BRFplus Simplification

Applies to:

Business Rule Framework plus (BRFplus) shipped with SAP NetWeaver 7.0 Enhancement Package 2. xBusiness Rules Management

Summary

BRFplus Simplification refers to an approach to reduce the number of object instances needed for rule modeling (less constant and range instances) as well as to simplifying the data model. The result is a significant reduction of the memory space needed for several expressions.

Authors: Carsten Ziegler, Michael Baer

Company: SAP AG

Created on: 7th July 2009

Authors Bio

Carsten Ziegler is the Architect and Project Manager of Business Rules Framework plus. He joined SAP in 2000. Since then he has been working in various projects as a developer, development architect and project lead.

Michael Bär is a Developer in the Business Rules Framework plus team. Since joining SAP in 2008, he has worked in the BRFplus team.

Table of Contents

Explanation of the Simplification

The term ‘simplification’ is a backend term from the BRFplus rules engine. In fact, it doesn’t simplify anything in the User Interface but aims at a reduction of object instances (less constant and range instances) as well as a simplification of the data model. The result is a significant reduction of the memory space needed for several expressions.

This brings many benefits. Not only the memory footprint has been reduced, but also the number of objects that belong to a simplified application, thus reducing the transport load of an application. All expressions created with the BRFplus UI will be created according to the simplified approach. When implementing reports or similar objects in the backend, the user now has the choice between the simplified and the old approach. It is recommended to use the simplified approach. Creation of a constant or range instance is still recommended when reuse is intended.

While simplification brings a lot of advantages, there is, however, a drawback for existing coding. Although the changes are not incompatible you must know about them to be prepared in your code. You will find an explanation of possible adjustments at the end of this document.

Simplified Components

Mainly two different types of components have been simplified:

Constant

Constants are used in many expression types. After the simplification, the constant changed to a direct value. So now the expression types don’t store the constant object any more, instead they store the direct value in a specific database table. Of course, the constant expression itself cannot be simplified.

Range

Like constants, ranges are used in many expression types. Here, the ranges are changed from explicit ranges (for which you needed an object of its own) to implicit ranges for which no object is needed. The necessary range table is also stored in a specific database table.

Expression Types That Have Been Simplified

Boolean

In most cases, the operands of a boolean expression are ranges. Now you have two possibilities:

- You can create an expression and use its ID as the operand, just like before.
- Or you can use an implicit range as the operand. The buffer is changed in the following way:

```
BEGIN OF s_buffer,
      operation         TYPE if_fdt_boolean=>operation,
      invert            TYPE abap_bool,      "invert the complete result
      first_operand_id  TYPE if_fdt_types=>id,
      s_first_range     TYPE cl_fdt_expr_sv=>s_param_range,
      first_invert      TYPE abap_bool,
      second_operand_id TYPE if_fdt_types=>id,
      s_second_range    TYPE cl_fdt_expr_sv=>s_param_range,
      second_invert     TYPE abap_bool,
      third_operand_id  TYPE if_fdt_types=>id,
      s_third_range     TYPE cl_fdt_expr_sv=>s_param_range,
      third_invert      TYPE abap_bool,
      ts_andor_operand  TYPE ts_andor_operand,
      ts_token          TYPE ts_token,
END OF s_buffer .
```

The red lines indicate the parts of the buffer where the implicit ranges are stored. The name of the according database table is FDT_EXPR_1304. All implicit ranges are stored in this table.

Case

A case expression is built using the following scheme:

```
CASE case_parameter
  WHEN test_parameter
        THEN return_parameter
WHEN OTHERS
        THEN other_parameter
```

The test_parameter, return_parameter and other_parameter are simplified with direct value usage. The buffer is changed in the following way:

```
BEGIN OF s_buffer,
     case_parameter   TYPE if_fdt_types=>id,
     other_parameter  TYPE if_fdt_types=>id,
     s_other_value    TYPE cl_fdt_expr_sv=>s_value,
     ts_when          TYPE ts_when,
     case_sensitivity TYPE abap_bool,
     executes_action  TYPE abap_bool,
     select1202       TYPE abap_bool, "needed for SAVE_BUFFER_DB
END OF s_buffer .
```

The return_parameter and the test_parameter are part of the ts_when table. Its structure is changed in the following way:

```
BEGIN OF s_when,
      position          TYPE fdt_expr_1202s-when_position,
      test_parameter    TYPE if_fdt_types=>id,
      s_test_value      TYPE cl_fdt_expr_sv=>s_value,
      return_parameter  TYPE if_fdt_types=>id,
      s_return_value    TYPE cl_fdt_expr_sv=>s_value,
END OF s_when .
```

The new lines are marked in red. The other_value is stored in the database table FDT_EXPR_1200, the test_value and return_value are stored in database table FDT_EXPR_1202.

Decision Table

A decision table expression has the following scheme:

Condition Column1	Condition Column2	Result Column1	Result Column2
Condition Cell 11	Condition Cell 12	Result Cell 13	Result Cell 14
Condition Cell 21	Condition Cell 22	Result Cell 23	Result Cell 24
Condition Cell 31	Condition Cell 32	Result Cell 33	Result Cell 34

The entries in the condition cells return a boolean result, and in most cases the conditions are ranges. Now implicit ranges may be used in condition cells instead of creating a range for it. Result cells are most likely constant values. The user has now the possibility to use a direct value instead of creating a constant for the result cell. If a user wants to use a nested expression or an action for the result cell, he cannot use the direct value input and has to do it like it was before the simplification. The same applies to non-range condition cells.

Two new elements have been added to the table data structure to implement simplification.

```
BEGIN OF s_table_data,
      col_no        TYPE int4,
      row_no        TYPE int4,
      expression_id TYPE if_fdt_types=>id,
      s_value       TYPE cl_fdt_expr_sv=>s_value,
      ts_range      TYPE ts_range,
END OF s_table_data .
```

The new lines are marked in red and indicate the variables, where the simplified data is stored. The according database tables are FDT_EXPR_0242 (value) and FDT_EXPR_0244 (ranges). Please see a detailed example for a simplified decision table in a later chapter.

Decision Tree

A decision tree expression consists of condition nodes and result nodes. Conditions are part of condition nodes. The condition returns a result of type boolean, while the conditions are usually ranges. Result nodes usually consist of a constant expression. Now it is possible to use implicit ranges for the condition nodes' condition and direct values in the result nodes.

The new elements have been added to the node property structure:

```
BEGIN OF s_node_prop,
  node           TYPE if_fdt_decision_tree~node_id,
  parent         TYPE if_fdt_decision_tree~s_tree_structure-parent,
  leaf           TYPE abap_bool,
  expression_id  TYPE if_fdt_types=>id,
  s_param_range  TYPE cl_Fdt_expr_sv=>s_param_range,
  s_result_value TYPE cl_Fdt_expr_sv=>s_value,
END OF s_node_prop .
```

The new lines are indicated in red. The corresponding database tables are FDT_EXPR_0230 (value) and FDT_EXPR_0231 (ranges).

Loop

Independent of the mode, a loop expression has always a table where all rules for one loop iteration are described. The user can define continue and exit conditions in this rule table. These conditions are usually ranges. Now it is possible to use implicit ranges instead of creating range objects. Some loop modes have also a condition, which is usually a range. Now it is also possible to use an implicit range for the condition.

The new elements have been added to the rules table:

```
BEGIN OF s_rule_cl,
       position             TYPE fdt_inc_expr_9002_key-pos,
       rule_id              TYPE if_fdt_types=>id,
  exit_condition_id     TYPE if_fdt_types=>id,
       exit_cond_range      TYPE cl_fdt_expr_sv=>s_param_range,
  continue_condition_id TYPE if_fdt_types=>id
  continue_cond_range   TYPE cl_fdt_expr_sv=>s_param_range,
END OF s_rule_cl.
```

The implicit range for the condition has been added to the buffer:

```
BEGIN OF s_buffer,
  loop_mode          TYPE if_fdt_loop=>loop_mode,
  number_of_repeats  TYPE if_fdt_loop=>number_of_repeats,
  loop_condition     TYPE if_fdt_types=>id,
  loop_cond_range    TYPE cl_fdt_expr_sv=>s_param_range,
  for_each_table     TYPE if_fdt_types=>id,
  ts_fe_cond         TYPE ts_fe_cond,
  ts_rule            TYPE ts_rule_cl,
END OF s_buffer .
```

The new lines are indicated in red. The corresponding database table is FDT_EXPR_9003.

Range

A range expression has always a low parameter and, depending on the chosen operation, a high parameter. These parameters are usually constants, so that a usage of direct value makes sense.

The new elements have been added to the range table and are indicated below in red.

```
BEGIN OF s_range,
          position     TYPE fdt_seqnr,
          sign         TYPE ddsign,
          option       TYPE ddoption,
          low          TYPE if_fdt_types=>id,
          s_low_value  TYPE s_value,
          high         TYPE if_fdt_types=>id,
          s_high_value TYPE s_value,
END OF s_range .
```

The according database table is FDT_EXPR_1104.

Search Tree

The search tree expression is similar to the decision tree. Here, each node can have a condition and a result. As the conditions are mostly ranges and the results are constants, it makes sense to use implicit ranges or direct values.

The new elements have been added to the node property table and are marked below in red.

```
BEGIN OF s_node_prop,
    node           TYPE if_fdt_search_tree=>node_id,
    parent         TYPE if_fdt_search_tree~s_tree_structure-parent,
    seqnr          TYPE if_fdt_search_tree~s_tree_structure-seqnr,
    condition_id   TYPE if_fdt_types=>id,
    s_param_range  TYPE cl_fdt_expr_sv=>s_param_range,
    result_id      TYPE if_fdt_types=>id,
    s_result_value TYPE cl_fdt_expr_sv=>s_value,
 END OF s_node_prop .
```

The according database tables are FDT_EXPR_0210 (value) and FDT_EXPR_0212 (range).

Rule

A rule expression can have a condition. If this condition is evaluated to true, then the true branch, otherwise the false branch of the rule is executed. This parameter is usually a range, so that a usage of an implicit range makes sense.

The new element has been added to the buffer and is indicated below in red.

```
 BEGIN OF s_buffer,
      condition          TYPE if_fdt_types=>id,
      cond_range         TYPE cl_fdt_expr_sv=>s_param_range,
      true_action        TYPE if_fdt_types=>id,
      false_action       TYPE if_fdt_types=>id,
      ts_action_extended TYPE ts_action_extended,
    END OF s_buffer .
```

The according database table is FDT_EXPR_3002.

Ruleset

A ruleset can have a condition. If this condition is evaluated to true, then the ruleset is evaluated, otherwise not. Each rule of the ruleset can also have a precondition. If evaluated to true, the rule will be processed, otherwise it will be skipped. These parameters are usually ranges, so that the usage of implicit ranges makes sense.

The new elements have been added to the buffer. In case of the rule preconditions, the new elements were added to the rules. The changes are indicated below in red.

```
BEGIN OF s_buffer,
    function_id  TYPE if_fdt_types=>id,
    condition_id TYPE if_fdt_types=>id,
    cond_range   TYPE cl_fdt_expr_sv=>s_param_range,
    switch       TYPE if_fdt_ruleset=>switch,
    priority     TYPE if_fdt_ruleset=>priority,
    ts_rule      TYPE ts_rule_cl,
    ts_variable  TYPE if_fdt_ruleset=>ts_variable,
    ts_init_expr TYPE if_fdt_ruleset=>ts_init_expr,
  END OF s_buffer .

BEGIN OF s_rule_cl,
      position      TYPE if_fdt_ruleset=>position,
      function_id   TYPE if_fdt_types=>id,
      condition_id  TYPE if_fdt_types=>id,
      cond_range    TYPE cl_fdt_expr_sv=>s_param_range,
      valid_from    TYPE if_fdt_types=>timestamp,
      valid_to      TYPE if_fdt_types=>timestamp,
      rule_id       TYPE if_fdt_types=>id,
      switch        TYPE if_fdt_ruleset=>switch,
      exit_ruleset  TYPE abap_bool,
      restart_option TYPE restart_option,
    END OF s_rule_cl .
```

The according database table is FDT_RLST_0001.

Overview

Expression	Part	Simplification Type
Boolean	First Operand	Implicit Range
Boolean	Second Operand	Implicit Range
Boolean	Third Operand	Implicit Range
Case	Other Parameter	Direct Value
Case	Test Parameter	Direct Value
Case	Return Parameter	Direct Value
Decision Table	Condition Cell	Implicit Range
Decision Table	Result Cell	Direct Value
Decision Tree	Condition Node	Implicit Range
Decision Tree	Result Node	Direct Value
Loop	Continue Condition	Implicit Range
Loop	Exit Condition	Implicit Range
Loop	Loop-Condition	Implicit Range
Range	Low Parameter	Direct Value
Range	High Parameter	Direct Value
Search Tree	Node Condition	Implicit Range
Search Tree	Node Result	Direct Value
Rule	Condition	Implicit Range
Ruleset	Condition	Implicit Range
Ruleset	Rule Precondition	Implicit Range

Detailed Example: Decision Table

This section shows how to create an implicit range for a decision table condition cell. In addition the creation of a direct value for a result cell is shown.

```
DATA: ls_table_data  TYPE if_fdt_decision_table=>s_table_data,
      lts_table_data TYPE if_fdt_decision_table=>ts_table_data,
      ls_range       TYPE if_fdt_decision_table=>s_range.
FIELD-SYMBOLS: <lv_value>         TYPE any.
* First example: Condition Cell => Implicit Range.
* The condition is in cell (1,1).
ls_table_data-row_no = 1.
ls_table_data-col_no = 1.
* Build up the range table, begin with position 1.
ls_range-position = 1.
* Choose the sign for the range Include / Exclude.
ls_range-sign     = if_fdt_range=>gc_sign_include.
* Choose the option for the range.
ls_range-option   = if_fdt_range=>gc_option_between.
* Create the data for the low value.
CREATE DATA ls_range-r_low_value TYPE if_fdt_types=>element_number.
ASSIGN ls_range-r_low_value->* TO <lv_value>.
<lv_value> = 1.
* Create the data for the high value.
CREATE DATA ls_range-r_high_value TYPE if_fdt_types=>element_number.
ASSIGN ls_range-r_high_value->* TO <lv_value>.
<lv_value> = 5.
* Insert the range in the range table.
INSERT ls_range INTO TABLE ls_table_data-ts_range.
* If you want to have more entries in the range table, just repeat the
upper procedure to create more entries and add them. Do not forget to add
the table data to the according table.
INSERT ls_table_data INTO TABLE lts_table_data.

* Second example: Result Cell => Direct Value.
* The result is in cell (1,2).
ls_table_data-row_no = 1.
ls_table_data-col_no = 2.
* Create the data for the result.
CREATE DATA ls_table_data-r_value TYPE if_fdt_types=>element_number.
ASSIGN ls_table_data-r_value->* TO <lv_value>.
<lv_value> = 10.
* That is all, don't forget to add the cell to the table data.
INSERT ls_table_data INTO TABLE lts_table_data.
```

CL_FDT_SIMPLIFIER

BRFplus offers a class CL_FDT_SIMPLIFIER which offers several services for simplification. You can simplify a single expression or all expressions of an application at once with the help of the methods SIMPLIFY_EXPRESSION and SIMPLIFY_APPLICATION_OBJECTS. An object can only be simplified if the following conditions are met:

- The sub-object (range, constant) is unnamed
- The only usage of the sub-object may be the one in the object
- The object must not have cross-application references

Impact of Simplification on Current Users Using a Decision Table

The simplified approach is used in several methods of class CL_FDT_CONVENIENCE. The methods are CREATE_SIMPLE_RANGE and FILL_DECISION_TABLE. In class CL_FDT_DECISION_TABLE itself, the two methods SET_TABLE_DATA and GET_TABLE_DATA are using the simplified approach. It is important to know that when you create an exit, report, or something similar, you can no longer rely on a filled EXPRESSION_ID in the table data (see the structure of the table data in a former chapter). Therefore, existing coding also has to be changed or it won't work in the future.

Demo Reports

In the package SFDT_DEMO_OBJECTS, several demo reports can be found. These reports explain the according expression type and are using the simplification technique. See the following list for all important demo reports.

Expression Type	Report Name
Boolean	FDT_DEMO_REPORT_BOOLEAN
Case	FDT_DEMO_REPORT_CASE
Decision Table	FDT_DEMO_REPORT_DECISION_TABLE
Decision Tree	FDT_DEMO_REPORT_DECISION_TREE
Loop	FDT_DEMO_REPORT_LOOP
Range	FDT_DEMO_REPORT_RANGE_NUM
Search Tree	FDT_DEMO_REPORT_SEARCH_TREE

Interface and type changes

This chapter gives an overview of all changes in the different interfaces. Also, all type changes are mentioned here. The new entries are all marked with red colour.

Boolean

1. Methods
   - GET_OPERANDS

```
methods GET_OPERANDS
   importing
     !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
   exporting
     !EV_FIRST_OPERAND_ID type IF_FDT_TYPES=>ID
     !ES_FIRST_OPERAND_RANGE type S_PARAM_RANGE
     !EV_FIRST_INVERT type ABAP_BOOL
     !EV_SECOND_OPERAND_ID type IF_FDT_TYPES=>ID
     !ES_SECOND_OPERAND_RANGE type S_PARAM_RANGE
     !EV_SECOND_INVERT type ABAP_BOOL
     !EV_THIRD_OPERAND_ID type IF_FDT_TYPES=>ID
     !ES_THIRD_OPERAND_RANGE type S_PARAM_RANGE
     !EV_THIRD_INVERT type ABAP_BOOL
     !ETS_ANDOR_OPERAND type TS_ANDOR_OPERAND
   raising
     CX_FDT_INPUT .
```

```
SET_OPERANDS
methods SET_OPERANDS
    importing
      !IV_FIRST_OPERAND_ID type IF_FDT_TYPES=>ID optional
      !IS_FIRST_OPERAND_RANGE type S_PARAM_RANGE optional
      !IV_FIRST_INVERT type ABAP_BOOL optional
      !IV_SECOND_OPERAND_ID type IF_FDT_TYPES=>ID optional
      !IS_SECOND_OPERAND_RANGE type S_PARAM_RANGE optional
      !IV_SECOND_INVERT type ABAP_BOOL optional
      !IV_THIRD_OPERAND_ID type IF_FDT_TYPES=>ID optional
      !IS_THIRD_OPERAND_RANGE type S_PARAM_RANGE optional
      !IV_THIRD_INVERT type ABAP_BOOL optional
      !ITS_ANDOR_OPERAND type TS_ANDOR_OPERAND optional
    raising
      CX_FDT_INPUT .

GET_USER_DEFINED_TOKEN
methods GET_USER_DEFINED_BOOLEAN
    importing
      !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
    exporting
      !EV_USER_DEFINED_BOOLEAN type USER_DEFINED_BOOLEAN
      !ET_USER_DEFINED_TOKEN type T_USER_DEFINED_TOKEN
      !EV_TOKEN_TABLE_ONLY type ABAP_BOOL
    raising
CX_FDT_INPUT .

SET_USER_DEFINED_BOOLEAN
methods SET_USER_DEFINED_BOOLEAN
   importing
    !IV_USER_DEFINED_BOOLEAN type USER_DEFINED_BOOLEAN optional
    !IT_USER_DEFINED_TOKEN type T_USER_DEFINED_TOKEN optional
   exporting
    !EV_SYNTAX_ERROR type ABAP_BOOL
   raising
    CX_FDT_INPUT .
```

2. Types

```
s_param_range (new type)
 BEGIN OF s_param_range,
      parameter_id TYPE if_fdt_types=>id,
      ts_range     TYPE if_fdt_range=>ts_range,
    END OF s_param_range .

s_user_defined_token (new type)
 BEGIN OF s_user_defined_token,
      token           TYPE fdt_boolean_token,
      operand_id      TYPE if_fdt_types=>id,
      s_operand_range TYPE s_param_range,
    END OF s_user_defined_token .
t_user_defined_token (new type)
t_user_defined_token TYPE STANDARD TABLE OF s_user_defined_token

s_andor_operand
   BEGIN OF s_andor_operand,
      position TYPE fdt_expr_1303-pos,
      id       TYPE if_fdt_types=>id,
      s_range  TYPE s_param_range,
```

```
      invert   TYPE abap_bool,
    END OF s_andor_operand .
```

Case

1. Methods
   - GET_OTHER_VALUE (new method)

```
methods GET_OTHER_VALUE
    importing
      !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
    returning
      value(RR_VALUE) type ref to DATA
    raising
      CX_FDT_INPUT .

SET_OTHER_VALUE (new method)
methods SET_OTHER_VALUE
    importing
      !IR_VALUE type ref to DATA
    raising
      CX_FDT_INPUT .
```

2. Types

```
s_when
BEGIN OF s_when,
      test_parameter    TYPE if_fdt_types=>id,
      r_test_value      TYPE REF TO data,
      return_parameter  TYPE if_fdt_types=>id,
      r_return_value    TYPE REF TO data,
      position           TYPE int1,
   END OF s_when .
```

ts_when

```
ts_when TYPE SORTED TABLE OF s_when WITH NON-UNIQUE KEY test_parameter
            WITH NON-UNIQUE SORTED KEY position COMPONENTS position.
```

Decision Table

1. Types

```
s_range (new type)
types S_RANGE type IF_FDT_RANGE=>S_RANGE .
```

ts_range (new type)

```
types TS_RANGE type IF_FDT_RANGE=>TS_RANGE .
```

s_table_data

```
 BEGIN OF s_table_data,
      col_no        TYPE int4,
      row_no        TYPE int4,
      expression_id TYPE if_fdt_types=>id,
      r_value       TYPE REF TO data,
      ts_range      TYPE ts_range,
    END OF s_table_data .
```

Decision Tree

1. Methods
   - GET_NODE_PROPERTY

```
methods GET_NODE_PROPERTY
    importing
      !IV_NODE type NODE_ID
      !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
    exporting
      !EV_CONDITION type IF_FDT_TYPES=>ID
      !ES_RANGE type S_PARAM_RANGE
      !EV_RESULT type IF_FDT_TYPES=>ID
      !ER_RESULT type ref to DATA
      !EV_PARENT type NODE_ID
      !EV_IS_LEAF type ABAP_BOOL
    raising
      CX_FDT_INPUT
      CX_FDT_CONFIG .
```

   - SET_NODE_PROPERTY

```
methods SET_NODE_PROPERTY
    importing
      !IV_NODE type NODE_ID
      !IV_CONDITION type IF_FDT_TYPES=>ID optional
      !IS_RANGE type S_PARAM_RANGE optional
      !IV_RESULT type IF_FDT_TYPES=>ID optional
      !IR_RESULT type ref to DATA optional
    raising
      CX_FDT_INPUT .
```

2. Types
   - s_param_range (new type)

```
 BEGIN OF s_param_range,
      parameter_id TYPE if_fdt_types=>id,
      ts_range     TYPE if_fdt_range=>ts_range,
    END OF s_param_range .
```

Loop

1. Methods
   - GET_LOOP_PROPERTIES

```
methods GET_LOOP_PROPERTIES
    importing
      !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
    exporting
      !EV_NUMBER_OF_REPEATS type NUMBER_OF_REPEATS
      !EV_LOOP_CONDITION type IF_FDT_TYPES=>ID
      !ES_LOOP_COND_RANGE type IF_FDT_RANGE=>S_PARAM_RANGE
      !ETS_FOR_EACH_WHERE_CONDITION type TS_FOR_EACH_WHERE_CONDITION
      !EV_FOR_EACH_TABLE type IF_FDT_TYPES=>ID
    raising
      CX_FDT_INPUT .
```

   - SET_LOOP_PROPERTIES

```
methods SET_LOOP_PROPERTIES
    importing
      !IV_NUMBER_OF_REPEATS type NUMBER_OF_REPEATS optional
      !IV_LOOP_CONDITION type IF_FDT_TYPES=>ID optional
```

```
      !IS_LOOP_COND_RANGE type IF_FDT_RANGE=>S_PARAM_RANGE optional
      !ITS_FOR_EACH_WHERE_CONDITION type TS_FOR_EACH_WHERE_CONDITION optional
      !IV_FOR_EACH_TABLE type IF_FDT_TYPES=>ID optional
    raising
      CX_FDT_INPUT .
```

2. Types
   - s_rule

```
BEGIN OF s_rule,
position              TYPE fdt_inc_expr_9002_key-pos,
rule_id               TYPE if_fdt_types=>id
exit_condition_id     TYPE if_fdt_types=>id
exit_cond_range       TYPE IF_FDT_RANGE=>s_param_range      continue_condition_id
 TYPE if_fdt_types=>id
continue_cond_range   TYPE IF_FDT_RANGE=>s_param_range
END OF s_rule .
```

Range

1. Types
   - s_range

```
BEGIN OF s_range,
      position     TYPE fdt_seqnr,
      sign         TYPE ddsign,
      option       TYPE ddoption,
      low          TYPE if_fdt_types=>id,
      r_low_value  TYPE REF TO data,
      high         TYPE if_fdt_types=>id,
      r_high_value TYPE REF TO data,
   END OF s_range .
```

   - ts_range

```
ts_range TYPE SORTED TABLE OF s_range
      WITH NON-UNIQUE KEY sign option low high
      WITH NON-UNIQUE SORTED KEY position COMPONENTS position .
```

Search Tree

1. Methods
   - CREATE_NODE

```
methods CREATE_NODE
    importing
      !IV_PARENT type NODE_ID optional
      !IV_SIBLING type NODE_ID optional
      !IV_CONDITION type IF_FDT_TYPES=>ID optional
      !IS_RANGE type S_PARAM_RANGE optional
      !IV_RESULT type IF_FDT_TYPES=>ID optional
      !IR_RESULT type ref to DATA optional
    exporting
      !EV_NODE type IF_FDT_SEARCH_TREE=>NODE_ID
    raising
      CX_FDT_INPUT .
```

   - GET_NODE_PROPERTY

```
methods GET_NODE_PROPERTY
    importing
```

```
    !IV_NODE type NODE_ID
    !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
  exporting
    !EV_CONDITION type IF_FDT_TYPES=>ID
    !ES_RANGE type S_PARAM_RANGE
    !EV_PARENT type NODE_ID
    !EV_RESULT type IF_FDT_TYPES=>ID
    !ER_RESULT type ref to DATA
  raising
    CX_FDT_CONFIG
    CX_FDT_INPUT .
```

- SET_NODE_PROPERTY

```
methods SET_NODE_PROPERTY
  importing
    !IV_NODE type NODE_ID
    !IV_CONDITION type IF_FDT_TYPES=>ID optional
    !IS_RANGE type S_PARAM_RANGE optional
    !IV_RESULT type IF_FDT_TYPES=>ID optional
    !IR_RESULT type ref to DATA optional
  raising
    CX_FDT_INPUT .
```

2. Types
   - s_param_range (new type)

```
BEGIN OF s_param_range,
  parameter_id TYPE if_fdt_types=>id,
  ts_range     TYPE if_fdt_range=>ts_range,
END OF s_param_range .
```

Rule

1. Methods
   - GET_CONDITION_RANGE(new method)

```
methods GET_CONDITION_RANGE
  importing
    !IV_TIMESTAMP type TIMESTAMP optional
  returning
    value(ES_CONDITION_RANGE) type IF_FDT_RANGE=>S_PARAM_RANGE
  raising
    CX_FDT_INPUT .
```

   - SET_CONDITION_RANGE (new method)

```
methods SET_CONDITION_RANGE
  importing
    value(IS_CONDITION_RANGE) type IF_FDT_RANGE=>S_PARAM_RANGE
  raising
    CX_FDT_INPUT .
```

   - CLEAR

```
methods CLEAR
  importing
    !IV_CONDITION type ABAP_BOOL optional
    !IV_TRUE_ACTION type ABAP_BOOL optional
    !IV_FALSE_ACTION type ABAP_BOOL optional
    !IV_TRUE_ACTION_EXTENDED type ABAP_BOOL optional
```

```
      !IV_FALSE_ACTION_EXTENDED type ABAP_BOOL optional
      !IV_CONDITION_RANGE type ABAP_BOOL optional .
```

Ruleset

1. Methods
   - GET_RULESET_CONDITION

```
methods GET_RULESET_CONDITION
    importing
      !IV_TIMESTAMP type IF_FDT_TYPES=>TIMESTAMP optional
    exporting
      !EV_CONDITION_ID type IF_FDT_TYPES=>ID
      !EV_NO_CONDITION type ABAP_BOOL
      !ES_CONDITION_RANGE type IF_FDT_RANGE=>S_PARAM_RANGE
    raising
      CX_FDT_INPUT .
SET_RULESET_CONDITION
methods SET_RULESET_CONDITION
    importing
      !IV_CONDITION_ID type IF_FDT_TYPES=>ID optional
      !IS_CONDITION_RANGE type IF_FDT_RANGE=>S_PARAM_RANGE optional
    preferred parameter IV_CONDITION_ID
    raising
      CX_FDT_INPUT .
```

2. Types
   - s_rule

```
BEGIN OF s_rule,
      position      TYPE if_fdt_ruleset=>position,
      function_id   TYPE if_fdt_types=>id,
      condition_id  TYPE if_fdt_types=>id,
      cond_range    TYPE if_fdt_range=>s_param_range,
      valid_from    TYPE if_fdt_types=>timestamp,
      valid_to      TYPE if_fdt_types=>timestamp,
      rule_id       TYPE if_fdt_types=>id,
      switch        TYPE if_fdt_ruleset=>switch,
      exit_ruleset  TYPE abap_bool,
      restart_option type RESTART_OPTION, "
    END OF s_rule .
```

Related Content

- BRFplus – The Very Basics
- Carsten Ziegler, About Business Rules:
  https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/9713
- Carsten Ziegler, BRFplus a Business Rule Engine written in ABAP,
  https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/8889
- Carsten Ziegler, Important Information for Using BRFplus

  https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/11632

  For more information, visit the Business Rules Management homepage.

Copyright