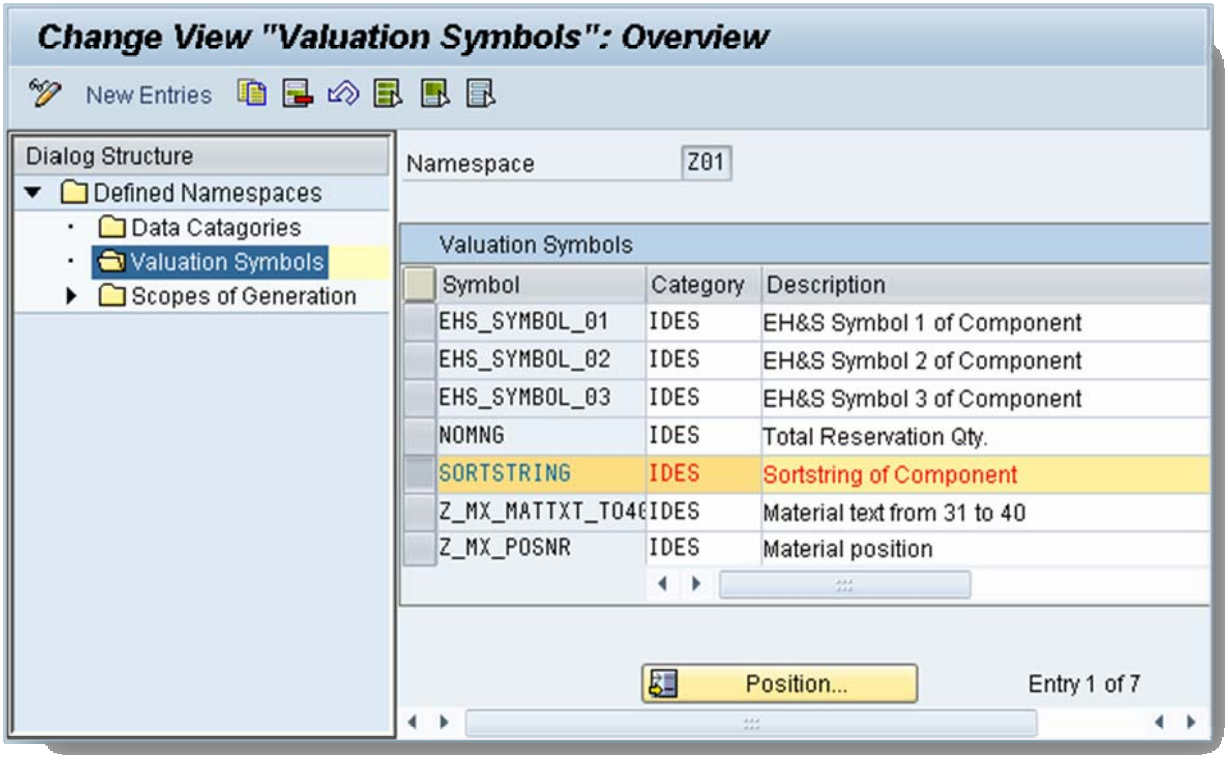
Data Categories

The category can be used to organize the symbols (Folder structure when browsing the valuation symbols). → Created category ‚IDES'



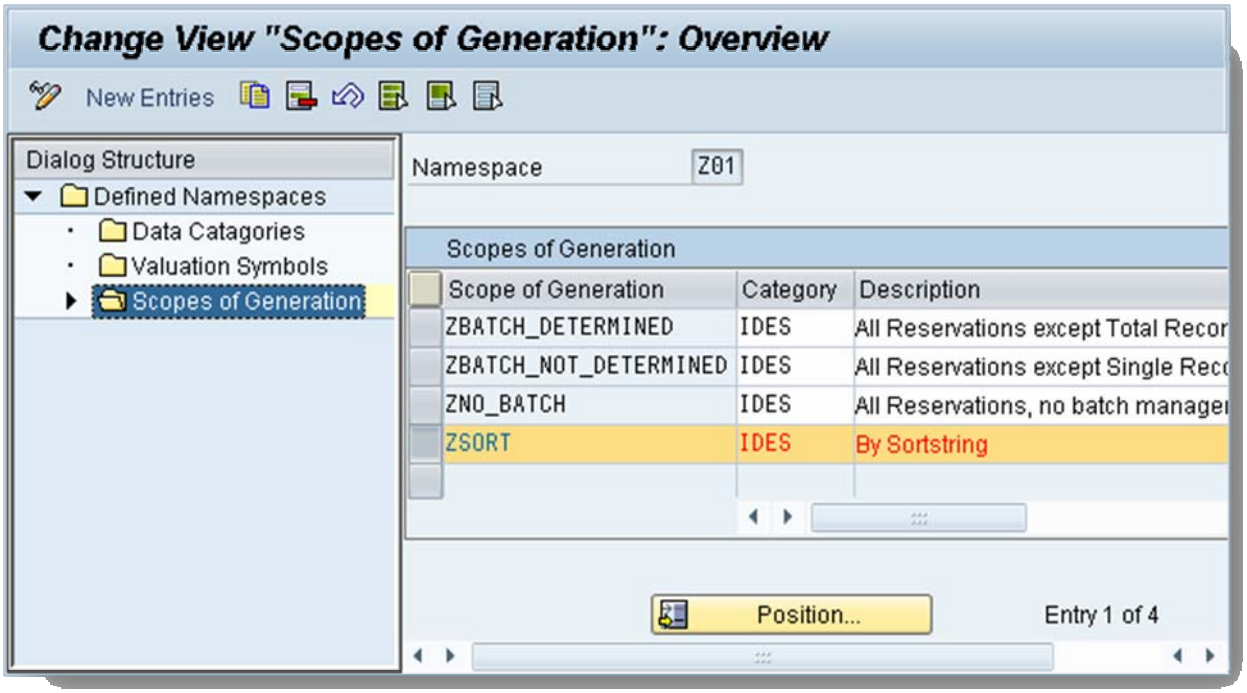
Valuation Symbols

Create the valuation symbols for the namespace that are used for the parameter definition. You must assign a data category to each valuation symbol. If you want to use your own valuation symbols, you must implement BAdI CMX_XS_SRV_SYM (described later in this document).

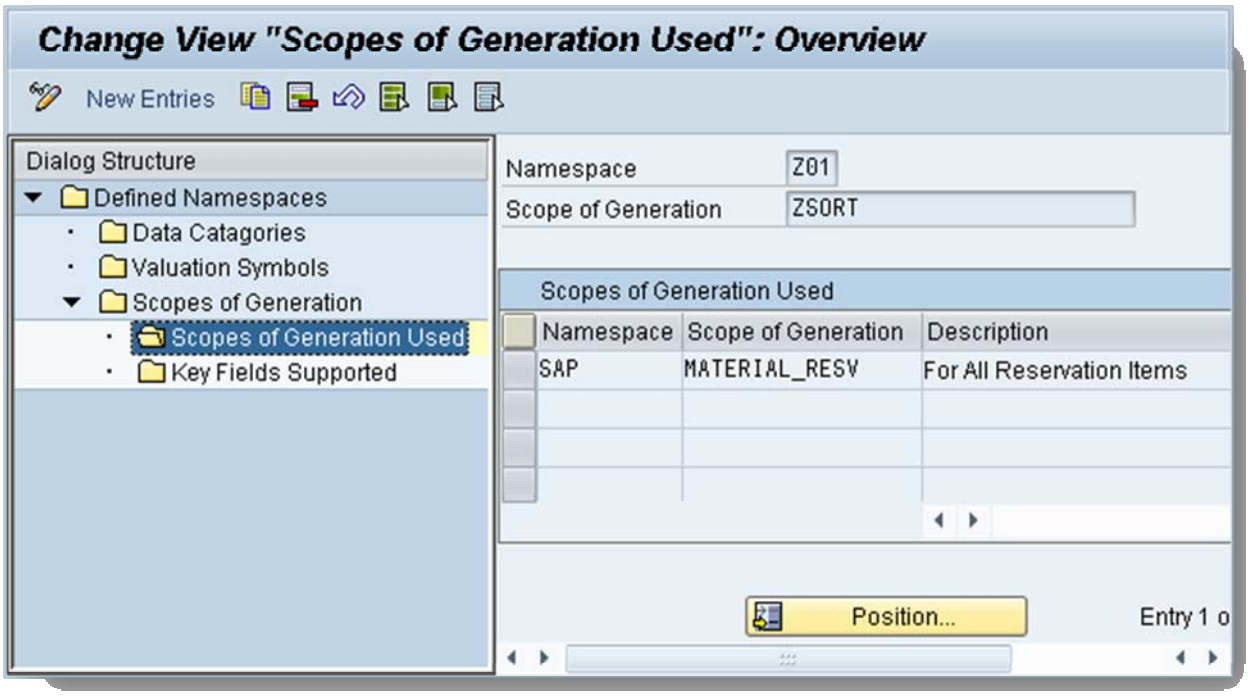


Scopes of Generation

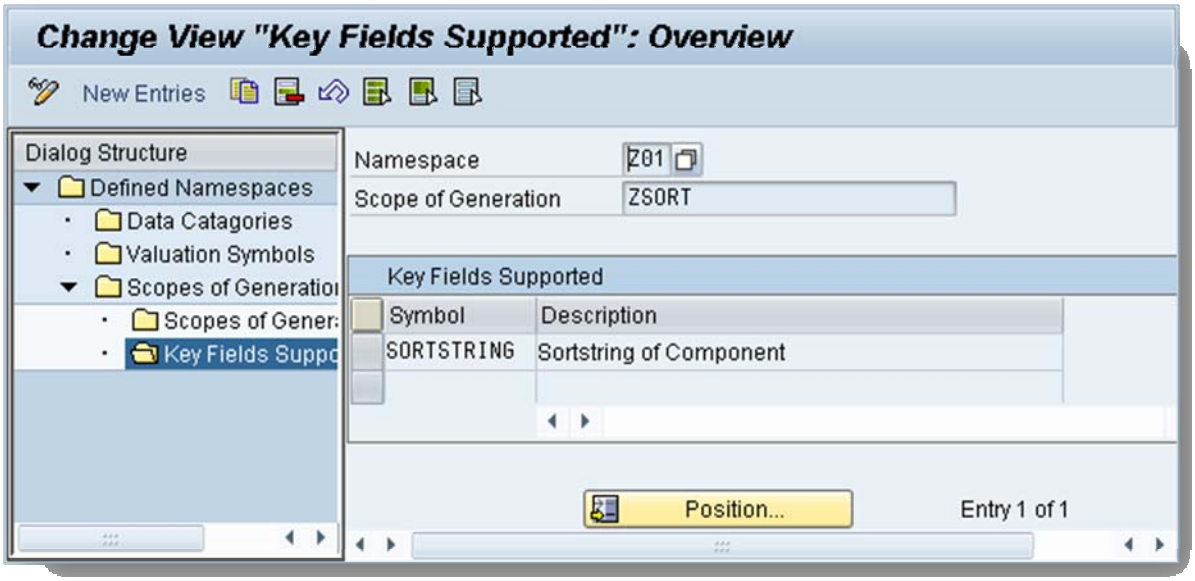
Here you add your own scopes of generation. You must assign a data category to the generation scope. If you want to use your own scopes of generation, you must implement the BAdI CMX_XS_SRV_GEN (described later in this document)



If the BAdI implementation for your own scope of generation contains further scopes of generation, you must list the namespaces for all scopes of generation used. The namespaces must all have been released.



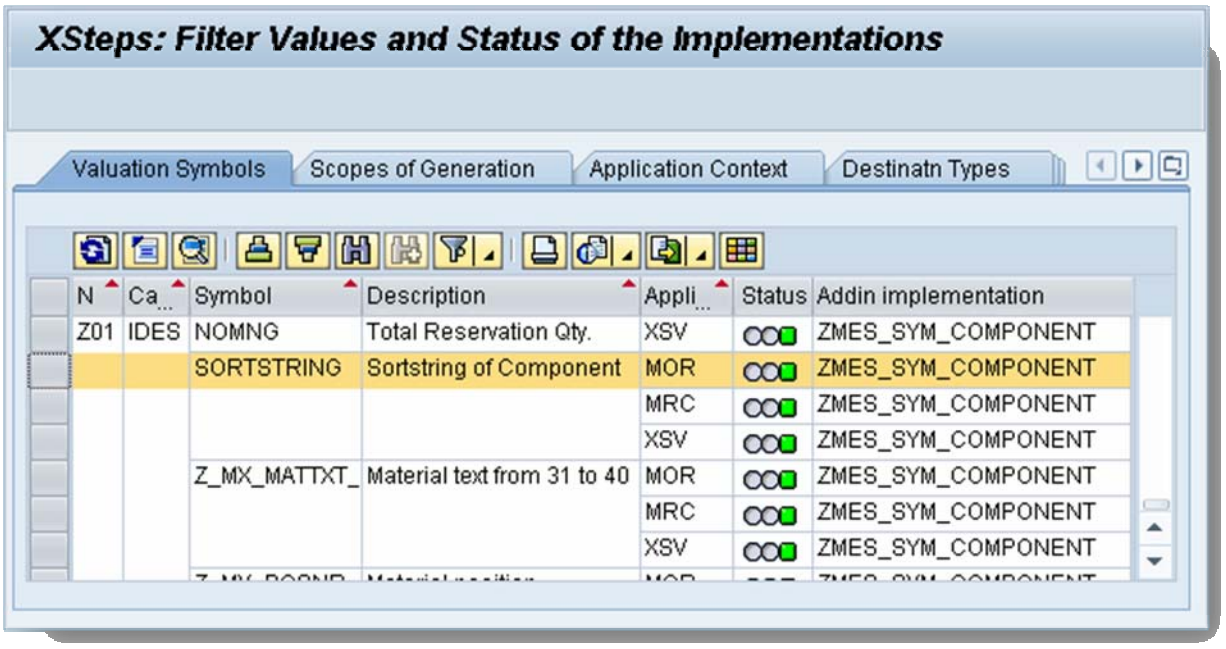
You can also assign key fields to the scope of generation that are set directly from the scope of generation:



Check Implementation Status [CMX05]

Here you can check the status of all XStep BAdI implementations. If a valuation symbol or generation scope is not active check the activation status:

- The Method (~GET_DATA)
- The Implementation



Note: This only checks whether there is active coding for each combination. The coding itself might not evaluate the elements correctly (e.g. due to conceptual errors)

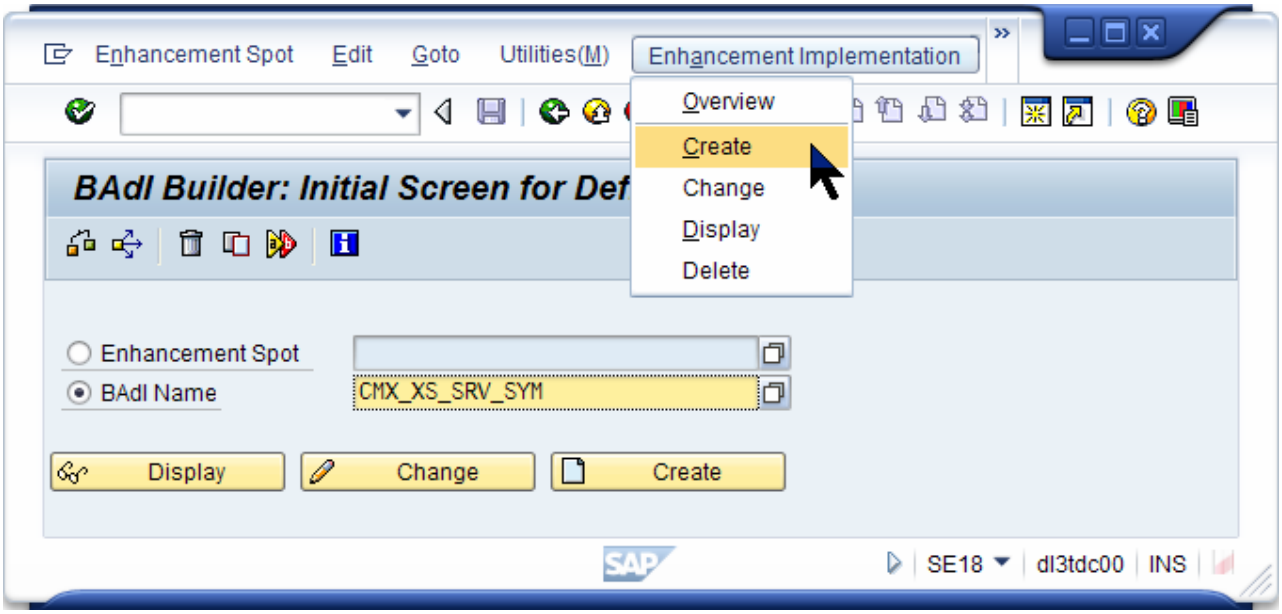
BAdI Implementation [SE18]

Create implementation

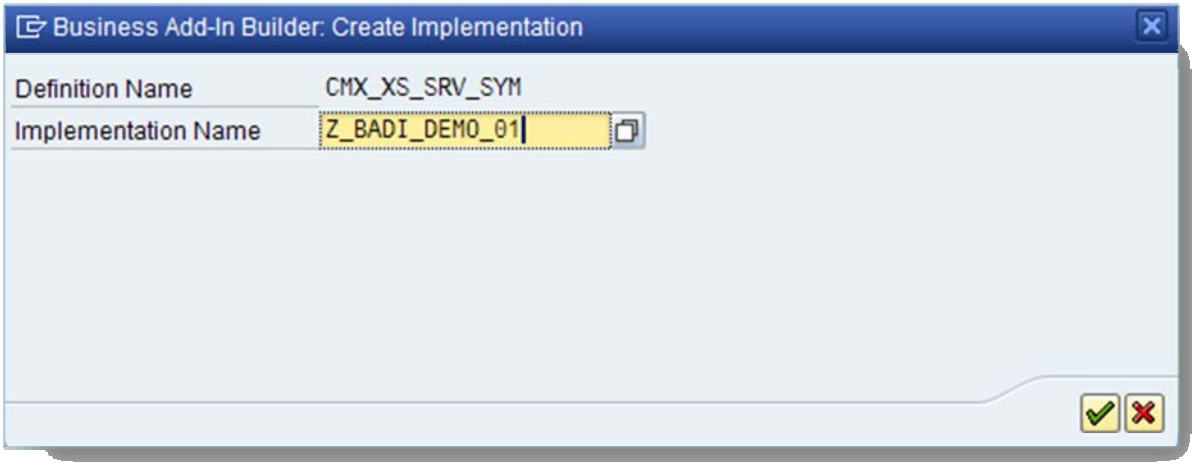
Enter BADI definition:

- CMX_XS_SRV_SYM for symbol valuation
- CMX_XS_SRV_GEN for generation scope

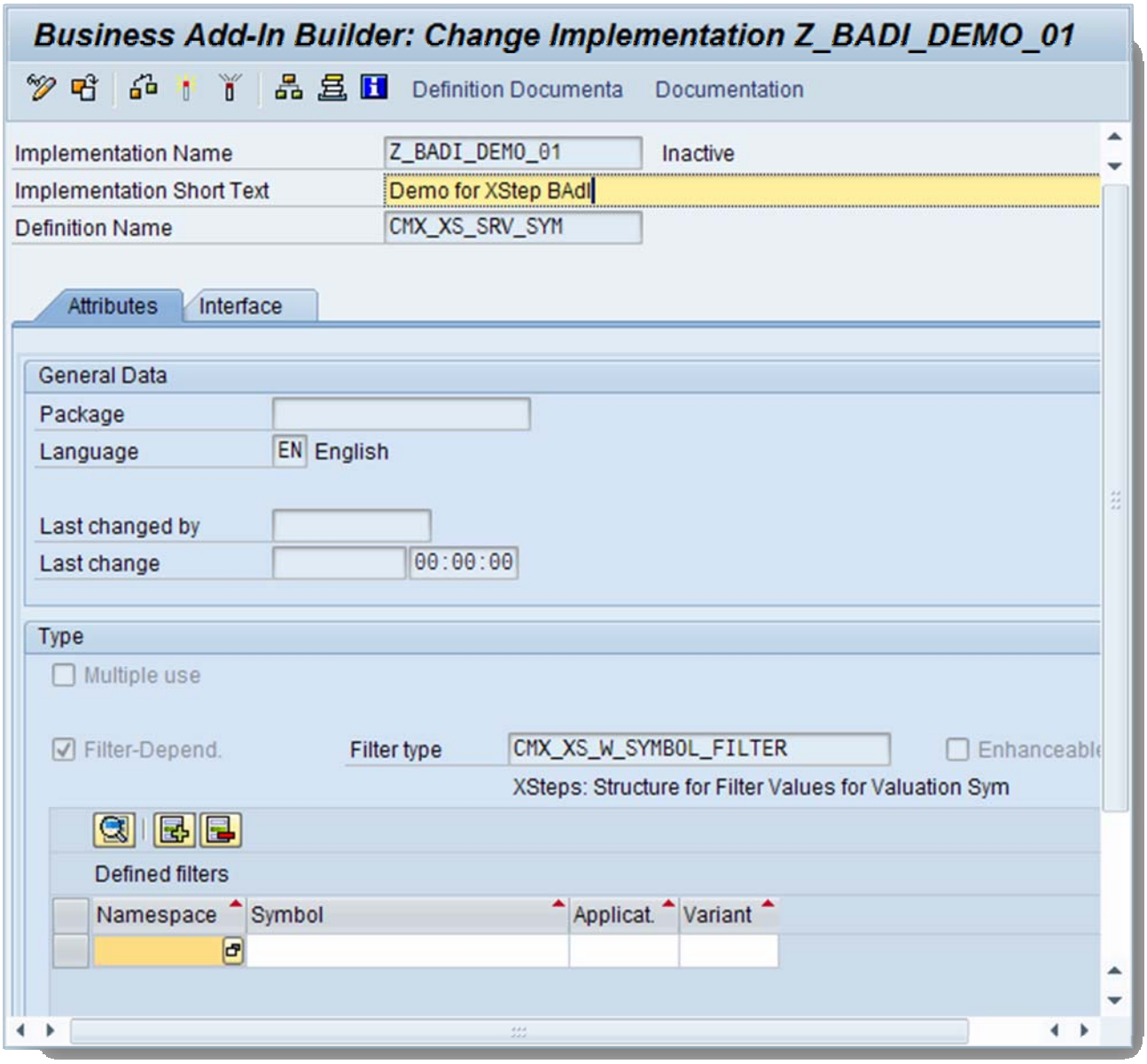
And create a new implementation:



Enter implementation name (e.g. ZPU_BI_SYM_PACK)



Enter short text and add at least one line for the filters:



Maintain Filter for the implementation

One implementation can be used to valuate several symbols in several applications (repository, recipe and process order). The decision which implementation is used for a symbol is done by filters.

To use one BAdI for all valuation symbols in your namespace (e.g. Z01) you would specify:

Namespace	Symbol	Applicat.	Variant
Z01	*	*	*

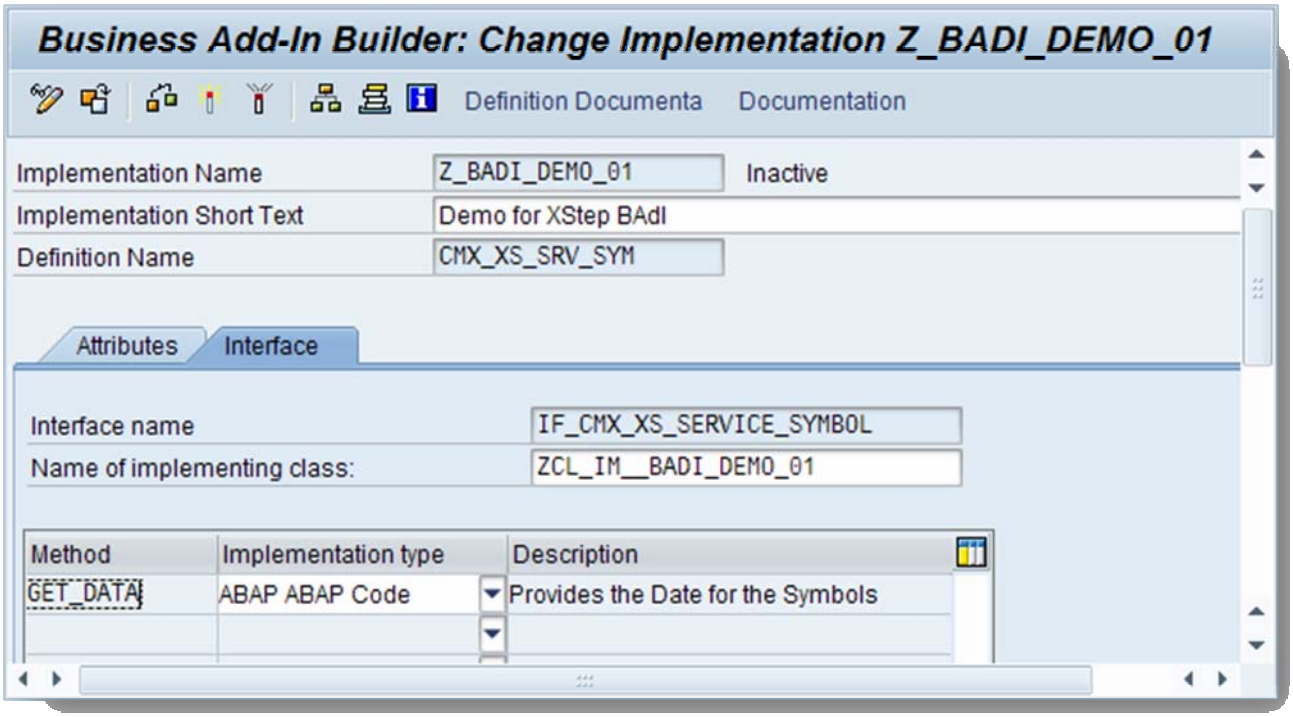
If you want to use this BAdI implementation just to provide dummy values in case of simulation in the SXS repository you would use:

Namespace	Symbol	Applicat.	Variant
Z01	SORTSTRING	XSV	*

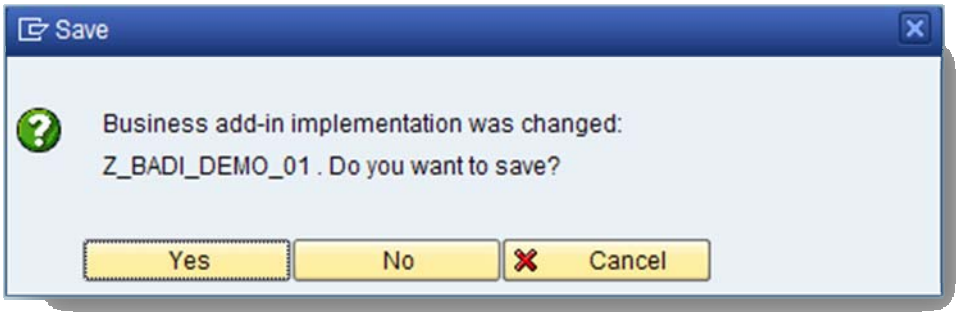
You should make sure that all existing BAdI implementation do not intersect in regard to their filters. For example, if you would have 2 BAdI implementations which have the above filters defined there would be an intersection for Z01/SORTSTRING/XSV due to the wildcards in the first filter.

Maintain coding (interface)

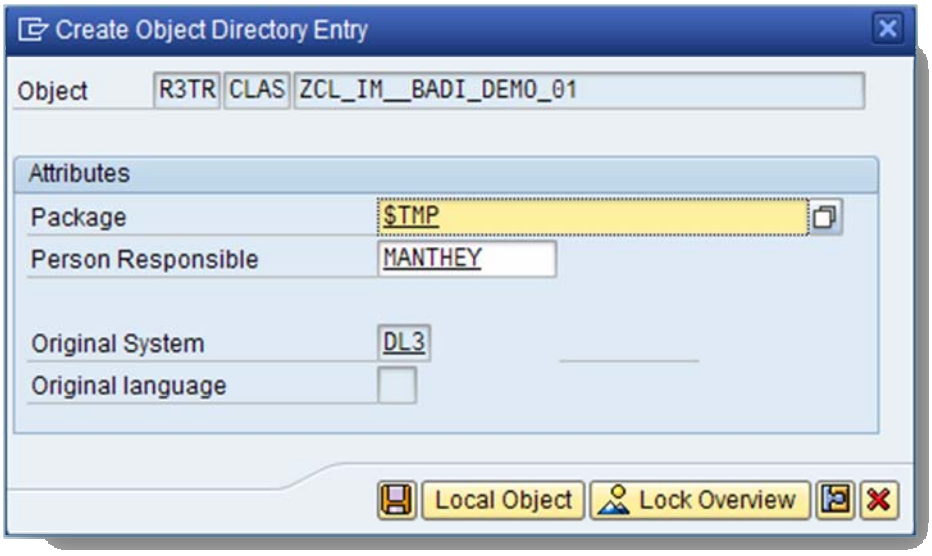
Select tab ‘Interface’ and double-click on method GET_DATA.



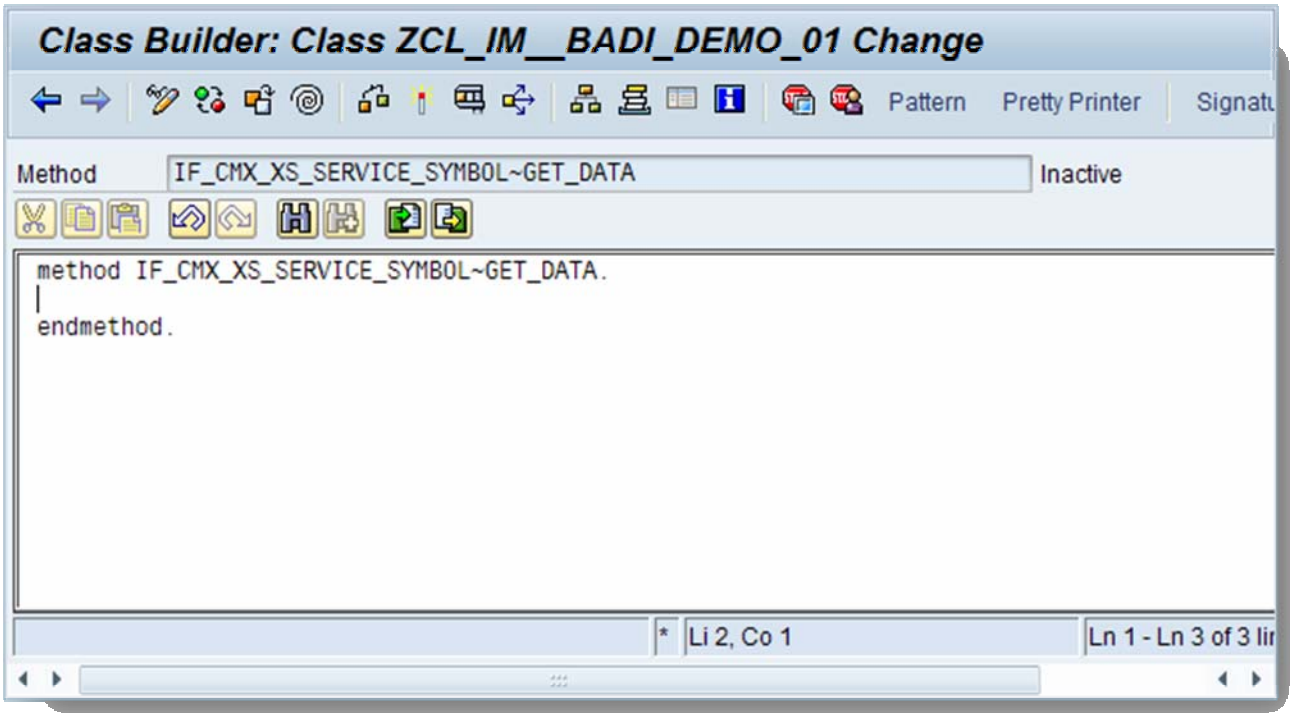
If you did not save your implementation so far you will get this popup:



Assign the implementation to a package:



Then maintain your coding:



After finishing the coding you need to activate both the method **and** the BAdI implementation! You can check this with CMX05 as explained before.

The BAdI implementation works similarly with the generation scope BAdI definition. There is some example coding available in the next chapter.

Now you can use and test your valuation symbols and generation scopes.

Sample coding

Symbol valuation

Sort string

The following sample code describes the custom symbol valuation of the sort string in the material list. Precondition to use this symbol is that the symbols for a reservation are known in this context (e.g. by using this symbol in an XStep that is generated for each material component). This means that you have to valuate also other symbols in the same XStep like:

- RESERVATION
- RESERVATION_ITEM

Parameter	Type	Typing Method	Associated Type
FLT_VAL	Importing	Type	CMX_XS_W_SYMBOL_FILTER
QUERY	Importing	Type Ref To	IF_CMX_XS_QUERY_SYMBOL

Code

```
method IF_CMX_XS_SERVICE_SYMBOL~GET_DATA.

*=====================================================================
* This is a very basic method for valuation of XStep symbols:
* - It contains a lot of hard coded elements!
* - There is no exception handling!
*
*=====================================================================

*=== Data declaration ================================================
  DATA: ls_tempsym       TYPE cmx_xs_w_symbol_name,
        ls_symbol        TYPE cmx_xs_w_symbol_name,
        l_ehs_symbol    TYPE        AUSP-ATWRT,
        l_rsnum          TYPE        resb-rsnum,
        l_rspos          TYPE        resb-rspos,
        l_rsart          TYPE        resb-rsart,
        l_sortf          TYPE        resb-sortf,
        l_matnr          TYPE        resb-matnr,
        l_qty            TYPE        resb-bdmng,
        ls_resbd         TYPE        resbd.

*=== Preparation =====================================================

*--- Set namespace of the valuated symbols --------------------------
ls_tempsym-namespace = 'SAP'.

IF flt_val-application = 'MOR'.

*   get reservation number from generated step
    ls_tempsym-symbol = 'RESERVATION'.
    query->get_value_into_numc(
                  EXPORTING
                       symbol  = ls_tempsym
                  IMPORTING
                        data   = l_rsnum ).
*   get reservation item from generated step
    ls_tempsym-symbol = 'RESERVATION_ITEM'.
    query->get_value_into_numc(
                  EXPORTING
```

```
                        symbol  = ls_tempsym
                    IMPORTING
                         data   = l_rspos ).
*   get reservation item type from generated step
    ls_tempsym-symbol = 'RESERVATION_ITEM_TYPE'.
    query->get_value_into_string(
                    EXPORTING
                        symbol  = ls_tempsym
                    IMPORTING
                         data   = l_rsart ).
*   Read corresponding reservation data
    CALL FUNCTION 'CO_BT_RESB_READ_WITH_KEY'
      EXPORTING
        flg_resbd       = space
        no_read_from_db = 'X'
        rsart_imp       = l_rsart
        rsnum_imp       = l_rsnum
        rspos_imp       = l_rspos
      IMPORTING
        resbd_exp       = ls_resbd
      EXCEPTIONS
        not_found       = 1
        OTHERS          = 2.

      l_sortf = ls_resbd-sortf.
      l_matnr = ls_resbd-matnr.

      IF ls_resbd-nomng IS initial.
        l_qty = ls_resbd-bdmng.
      ELSE.
        l_qty = ls_resbd-nomng.
      ENDIF.

ELSE.

* Put some dummy evaluation when the calling application is
* not the process order
   l_sortf = '<no value>'.

ENDIF.

*=== Valuation depending on the filter symbol ======================
  ls_symbol-namespace = flt_val-namespace.
  ls_symbol-symbol    = flt_val-symbol.

  CASE flt_val-symbol.

*___ Sortstring of the Reservation Item _____________________________
    WHEN 'SORTSTRING'.
      query->set_value_from_string( symbol = ls_symbol
                                    data   = l_sortf
                                    domain = 'DDIC/SORTP' ).

  ENDCASE.
*====================================================================
*Different Valuation Methods:
*SET_VALUE_FROM_FLOAT
*SET_VALUE_FROM_INTEGER
*SET_VALUE_FROM_NUMC
*SET_VALUE_FROM_PACKED
*SET_VALUE_FROM_STRING
*SET_VALUE_FROM_STRUCTURE
*SET_VALUE_FROM_TABLE
*SET_VALUE_FROM_TIME
```

```
*SET_VALUE_FROM_XSTRING
*SET_VALUE_FROM_TEXT

ENDMETHOD.
```

Notes on the valuation method (query->set_value_from…):

The method depends on the type of the data field you want to update. The following methods are available:

- SET_VALUE_FROM_FLOAT
- SET_VALUE_FROM_INTEGER
- SET_VALUE_FROM_NUMC
- SET_VALUE_FROM_PACKED
- SET_VALUE_FROM_STRING
- SET_VALUE_FROM_STRUCTURE
- SET_VALUE_FROM_TABLE
- SET_VALUE_FROM_TIME
- SET_VALUE_FROM_XSTRING
- SET_VALUE_FROM_TEXT

To determine which domain string you have to use, you have to go to the field in DDIC that you used to define the variable and take the name of the data element.

Example:

In the coding above the variable l_sortf was defined as follows:

```
l_sortf TYPE resb-sortf
```

Call table RESB in SE11 and look for field SORTF. There the data element SORTP is used to define the field. Therefore the domain you have to use is 'DDIC/SORTP'.

Don't use the domain that you find in the data element in DDIC! (In this example 'CHAR10')

Generation scope

Simple generation with filter

The basis of this generation scope is the SAP standard scope of generation 'MATERIAL_RESV' ('For All Reservation Items'). Additionally a user-definable filter option is provided: If there is a parameter in the XStep that is called 'F_SORT' the value of that parameter is used to filter the resulting list.

Example: If F_SORT='A*' the generation scope returns all material reservations where the sort string begins with an 'A'.

Parameter	Type	Typing Method	Associated Type
FLT_VAL	Importing	Type	CMX_XS_W_GENERATION_FILTER
QUERY	Importing	Type Ref To	IF_CMX_XS_QUERY_GENERATION
Code			

```
METHOD IF_CMX_XS_SERVICE_GENERATION~GET_DATA.

  DATA:  l_count          TYPE        i,
         l_index          TYPE        i,
         l_para_name      TYPE        cmx_xs_param_name,
         l_para_value     TYPE        cmx_types_w_value,
         l_sort           TYPE        usrchar20,
         l_rsnum          TYPE        resb-rsnum,
         l_rspos          TYPE        resb-rspos,
         l_rsart          TYPE        resb-rsart,
         ls_resbd         TYPE        resbd,
         ls_symbol        TYPE        cmx_xs_w_symbol_name,
         ls_generation    TYPE        cmx_xs_w_generation_name,
         lt_parameter     TYPE        cmx_xs_t_parameter,
         lo_parameter     TYPE REF TO if_cmx_xs_parameter,
         lo_step          TYPE REF TO if_cmx_xs_step,
         lo_query_gen     TYPE REF TO if_cmx_xs_query_symbol.

* Execute the standard generation for all operations
  ls_generation-namespace = 'SAP'.
  ls_generation-generation = 'MATERIAL_RESV'.
  l_count = query->add_query( ls_generation ).

* Get the parameter SORT from the step
* If it is not given, treat it as space
  lo_step = query->get_step( ).
  lt_parameter = lo_step->get_parameters( ).
  LOOP AT lt_parameter INTO lo_parameter.
    l_para_name = lo_parameter->get_name( ).
    IF l_para_name = 'F_SORT'.
      l_para_value = lo_parameter->get_value( ).
      CALL METHOD l_para_value-domobj->conv_into_string
        EXPORTING
          intval = l_para_value-intval
        IMPORTING
          data   = l_sort.
      EXIT.
    ENDIF.
  ENDLOOP.


* Check the sortstring of every generated reservation item
  l_index = 1.
  WHILE l_index <= l_count.
*   Determine generated step # l_index
    lo_query_gen = query->get_symbols( l_index ).
    ls_symbol-namespace = 'SAP'.
*   get reservation number from generated step
    ls_symbol-symbol = 'RESERVATION'.
    lo_query_gen->get_value_into_numc(
                   EXPORTING
                        symbol  = ls_symbol
                   IMPORTING
                         data   = l_rsnum ).
*   get reservation item from generated step
    ls_symbol-symbol = 'RESERVATION_ITEM'.
    lo_query_gen->get_value_into_numc(
                   EXPORTING
                        symbol  = ls_symbol
                   IMPORTING
                         data   = l_rspos ).
*   get reservation item type from generated step
    ls_symbol-symbol = 'RESERVATION_ITEM_TYPE'.
```

```
    lo_query_gen->get_value_into_string(
                   EXPORTING
                        symbol  = ls_symbol
                   IMPORTING
                         data   = l_rsart ).
*   Read corresponding reservation data
    CALL FUNCTION 'CO_BT_RESB_READ_WITH_KEY'
      EXPORTING
        flg_resbd       = space
        no_read_from_db = 'X'
        rsart_imp       = l_rsart
        rsnum_imp       = l_rsnum
        rspos_imp       = l_rspos
      IMPORTING
        resbd_exp       = ls_resbd
      EXCEPTIONS
        not_found       = 1
        OTHERS          = 2.
*   If the value of the parameter does not match,
*   delete the generated step
    IF ls_resbd-sortf NP l_sort.
      query->delete( l_index ).
      l_count = l_count - 1.
    ELSE.
      l_index = l_index + 1.
    ENDIF.
  ENDWHILE.

ENDMETHOD.
```

Material components w/o context restriction

When you use the standard generation scope for material components in an XStep environment where a phase context is set the result will always be restricted to the components that are assigned to that phase. Sometimes it can be useful to have the complete list of components despite of that context.

This method builds a completely new generation (in contrast to the last example where a standard generation scope was re-used). The filtering via sort string is also used as explained before.

Parameter	Type	Typing Method	Associated Type
FLT_VAL	Importing	Type	CMX_XS_W_GENERATION_FILTER
QUERY	Importing	Type Ref To	IF_CMX_XS_QUERY_GENERATION

Code

```
method IF_CMX_XS_SERVICE_GENERATION~GET_DATA.

*=====================================================================
* Method for generation of material components independent of the
* used context
*
* Derived from the standard class CL_IM_COCR_CMX_BI_GEN_MAT and
* method ORDER_MAT_GENERATE
*
*=====================================================================


*=== Data declaration ================================================
  DATA:  l_count          TYPE        i,
         l_index          TYPE        i,
         l_para_name      TYPE        cmx_xs_param_name,
         l_para_value     TYPE        cmx_types_w_value,
         l_sort           TYPE        usrchar20,
```

```
        l_rsnum          TYPE        resb-rsnum,
        l_rspos          TYPE        resb-rspos,
        l_rsart          TYPE        resb-rsart,
        l_lines_comp     TYPE        sy-tabix,
        ls_resbd         TYPE        resbd,
        ls_symbol        TYPE        cmx_xs_w_symbol_name,
        ls_generation    TYPE        cmx_xs_w_generation_name,
        ls_context       TYPE        cocr_cmx_s_con_data,
        lt_resbd         TYPE TABLE OF resbd,
        lt_parameter     TYPE        cmx_xs_t_parameter,
        lo_parameter     TYPE REF TO if_cmx_xs_parameter,
        lo_step          TYPE REF TO if_cmx_xs_step,
        lo_query_gen     TYPE REF TO if_cmx_xs_query_symbol,
        lo_context       TYPE REF TO if_cmx_xs_context,
        lo_symbol        TYPE REF TO if_cmx_xs_query_symbol.


*=== Preparation ====================================================

*___ Get some parameters from the step _____________________________
* If it is not given, treat it as space
  lo_step = query->get_step( ).
  lt_parameter = lo_step->get_parameters( ).
  LOOP AT lt_parameter INTO lo_parameter.
    l_para_name = lo_parameter->get_name( ).
*   ___ Sortstring ____________________________________________________
    IF l_para_name = 'F_SORT'.
      l_para_value = lo_parameter->get_value( ).
      CALL METHOD l_para_value-domobj->conv_into_string
        EXPORTING
          intval = l_para_value-intval
        IMPORTING
          data   = l_sort.
      EXIT.
    ENDIF.
  ENDLOOP.
  IF l_sort IS INITIAL.
    l_sort = '*'.
  ENDIF.


* Get context
  lo_context = query->get_context( ).
  CHECK NOT lo_context IS INITIAL.

*--- Copied and adapted from standard method xs_get_context
* (class CL_COCR_CMX_TOP) ---
*      ls_context = me->xs_get_context( lo_context ).
  DATA:  ls_context_root  TYPE        cocr_cmx_s_cr_data,
         ls_context_step  TYPE        cocr_cmx_s_cs_data.
  CLEAR ls_context.
  IF lo_context->get_category( ) = if_cmx_xs_context=>co_category_root.
    CALL METHOD lo_context->get_data
      IMPORTING
        data = ls_context_root.
    ls_context-aufnr  = ls_context_root-aufnr.
    ls_context-aufpl  = ls_context_root-aufpl.
    ls_context-objtyp = 'H'.
  ELSE.
    CALL METHOD lo_context->get_data
      IMPORTING
        data = ls_context_step.
    ls_context-aufnr  = ls_context_step-aufnr.
    ls_context-aufpl  = ls_context_step-aufpl.
    ls_context-aplzl  = ls_context_step-aplzl.
```

```
    ls_context-objtyp = ls_context_step-objtyp.
  ENDIF.


  CHECK NOT ls_context-aufnr IS INITIAL.

* Read RSNUM for order and check if reservation buffer
* table is filled
  CALL FUNCTION 'CO_BT_CAUFV_READ_WITH_KEY'
    EXPORTING
      aufnr_act      = ls_context-aufnr
      no_dialog_info = 'X'
    IMPORTING
      rsnum_exp      = l_rsnum
    EXCEPTIONS
      not_found      = 1
      OTHERS         = 2.

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

  PERFORM check_read_db IN PROGRAM saplcobc USING l_rsnum.

* Generation for order components/materials
  CALL FUNCTION 'CO_BC_RESBD_TAB_TO_ORDER_GET'
    EXPORTING
      aufnr_imp          = ls_context-aufnr
      flg_check_log_loe  = 'X'
      flg_check_vbkz_del = 'X'
    TABLES
      resbd_tab          = lt_resbd.

* Delete components without proper sort string
  DELETE lt_resbd WHERE sortf NP l_sort.

* Here come further restrictions (other generation scopes)
  CASE flt_val-generation.

    WHEN 'ZPP_RES_NOCONTEXT'.
      " Do nothing else

    when 'ZPP_BULKCHARGEN'.
      DELETE lt_resbd WHERE postp NE 'Z'.
      DELETE lt_resbd WHERE SPLKZ NE '2'.

  ENDCASE.


* Create query objects for products (this is where the generation
* of lines happens)
  DESCRIBE TABLE lt_resbd LINES l_lines_comp.
  CHECK l_lines_comp > 0.
  query->set_count( l_lines_comp ).

* Set namespace of symbols
  ls_symbol-namespace = 'SAP'.

* Here the symbols for some key fields (e.g. reservation number)
* will be valuated to enable
* the valuation of other derived symbols (e.g. material quantity)
  LOOP AT lt_resbd INTO ls_resbd.
    l_lines_comp = sy-tabix.
```

```
* Get query symbols
    lo_symbol = query->get_symbols( l_lines_comp ).

* Evaluate reservation keys
    ls_symbol-symbol = 'RESERVATION'.
    lo_symbol->set_value_from_numc( symbol = ls_symbol
                                    data   = ls_resbd-rsnum
                                    domain = '/DDIC/RSNUM' ).
    ls_symbol-symbol = 'RESERVATION_ITEM'.
    lo_symbol->set_value_from_numc( symbol = ls_symbol
                                    data   = ls_resbd-rspos
                                    domain = '/DDIC/RSPOS' ).
    ls_symbol-symbol = 'RESERVATION_ITEM_TYPE'.
    lo_symbol->set_value_from_string( symbol = ls_symbol
                                      data   = ls_resbd-rsart
                                      domain = '/DDIC/RSART' ).

* From here on you can pre-valuate often-used symbols for
* performance improvement
* Domain names can be looked up in the attributes of
* top class CL_COCR_CMX_TOP
* Example: Material number
*        ls_symbol-symbol = 'MATERIAL'.
*        lo_symbol->set_value_from_string( symbol = ls_symbol
*                                          data   = ls_resbd-matnr
*                                          domain = '/DDIC/MATNR' ).

  ENDLOOP.

endmethod.
```

Related Content

For more information, visit the Manufacturing homepage.

Copyright