

# How to Create and Process a Simple Rule



## Applies to:

Business Rules Framework plus shipped with SAP NetWeaver 7.0 Enhancement Package 1.

## Summary

This tutorial explains how to create a simple rule and process it. The rule design steps can be performed by program code or with help of a UI. The rule processing needs some application code as the application developer needs to invoke the rule processing.

**Author:** Shashi Kanth Kasam, Orenthung Ovung

**Company:** SAP Labs India

**Created on:** 19 August 2008

## About the Authors



Shashi Kanth Kasam is a Rules Developer in the BRFplus team. He has 2 years of experience in building business rules and has been part of this team since April 2008.



Orenthung Ovung is an Information Developer in the BRFplus team. He has been part of this team since March 2008.

## Table of Contents

Prerequisites .....	3
Designing Rules Using the API .....	3
Procedure.....	3
Factory .....	3
Working with the Function.....	3
Working with Constant Expressions.....	4
Creating the Data Objects.....	4
Activating the Function.....	4
Simulating the Function .....	5
The Complete Report .....	6
Defining Rules Using the UI.....	8
Procedure.....	8
Creating the application .....	8
Creating the Function.....	8
Creating a Constant Expression .....	8
Assigning Top Expression to the Function .....	8
Activating the Function.....	9
Related Content.....	10
Copyright.....	11

## Prerequisites

You have a basic knowledge of BRFplus.

## Designing Rules Using the API

### Procedure

#### Factory

The factory is the central point of entry for all BRFplus users to directly access the API. The factory is also the basis for implementing BRFplus business objects and dependent objects. The returned instances are described by the class implementing the **get\_instance** method of the **if\_fdt\_factory** interface. This in turn facilitates the replacement of any BRFplus class except the factory class itself. One can access an existing object or create a new object. The following instance can be used to create local objects that are not transported.

```
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance(
    if_fdt_constants=>gc_application_tmp ).
```

**IF\_FDT\_CONSTANTS** is the central constants interface. **GC\_APPLICATION\_TMP** is a local application which can be used for testing purposes and to create local objects that are not transported.

#### Creating New Objects

The factory created above can be used to create the objects needed for this tutorial.

```
lo_function ?= lo_factory->get_function( ).
lo_constant ?= lo_factory->get_expression(
    iv_expression_type_id = if_fdt_constants=>gc_exty_constant ).
```

Methods like **GET\_FUNCTION** and **GET\_EXPRESSION** can take the parameter **IV\_ID** as input. The parameter **IV\_ID** is the unique object ID (UUID).

**IF\_FDT\_CONSTANTS=>GC\_APPLICATION\_TMP** is the ID for the BRFplus application.

#### Note:

- To work with existing objects, the unique object ID has to be supplied. The UUID is created automatically for new objects.
- Every object that is persistent in the database has a UUID.

Using the three lines of code given above, a new function is created with the method **GET\_FUNCTION**, and a new expression of type constant is created with the method **GET\_EXPRESSION**. The expression type (ID of a BRFplus object) must be supplied while creating a new expression.

For instance, if you work with an application-specific factory which is retrieved independently, you need to define it before the application is set for the new objects. The objects are assigned to the application and may also inherit attributes such as persistency type, local property or the exit class from the application.

#### Working with the Function

The objects must be enqueued before they are saved. The following lines of code show the enqueueing of the function and the setting of its attributes.

```
lo_function->if_fdt_transaction~enqueue( ).
lo_function->if_fdt_admin_data~set_name( 'HELLO_WORLD' ).
lo_function->if_fdt_admin_data~set_texts(
    iv_text          = 'Hello World Tutorial'
    iv_short_text    = 'Hello World' ).
```

The function needs a top expression. In this tutorial, the top expression is a constant expression type.

## Working with Constant Expressions

The constant expression type is the most simple expression type. You can set a constant value which is returned by the expression when processed. There is no conditional aspect for the expression.

Set the constant value and assign the expression as the top expression as shown below:

```
lo_constant->if_fdt_transaction~enqueue( ).
lo_constant->set_constant_value( 'Hello World.' ).
lo_function->set_expression( lo_constant->mv_id ).
```

Attributes such as name, short text and text are available for any persistent BRFplus objects. For instance, you can set the name and other attributes for the expression.

In BRFplus, there are named and unnamed objects. A named object is an object that has a name (for example, the function in this case). Named objects can be reused. An unnamed object is an object that does not have a name or the name is set to initial (for example, the constant expression).

To create a constant expression which can be used in various rules, you need to give it a name. If you want to use the expression or the application you also need to set the access level. The default is that the object is only visible within the application for which it is created.

Both functions and expressions have results. The result is defined by a data object. Some expressions like constant expressions derive the result data object depending on the attribute settings. A function can take over the result from its top expression. In this tutorial, the function takes the result object from the top expression (constant expression) and so the result is not explicitly set for the function.

## Creating the Data Objects

For creating the data objects, the following data declarations are needed:

```
DATA: lo_factory          TYPE REF TO if_fdt_factory,
      lo_constant        TYPE REF TO if_fdt_constant,
      lo_function        TYPE REF TO if_fdt_function,
      lt_message         TYPE if_fdt_types=>t_message,
      lv_activation_failed TYPE abap_bool,
      lx_fdt             TYPE REF TO cx_fdt,
      lo_result          TYPE REF TO if_fdt_result,
      lv_name            TYPE if_fdt_types=>name,
      lv_string          TYPE string.
```

```
FIELD-SYMBOLS <ls_message> TYPE if_fdt_types=>s_message.
```

## Activating the Function

The objects have to be activated once they are created. Only the active version of the object is relevant for processing. BRFplus uses a versioning concept that allows processing of past versions as well. Checks are done automatically during activation. The application has to be activated before the function is activated.

BRFplus objects can be processed without saving the changes. Use the code snippet given below to activate and save the object.

```
lo_function->if_fdt_transaction~activate(
  EXPORTING iv_deep          = abap_true
  IMPORTING et_message      = lt_message
           ev_activation_failed = lv_activation_failed ).
```

Try.

```
lo_function->if_fdt_transaction~save( iv_deep = ABAP_true ).
Commit work.
Catch cx_fdt into lx_fdt.
Rollback work.
ENDTRY.
```

```
lo_function->if_fdt_transaction~dequeue( iv_deep = abap_true ).
```

The function gets activated and at the same time the changes in the constant expression are activated. This is possible because the constant expression is referenced by the function as its top expression. Finally, save and dequeue both objects.

If **EV\_ACTIVATION\_FAILED** is set to **true ('X')** then the activation was not successful. In such a situation, the object cannot be processed though it can be saved.

```
IF lv_activation_failed EQ abap_true.
  LOOP AT lt_message ASSIGNING <ls_message>.
    WRITE / <ls_message>-text.
  ENDLOOP.
  RETURN. ">>>
ENDIF.
```

### Simulating the Function

To process the function and check the result, use the following code:

```
TRY.
  lo_function->process( IMPORTING eo_result = lo_result ).
  lo_result->get_value( IMPORTING ea_value = lv_string ).
  WRITE lv_string .
CATCH cx_fdt INTO lx_fdt.
  LOOP AT lx_fdt->mt_message ASSIGNING <ls_message>.
    WRITE / <ls_message>-text.
  ENDLOOP.
ENDTRY.
```

## The Complete Report

```

*&-----*
*& Report   FDT_SDN_HELLO_WORLD
*&
*&-----*
*&
*&
*&-----*

REPORT   FDT_SDN_HELLO_WORLD.

DATA: lo_factory          TYPE REF TO if_fdt_factory,
      lo_constant        TYPE REF TO if_fdt_constant,
      lo_function        TYPE REF TO if_fdt_function,
      lt_message         TYPE if_fdt_types=>t_message,
      lv_activation_failed TYPE abap_bool,
      lx_fdt             TYPE REF TO cx_fdt,
      lo_result          TYPE REF TO if_fdt_result,
      lv_name            TYPE if_fdt_types=>name,
      lv_string          TYPE string.

FIELD-SYMBOLS <ls_message> TYPE if_fdt_types=>s_message.

* get a reference to the instance of the factory
lo_factory = cl_fdt_factory=>if_fdt_factory~get_instance(
  if_fdt_constants=>gc_application_tmp ).
* note that you are not required to use an application as input; however,
* we recommend to do so and work with a specific application instead of
* the generic local FDT application

* get the objects we need for this demo
lo_function ?= lo_factory->get_function( ).
lo_constant ?= lo_factory->get_expression(
  iv_expression_type_id = if_fdt_constants=>gc_exty_constant ).

* set some attributes for the function
lo_function->if_fdt_transaction~enqueue( ).

*lo_function->if_fdt_admin_data~set_name( 'HELLO_WORLD' ).
lv_name = cl_fdt_services=>get_unique_name( ).
lo_function->if_fdt_admin_data~set_name( lv_name ).
* you may need to replace the name because the function name needs to
* unique within an application; as soon as someone has run this report
* the name is occupied (within this application); Use GET_UNIQUE_NAME method
* of the class CL_FDT_SERVICES to get a unique name.
lo_function->if_fdt_admin_data~set_texts(
  iv_text      = 'Hello World Tutorial'   "#EC NOTEXT
  iv_short_text = 'Hello World' ).      "#EC NOTEXT

* set constant value
lo_constant->if_fdt_transaction~enqueue( ).
lo_constant->set_constant_value( 'Hello World.' ).  "#EC NOTEXT
* make the constant expression the top expression of the function
lo_function->set_expression( lo_constant->mv_id ).

* no need to set a result in this case since the constant can derive
* its result data object automatically

* activate and save the changes; only activated changes are relevant

```

```

* for processing; however, active memory states can also be processed
* without a need to save to the DB
lo_function->if_fdt_transaction~activate(
  EXPORTING iv_deep          = abap_true
  IMPORTING et_message       = lt_message
            ev_activation_failed = lv_activation_failed ).

```

Try.

```

  lo_function->if_fdt_transaction~save( iv_deep = ABAP_true ).
  Commit work.
Catch cx_fdt into lx_fdt.
  Rollback work.
ENDTRY.

```

```

lo_function->if_fdt_transaction~dequeue( iv_deep = abap_true ).

```

\* error handling

```

IF lv_activation_failed EQ abap_true.
  LOOP AT lt_message ASSIGNING <ls_message>.
    WRITE / <ls_message>-text.
  ENLOOP.
  RETURN. ">>>
ENDIF.

```

TRY.

```

* process the function
  lo_function->process( IMPORTING eo_result = lo_result ).
* get the result
  lo_result->get_value( IMPORTING ea_value = lv_string ).
  WRITE lv_string .
  CATCH cx_fdt INTO lx_fdt.
* error handling
  LOOP AT lx_fdt->mt_message ASSIGNING <ls_message>.
    WRITE / <ls_message>-text.
  ENLOOP.
ENDTRY.

```

## Defining Rules Using the UI

### Procedure

#### Creating the application

1. In the menu bar, choose *Workbench -> Create Application...*
2. In the *Object Creation* dialog box that appears, enter **Z\_Hello\_World** in the *Name* field.
3. Choose *System* as the storage type and select the *Create Local Application* check box.
4. Enter **\$TMP** in the *Development Package* field.
5. Choose *Create & Navigate To Object*.
6. Save the application.
7. Choose *Activate* button.
8. In the *Confirmation of Activation* dialog box that appears, choose *OK*.  
The application gets activated.

#### Creating the Function

1. Under the *Detail* section, choose the *Assigned Objects* tab.  
The *Assigned Objects* tab page appears.
2. Choose *Function* from the *Type* field and choose *Create Object*.
3. In the *Object Creation* dialog box that appears, enter **Hello\_World\_Function** in the *Name* field and choose *Create & Navigate To Object*.  
The function is created and opens in the *Object Manager* panel.

#### Creating a Constant Expression

1. Navigate to the application by choosing *Z\_HELLO\_WORLD* next to the *Application* field under the *General* section.
2. Under the *Detail* section, choose *Assigned Objects* tab.  
The *Assigned Objects* tab page opens.
3. Choose *Expression* from the *Type* field and choose *Create Object*.
4. In the *Object Creation* dialog box that appears, enter **Hello\_World\_Constant** in the *Name* field; choose *Constant* in the *Type* field.
5. In the *Constant* section that appears, choose *Text* from the *Type* field.
6. In the *Value* field that appears, enter **Hello World**.
7. Choose *Create*.

#### Assigning Top Expression to the Function

1. Choose *Function* from the *Type* field.  
The *HELLO\_WORLD\_FUNCTION* appears in the table.
2. Choose *HELLO\_WORLD\_FUNCTION*.  
The function appears in the *Object Manager* panel.
3. Choose  (*Graphical Access*) next to the *Top Expression* field.
4. In the context menu, choose *Select*.
5. In the *Object Query* dialog box that appears, select the *HELLO\_WORLD\_CONSTANT* constant expression.  
The constant expression is assigned as the top expression of the function.
6. Choose the *Signature* tab.

7. Choose  (*Graphical Access*) under the *Result Data Object* section.
8. In the context menu, choose *Default Objects* → *Text*

### Activating the Function

1. Choose *Activate* button.
2. In the *Confirmation of Activation* dialog box that appears, check the *Include Referenced Objects* checkbox.

A list of the affected objects appears.

3. Choose *OK*.

The function and the constant expression get activated.

## Related Content

- [BRFplus – The Very Basics](#)
- [Articles in the Business Rules Management](#) section
- Wikipedia, Business Rules: [http://en.wikipedia.org/wiki/Business\\_rules](http://en.wikipedia.org/wiki/Business_rules)
- Wikipedia, Business Rule Management System: [http://en.wikipedia.org/wiki/Business\\_Rule\\_Management\\_System](http://en.wikipedia.org/wiki/Business_Rule_Management_System)
- 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>
- Rajagopalan Narayanan, Business Rules and Software Requirements, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/70c669d8-3ac2-2a10-0e96-c7c3786168f0>
- Rajagopalan Narayanan, Seven Tips for Your First Business Rules Project, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/201a9e3d-3ec2-2a10-85b2-ce56d276dd7a>
- Rajagopalan Narayanan, Real World Return of Investment Scenarios with Business Rules Management, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/b050905e-3cc2-2a10-979a-81a57a787f56>
- Rajagopalan Narayanan, Five Reasons to Build Agile Systems Using Business Rules Management Functionality, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/504486eb-43c2-2a10-f5a7-e84ef3fd45be>
- Rajagopalan Narayanan, How Business Rules Management Functionality Helps Tariff Plans Management in Transportation and Shipping, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/40a9cf69-40c2-2a10-8a8b-969fb311dd31>
- Rajagopalan Narayanan, Getting Started with Business Rules Management, <https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/70c669d8-3ac2-2a10-0e96-c7c3786168f0>

## Copyright

© 2008 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, System i, System i5, System p, System p5, System x, System z, System z9, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, POWER5+, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

Any software coding and/or code lines/strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.