How to Use a Decision Table as a Custom Value List

Applies to:

SAP Decision Service Management 1.0 (or **BRFPlus** on SAP NetWeaver 7.02 or above)

Summary

When using business rules to provide more flexible, adaptable, and provable replacements for traditional Z tables, Decision Tables provide an easy way to hold valid value lists that change over time. We demonstrate how to use a Decision Table as a valid value list for another rules Data Object, i.e. the business rules rough equivalent to a ABAP Data Dictionary foreign key.

Author: Jocelyn Dart

Company: SAP Australia

Created on: 10 February 2015

Author Bio

Jocelyn Dart is a Platinum Consultant has worked for SAP Australia for over 20 years and worked with over 70 organizations in both the Public and Private Sectors. For the last 2-3 years she been helping customers implement business rules, and has presented at conferences and run workshops on SAP Decision Service Management and BRFPlus.

Table of Contents

Pre-Requisites

- SAP Decision Service Management 1.0, or BRFPlus on SAP NetWeaver 7.02 or above
    - Note: This example was originally worked on a SAP Decision Service Management 1.0 SP03 (SAP NetWeaver 7.40 SP09) system. However the same principles can be used on earlier BRFPlus releases, although some code adjustments may possibly be needed on earlier releases.
- Some ABAP development skills
- A Rules Application created in the BRFPlus workbench of SAP Decision Service Management or SAP NetWeaver containing:
    - A rules Data Object of type Element which will be used as the target of the value list
    - Decision Table with at least 2 columns that is the source of the value list:
        - One column will hold the list of valid values
        - The other column will hold descriptions of the values

For simplicity, the column of valid values is assumed to be entered into the Decision Table cells using only the Direct Input option. It is technically possible to extend the technique to cover expressions that are constants, and ranges that use Equals values, but this will not be shown in this document.

The descriptions may be of any data type, and entered into the Decision Table cells in any format (direct input, ranges, or expressions).

Note: The value and description columns may be any columns of the decision table. The decision table may also hold other columns (not used in the value list) alongside the value and description columns.

Create a Value List from a Decision Table

The steps included in this document assume that the Rules Application does not currently have an Exit Class associated with it, and that the Exit Class will be created from scratch.

However it is, of course, equally possible to extend an existing Exit Class to add custom defined value lists.

In the example a column Reference Code of one Decision Table Document Type Preferences will use a value list sourced from another Decision Table Valid Reference Codes.

Such a reference code could be used as part of an overarching decision service (aka BRFPlus function), where the reference code is used to broadly categorize documents, e.g. for analytics or security authorization purposes.

Note: The overarching decision service is not shown and is not relevant to this technique.

Target Decision Table: *Document Type Preferences*

Columns: *Document Type, Maximum Size, Document Reference Code*

Document Reference Code is the target element that will be linked to the value list. The screenshots below shows the appearance of the decision table before the value list has been applied. The user has to manually enter the value of Document Reference Code and does not receive any value help during field entry.

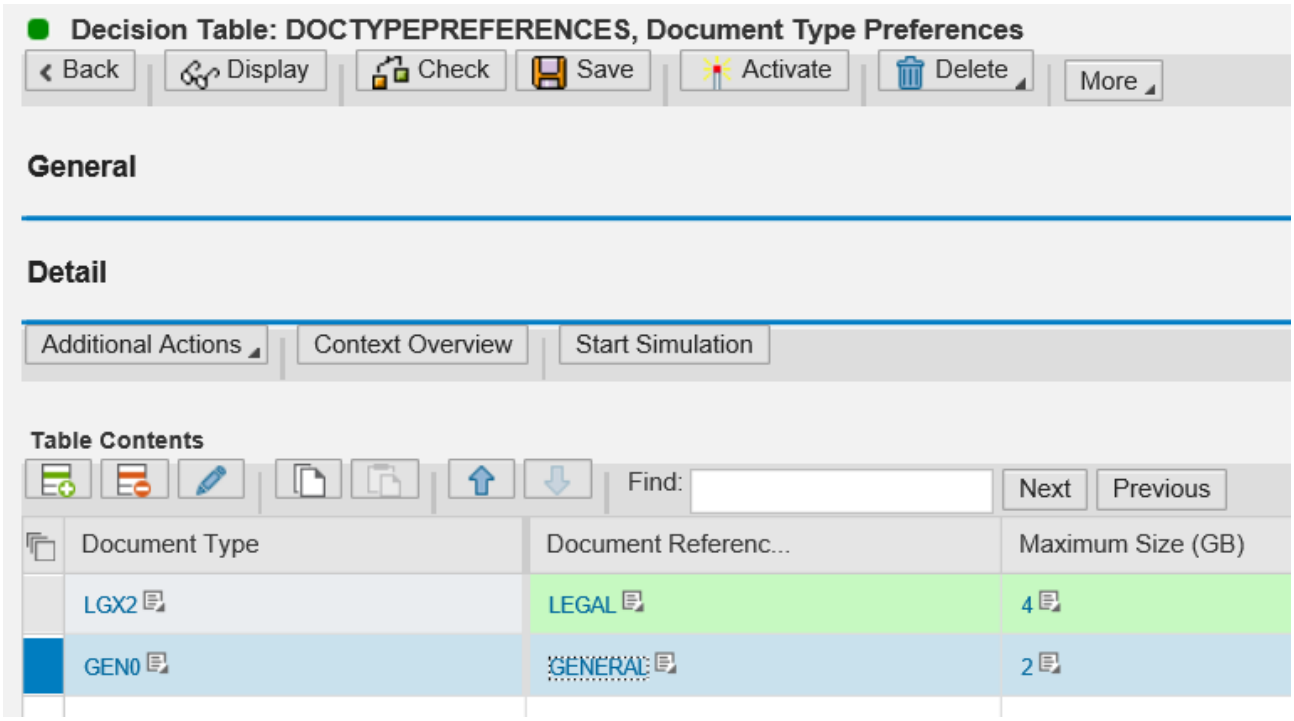


Figure 1 - Document Type Preferences decision table BEFORE value list is created

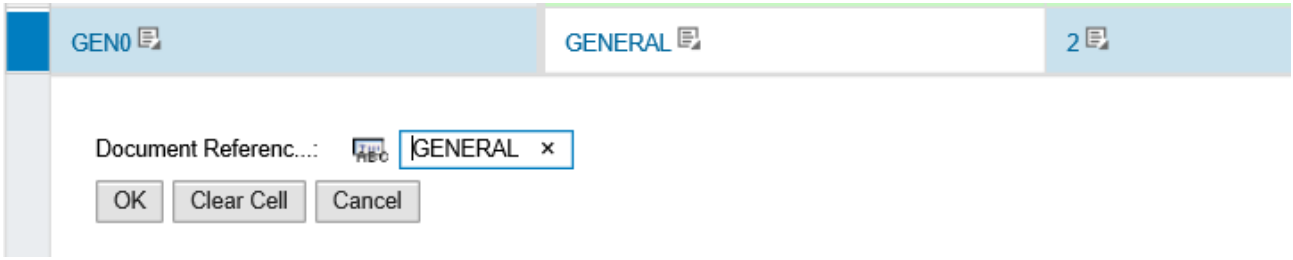


Figure 2 - Entry into the Document Reference Code is manual, no value help is provided

Source Decision Table: *Valid Reference Codes*

Columns: *Reference Code, Reference Description*

This decision table holds the list of valid values and their descriptions that will be used to create the value list.

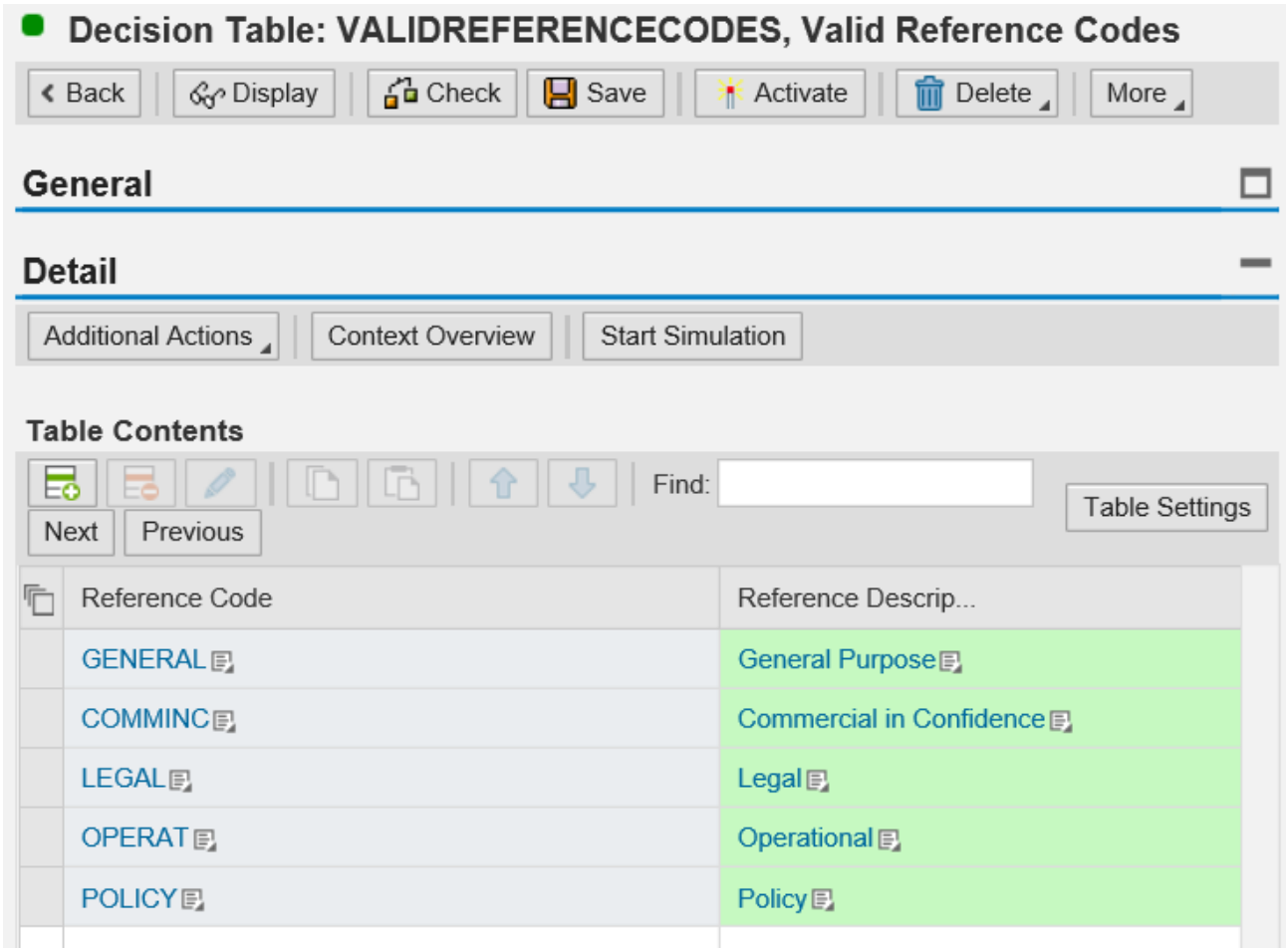


Figure 3 - Source Decision Table that holds the list of valid values and their descriptions

In preparation for building the value list, gather the technical IDs from the General section of:

1. The target Data Object of type Element, to which the value list will be assigned
    - In the example: *Document Reference Code*
2. The source Decision Table that holds the valid values and their matching descriptions
    - In the example: *Valid Reference Codes*
3. The Data Objects of type Element assigned as columns of the source Decision Table, which will provide the valid values and their matching descriptions.
    - In the example: *Reference Code, Reference Description*

Note: The value and description columns may be any columns of the decision table. However… if you use a condition column then care must be taken in converting ranges and expressions to selectable values. For a simple example use result columns with direct input values.

Create a Custom Defined Value List

The principles and options for providing value lists in DSM and BRFPlus is covered in greater detail elsewhere in Customer-Defined Value Lists in SAP NetWeaver Decision Management.

This document describes how to use the Exit Class option to create a value list filled from the content of a Decision Table.

To keep the code manageable and pragmatic, the example is limited to showing how the following types of content assigned to the source value element, may be used to fill a value list:

- Direct value Input
- A single “Equals” range compared to Direct Value Input
- A single “Equals” range compared to a Constant expression

Typically direct value input is used for action columns, and a range is used for a condition column.

Note: The value list will act as a value help, i.e. it will **assist** the user to enter correct values in the target element. Unlike a data dictionary foreign key, the existence of a value help does NOT necessarily restrict the entry of other values, not listed in the Source Decision Table, into the target element.

Create a Rules Application Exit Class

Create a ABAP Class that supports interface IF_FDT_APPLICATION_SETTINGS.

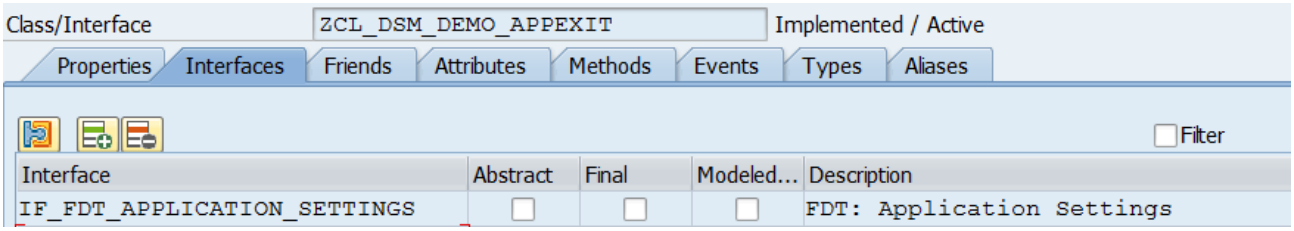


Figure 4 - Interface IF_FDT_APPLICATION_SETTINGS makes the class a Rules Application Exit Class

Set the Exit Class to provision custom Value Lists

Add the CLASS_CONSTRUCTOR method to the ABAP Class.

Implement the CLASS_CONSTRUCTOR method with the following code. This will indicate to the Rules Application that the exit class provides custom value lists:

```
if_fdt_application_settings~gv_get_element_values = abap_true.
```

Link the Source Value List to the Target Data Object

Implement the method IF_FDT_APPLICATION_SETTINGS~GET_ELEMENT_VALUES. In this method the target Data Object of type Element is linked to the custom-defined value list.

```
METHOD if_fdt_application_settings~get_element_values.

  CONSTANTS:
    lc_elementid   TYPE if_fdt_types=>id
       VALUE '005056AC35D91EE4ACA3C55A1AA354E6',"Target Element
    lc_dectableid  TYPE if_fdt_types=>id
       VALUE '005056AC35D91EE4AC9F32BABB880D99',"Source Decision Table
    lc_valuecol_id TYPE if_fdt_types=>id
       VALUE '005056AC35D91EE4ACA3AD3702A2B4C7',"Source Value Column
    lc_desccol_id  TYPE if_fdt_types=>id
       VALUE '005056AC35D91EE4ACA3AFF72EDA54C7'."Source Description Column
```

```
    DATA:
      lv_timestamp TYPE if_fdt_types=>timestamp.

    CLEAR: ev_no_checklist, ev_applicable, et_value.

* Only get values for this element
    CHECK iv_id EQ lc_elementid.

* Get the current timestamp to default to current values of the Decision Table
    IF iv_timestamp IS INITIAL.
      GET TIME STAMP FIELD lv_timestamp.
    ELSE.
      lv_timestamp = iv_timestamp.
    ENDIF.

    et_value =  convert_dt_to_valuelist(
      EXPORTING
        iv_timestamp = lv_timestamp
        iv_dectable_id = lc_dectableid
        iv_valuecol_id = lc_valuecol_id
        iv_textcol_id  = lc_desccol_id ).

* Confirm the checklist is ok to use
    IF et_value IS NOT INITIAL.
      ev_no_checklist = abap_false.
      ev_applicable = abap_true.
    ENDIF.


  ENDMETHOD.
```

Convert Decision Table content to fill a custom Value List

Implement a Static, Private method CONVERT_DT_TO_VALUELIST with signature:

IV_TIMESTAMP	IMPORTING OPTIONAL	TYPE IF_FDT_TYPES=>TIMESTAMP
IV_DECTABLE_ID	IMPORTING MANDATORY	TYPE IF_FDT_TYPES=>ID
IV_VALUECOL_ID	IMPORTING MANDATORY	TYPE IF_FDT_TYPES=>ID
IV_TEXTCOL_ID	IMPORTING MANDATORY	TYPE IF_FDT_TYPES=>ID
ET_VALUE	RETURNING	TYPE IF_FDT_APPLICATION_SETTINGS=>T_VALUE

```
  METHOD convert_dt_to_valuelist.
    DATA:
    lr_admin_data  TYPE REF TO if_fdt_admin_data,
    lr_dectable    TYPE REF TO if_fdt_decision_table,
    lr_constant    TYPE REF TO if_fdt_constant,

    lt_tabledata   TYPE if_fdt_decision_table=>ts_table_data,
    lt_columns     TYPE if_fdt_decision_table=>ts_column,

    ls_text        TYPE if_fdt_types=>element_text,
    ls_value       TYPE if_fdt_application_settings=>s_value,

    lv_colno_value TYPE int4,
```

```
    lv_colno_text  TYPE int4.

    FIELD-SYMBOLS:
      <fs_tabledata>  TYPE if_fdt_decision_table=>s_table_data,
      <fs_tabledata2> TYPE if_fdt_decision_table=>s_table_data,
      <fs_valuedata>  TYPE any,
      <fs_textdata>   TYPE any,
      <fs_column>     TYPE if_fdt_decision_table=>s_column,
      <fs_range>      TYPE if_fdt_range=>s_range.


* Get the reference to the decision table
    cl_fdt_factory=>get_instance_generic(
      EXPORTING iv_id = iv_dectable_id
      IMPORTING eo_instance = lr_admin_data ).

    lr_dectable ?= lr_admin_data.
    CHECK lr_dectable IS BOUND.

* Read all the decision table data
    TRY.
        lr_dectable->get_table_data(
          EXPORTING
            iv_timestamp = iv_timestamp
          IMPORTING
            ets_data     = lt_tabledata ).
      CATCH cx_fdt_input .
* Exit if we couldn't read the source decision table
        EXIT.
    ENDTRY.
* Check the source decision table has at least one value in it
    CHECK lt_tabledata IS NOT INITIAL.
* Read the columns of the source decision table
    TRY.
        CALL METHOD lr_dectable->get_columns
          EXPORTING
            iv_timestamp = iv_timestamp
          IMPORTING
            ets_column   = lt_columns.
      CATCH cx_fdt_input .
        EXIT.
    ENDTRY.

* Find the column number of the source value column
    READ TABLE lt_columns ASSIGNING <fs_column>
      WITH KEY object_id = iv_valuecol_id.
    IF sy-subrc EQ 0.
      lv_colno_value = <fs_column>-col_no.
    ENDIF.

* Find the column number of the source description column
    READ TABLE lt_columns ASSIGNING <fs_column>
      WITH KEY object_id = iv_textcol_id.
    IF sy-subrc EQ 0.
      lv_colno_text = <fs_column>-col_no.
    ENDIF.
```

```
* Just in case… check we found both columns
    CHECK lv_colno_value IS NOT INITIAL.
    CHECK lv_colno_text IS NOT INITIAL.

* Loop over the value column
    LOOP AT lt_tabledata ASSIGNING <fs_tabledata>
      WHERE col_no = lv_colno_value.

      CLEAR: ls_value.

      IF <fs_tabledata>-expression_id IS NOT INITIAL.
* Hmmmm... that's a little too hard..
        CONTINUE.
* Instead why don't we stick to:
* - Direct value input; or
* - A single inclusive equals range compared to a direct value; or
* - A single inclusive equals range compared to a constant



      ELSEIF <fs_tabledata>-ts_range IS NOT INITIAL.
* - A single inclusive equals range compared to a direct value; or

        CHECK lines( <fs_tabledata>-ts_range ) = 1.
        ASSIGN <fs_tabledata>-ts_range[ position = 1 ] TO <fs_range>.
        CHECK <fs_range>-sign = 'I'
          AND <fs_range>-option = 'EQ'.

        IF <fs_range>-r_low_value IS NOT INITIAL.
          ASSIGN <fs_range>-r_low_value->* TO <fs_valuedata>.
          IF <fs_valuedata> IS ASSIGNED.
            TRY.
                ls_value-value = <fs_valuedata>.
              CATCH cx_sy_conversion_error.
* Just in case we have a problem
                CONTINUE.
            ENDTRY.
          ENDIF.


        ELSEIF <fs_range>-low IS NOT INITIAL.
* - A single inclusive equals range compared to a constant

* Get the expression - so long as it is a constant
          cl_fdt_factory=>get_instance_generic(
            EXPORTING iv_id = <fs_range>-low
            IMPORTING eo_instance = lr_admin_data ).

          TRY.
              lr_constant ?= lr_admin_data.
              IF lr_constant IS BOUND.
                TRY.
                    lr_constant->get_constant_value(
                      EXPORTING
                        iv_timestamp = iv_timestamp
                      IMPORTING
                        ea_value = ls_value-value )
```

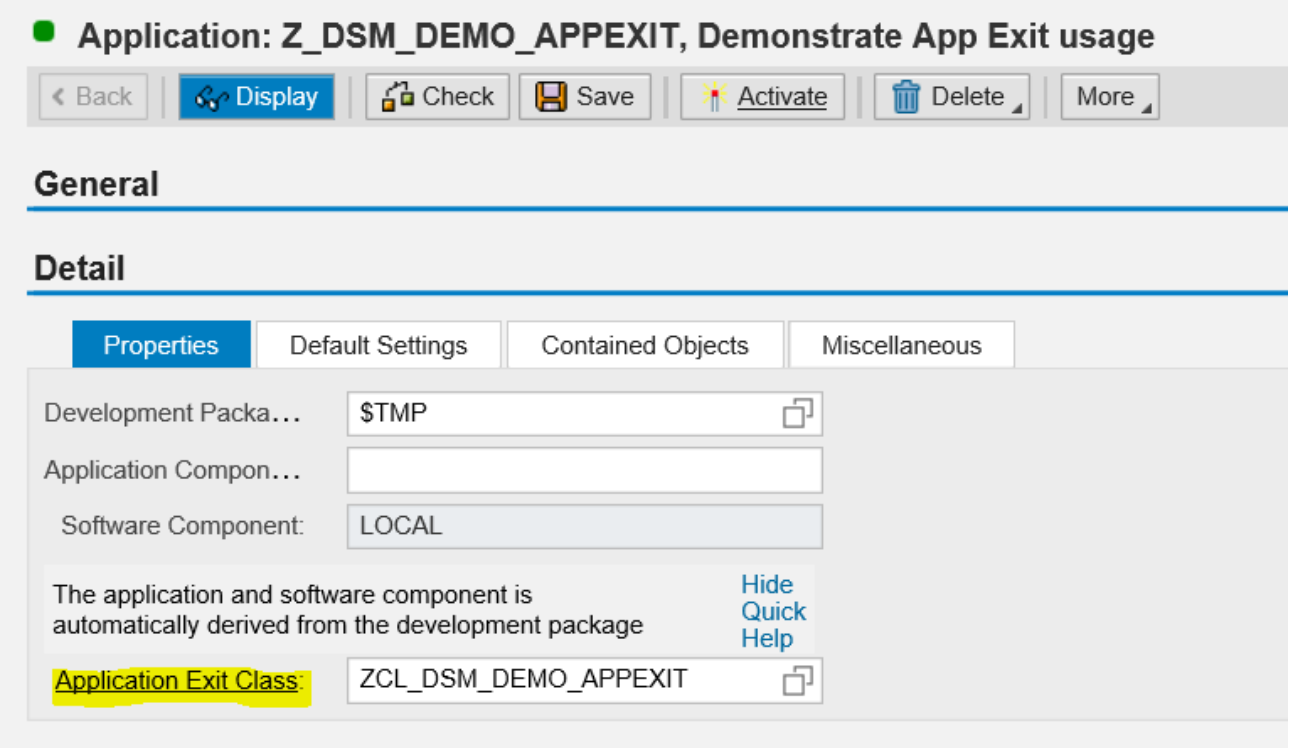
```
                    .
                  CATCH cx_fdt_input .
                    CONTINUE.
                ENDTRY.

              ELSE.
                CONTINUE.
              ENDIF.
            CATCH cx_sy_conversion_error.
* Just in case we have a problem
              CONTINUE.
          ENDTRY.
        ELSE.
          CONTINUE.
        ENDIF.

      ELSE.
* - Direct value input

* Get the Direct Input of the value cells in the source decision table
        ASSIGN <fs_tabledata>-r_value->* TO <fs_valuedata>.
        IF <fs_valuedata> IS ASSIGNED.
          TRY.
              ls_value-value = <fs_valuedata>.
            CATCH cx_sy_conversion_error.
* Just in case we have a problem
              CONTINUE.
          ENDTRY.
        ENDIF.
      ENDIF.

* Get the description cell that matches the value cell
      READ TABLE lt_tabledata
        ASSIGNING <fs_tabledata2>
        WITH KEY
          col_no = lv_colno_text
          row_no = <fs_tabledata>-row_no.

      IF sy-subrc EQ 0.

        IF <fs_tabledata2>-expression_id IS NOT INITIAL.
* Get the string format of the expression
          cl_fdt_factory=>get_instance_generic(
            EXPORTING iv_id = <fs_tabledata2>-expression_id
            IMPORTING eo_instance = lr_admin_data ).

          lr_admin_data->to_string(
            EXPORTING
              iv_timestamp  = iv_timestamp
              iv_max_length = if_fdt_constants=>gc_tostring_max_length
              iv_mode       = if_fdt_constants=>gc_tostring_mode_specific
            RECEIVING
              rv_string     = ls_value-text )
              .


        ELSEIF <fs_tabledata2>-ts_range IS NOT INITIAL.
```

```
* Getting the string format of a range is also possible but let's keep it simple
          ls_value-text = |Range of values|.


        ELSE.
* Read direct value input of the description cell
          ASSIGN <fs_tabledata2>-r_value->* TO <fs_textdata>.
          IF <fs_textdata> IS ASSIGNED.
            TRY.
                ls_value-text = <fs_textdata>.
              CATCH cx_sy_conversion_error.
                ls_value-text = |Unexpected data in text column?|.
            ENDTRY.
          ELSE.
            IF ls_value-value IS NOT INITIAL.
              ls_value-text = |*** Missing Description for value | && ls_value-value.
            ELSE.
              ls_value-text =
|*** Missing Description for row number | && <fs_tabledata>-row_no.
            ENDIF.
          ENDIF.
        ENDIF.
      ENDIF.
* If all else fails put something…
      IF ls_value-text IS INITIAL.
        ls_value-text =
|*** Missing Description for row number | && <fs_tabledata>-row_no.
      ENDIF.
      APPEND ls_value TO et_value.
      UNASSIGN: <fs_valuedata>, <fs_textdata>.
    ENDLOOP.

  ENDMETHOD.
```

Assign the Exit Class to the Rules Application

Activate the Exit Class.

Assign the Exit class to the Properties of the Rules Application.



Activate the rules application.

Testing the Value List

Edit the target element in the target decision table. Notice that when editing the value a value help (dropdown) button appears. Use the button to view the value list. Select a value from the value list and check it has been assigned to the target cell.

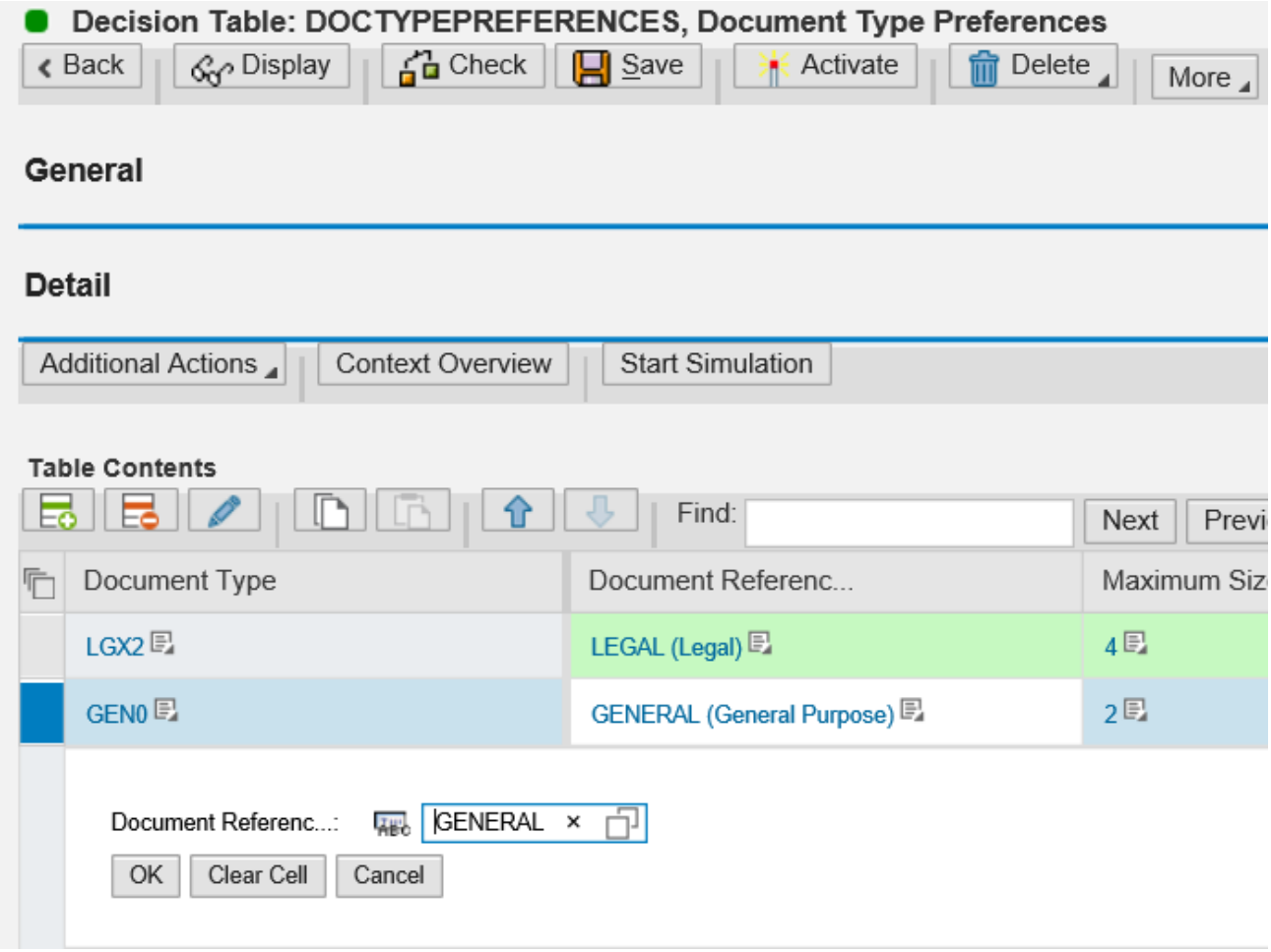


Figure 5 - In edit mode, target element shows a value help button

When the dropdown button is pressed the search result list shows the values from the source Decision Table.

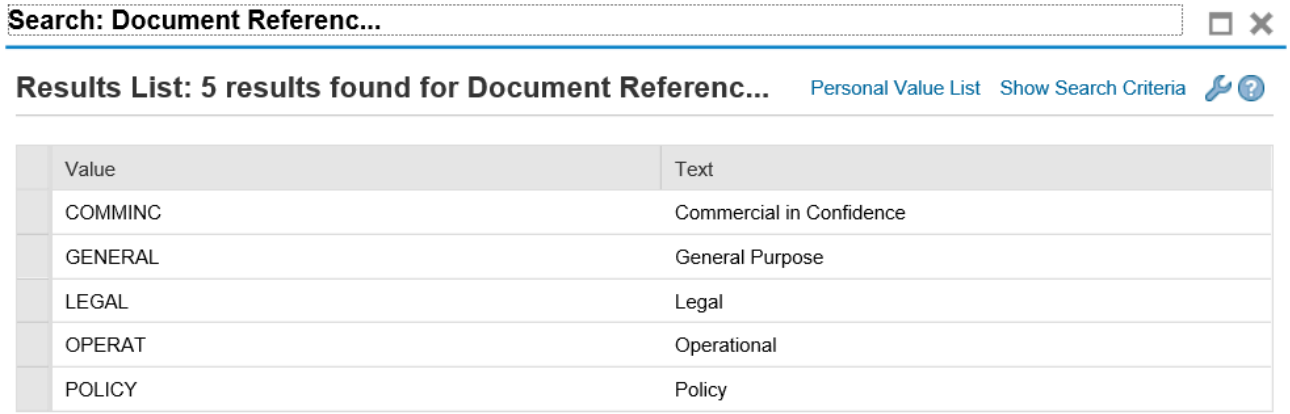


Figure 6 - On pressing the value help button, the value list appears

In the screenshot below the final result is shown. The value of the column matches the value list, e.g. LEGAL. Next to the value the description is shown in parentheses, e.g. (Legal).

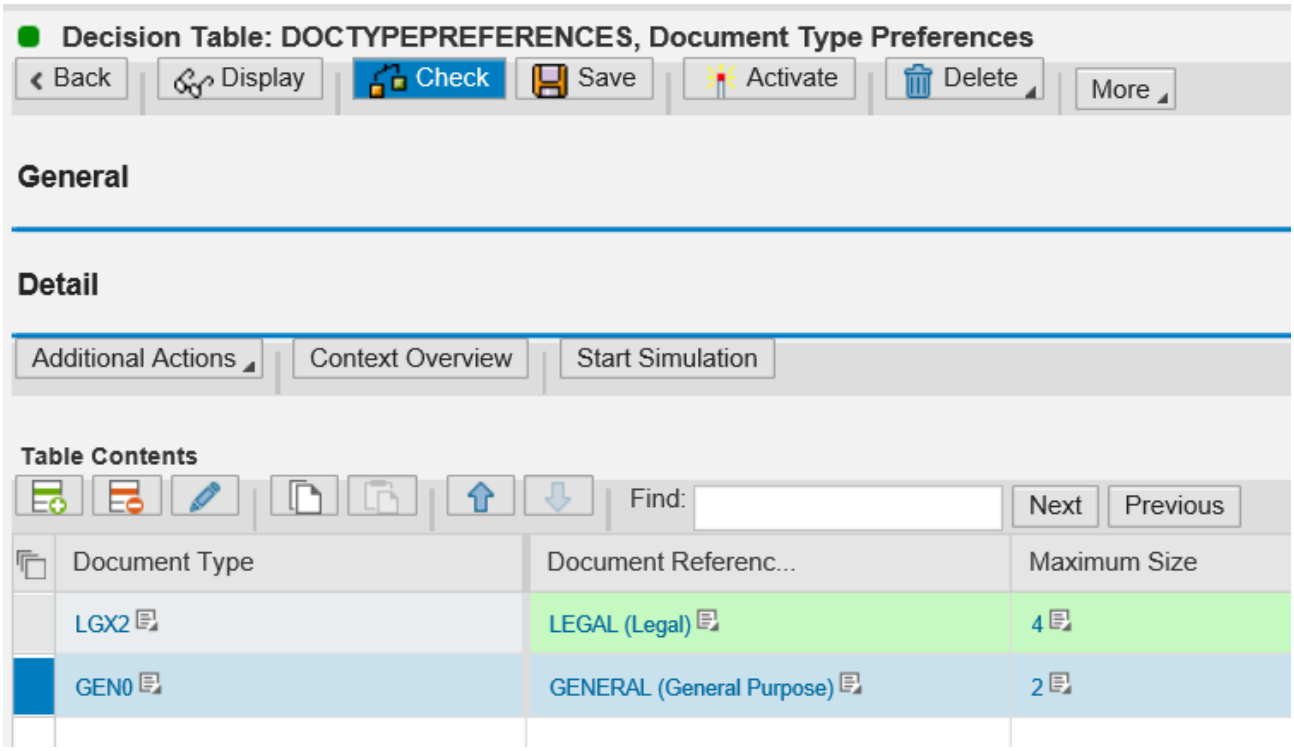


Figure 7 - Once selected, the target element shows the assigned value (and its matching description)

Debugging and Resolving Issues

Any problems with resolving the creation of your value list are usually easily resolved by debugging your code via CL_FDT_APPLICATION_EXIT=> GET_ELEMENT_VALUES.

To do this

1. Set a breakpoint in your own application exit method IF_FDT_APPLICATION_SETTINGS=>GET_ELEMENT_VALUES
2. In transaction SE24, enter the class CL_FDT_APPLICATION_EXIT and press the Test (Execute in Test Environment) button
3. Select the method GET_ELEMENT_VALUES
4. Put the ID of your target element Data Object into the parameter IV_ID
5. Press Execute
6. And start debugging!

Taking the technique further

This technique could be extended to provide the same capabilities for multiple target elements with their appropriate source decision tables; for example by using custom attributes to hold the IDs of the target element, source decision table, value column and description column.

Related Content

Customer-Defined Value Lists in SAP NetWeaver Decision Service Management

How to Kill Custom Code and Z-Tables

BRFplus Application Exits

Custom-Defined Attributes for BRFplus with SAP NetWeaver Decision Service Management

Copyright