BRF: Business Rule Framework

Applies to:

For more information, refer to BRF.

Summary

This tutorial explains about the Business Rule Framework. BRF is a framework used to customize the events and actions. Events are similar to the object oriented events which triggers certain process. For each event there can be an action attached which performs some activity when event is triggered.

Author: Seemanthini R

Company: SAP Labs India

Created on: 03 March 2009

Author Bio

I have almost three and half years of experience in SAP Labs India as Senior Consultant – Development. Have worked on various fields like ABAP, ABAP OO, Control Frameworks, CRM Web UI, Rule contexts and BRF.

Table of Contents

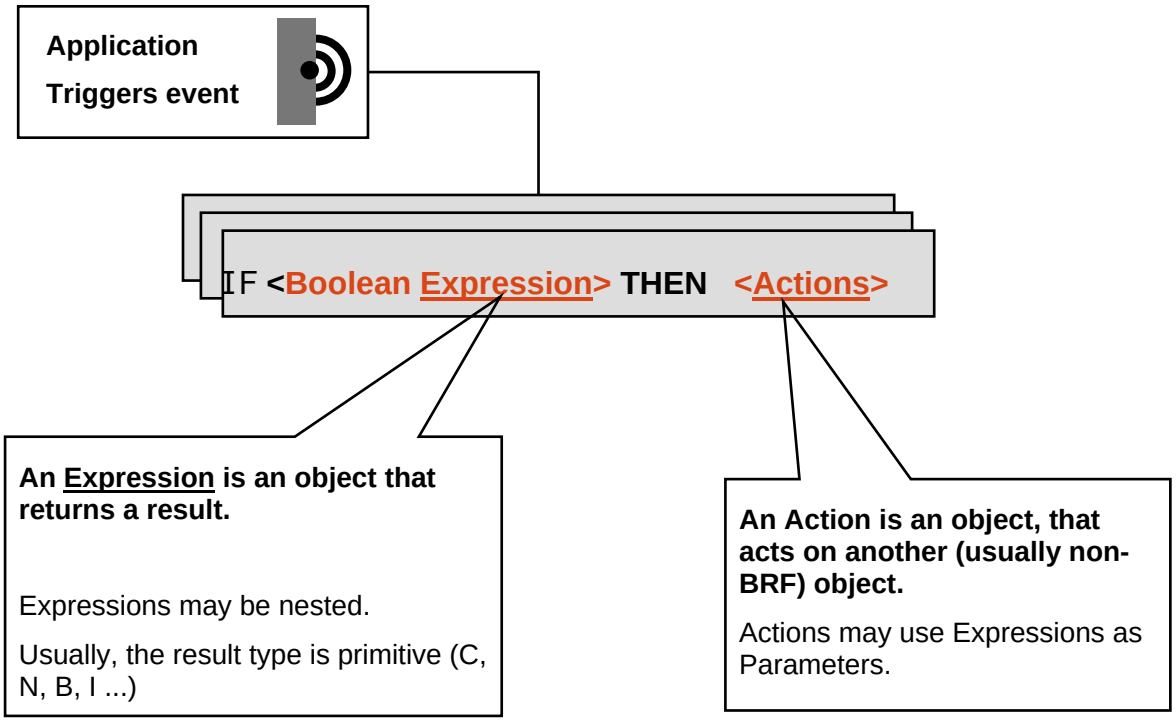
What is BRF?

Business Rule Framework is a framework used to customize the events and actions. Events are similar to the object oriented events which triggers certain process. For each event there can be an action attached which performs some activity when event is triggered.

You can use the BRF rule tool to define and implement technical and business process oriented sets of rules. The Claims Management system uses the BRF exclusively as a business process oriented set of rules.

Purpose of using Business rule Framework

The BRF is an event-controlled runtime environment for processing rules. Single BRF object can have any number of events. Each event can have any number of rules attached to it, whereby a rule normally consists of a Boolean expression and an action. If the expression returns the value TRUE, the action is executed. This is shown in the picture 1.



Picture 1: Execution of Rules in the BRF, using expression and action

The BRF also contains a maintenance environment in which you can edit and configure BRF objects. You can configure both technically oriented as well as business process oriented rules. This means that you configure the rules in the maintenance environment, and the rules are processed in the runtime environment.

The BRF is object-oriented and therefore offers appropriate enhancement mechanisms that are modification-free and upgrade-independent.

Advantages of BRF

- Easy implementation and configuration of rules with minimal coding
- Easy maintenance of rules as there is not much coding involved
- Easy extension in order to support application specific data
- Support for keeping track of Rules execution
- Enables mapping of own business processes
- Guarantees application-independence

Benefits of BRF

From SAP application's point of view

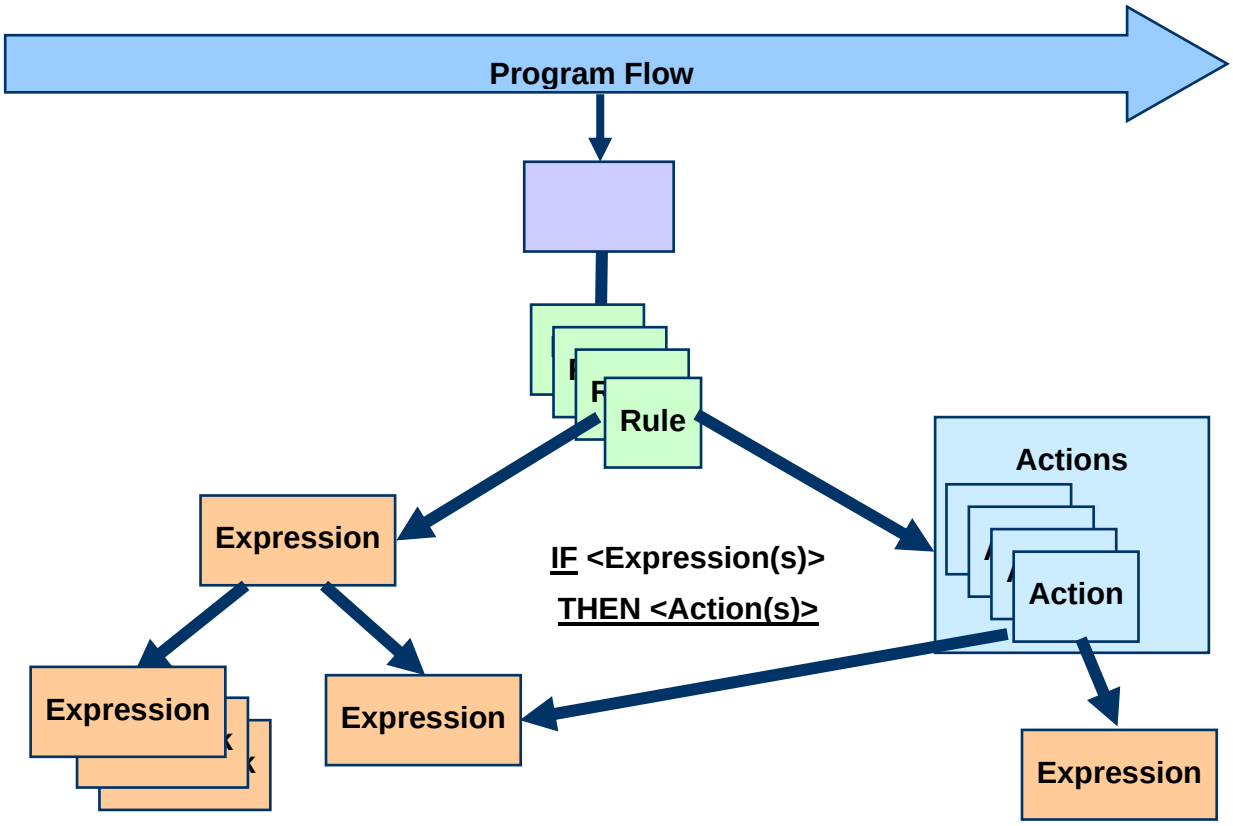
- Easy maintenance of business/technical rules
- Reduces coding – and thereby potential for errors
- Reduces customizing
- Guarantees Application independence
- Guarantees modification-free extensibility
- Extensibility: BRF is available to customers and partners for own development
- BRF can be used to extend existing applications (e.g. Case Management, Business Partner) to handle Rules
- Where Used List for BRF Objects
- Trace functionality helps development to easily test the rules
- Easy to copy from one client to another client of same system.

From SAP customer's point of view

- Customers need not share business rules with SAP
- Flexible Rule Maintenance
- Easy maintenance of rules as there is not much coding involved
- Cost Reduction (TCO)
- Reusability of Rule Sets
- Time dependent rules (Validity periods for each rules)
- Self contained documentation in the form of Network Graphics

First step towards your own BRF

BRF Objects



Picture 2: BRF at Runtime – An Example

Application class

- Application class identifies an application that uses the BRF (Business Rule Framework).
- You must specify the application class for each BRF object (event, expression, action).
- Application classes are logical transport objects. This means that an application class can only occur once throughout all SAP systems. (Client independent)

Events

- Events may be considered the “Entry point/gateway” to Rule Evaluation and Execution in BRF.
- Events are implemented at specific points within a business process – e.g.: when a document is stored, when a payment is transferred etc.
- An Event is associated with one or more “Rules” which will be executed when the Event is triggered.
- Triggering an Event has to be done via coding.
- An Event is provided with an application specific context.

Event Context

Very simple applications might be able to work without any context information. Generally, though, an event corresponds to a context, and an event object contains information on the context. This means that different events have different contexts.

Example: In invoice verification, for example, two events with different contexts would be set up:

- Check invoice header (the invoice number could form the context, for example)
- Check invoice item (the combination of invoice number and item number could form the context, for example)

The BRF does not prescribe what a context is or how to implement it.

Here follows a description of two options for implementing a context:

- You regard all the data of your application as the context. In other words, you actively make available to the BRF **all** the data that might be used in rules at some stage.
- You define a very limited context that contains the minimum amount of data required. In the case of invoice verification at item level, this would be the invoice/item number. All other data on the item can only be derived indirectly via suitable data sources and the invoice/item number.

BRF events and their respective contexts (implementation and scope) are therefore defined by the developer of the application.

Rules

- Rules are assigned to an event and will be processed in the order they were created
- Syntax is: IF expression THEN action
- Rules can be time dependent. This can be achieved by using the date fields of the rules.
- Rules can have a gate expression

Gate Expression

Gate Expression is a Boolean expression that determines at runtime whether a rule or an entire rule set really should be processed. A gate expression must always be Boolean. Other result categories are not permitted.

- Rules can be grouped into a Rule set
- A rule set can be assigned to several events (reuse)

Key fields of the Rules

Rules always have at least the following key fields:

- Event
- Expression
- Action

In some applications it is necessary to add other key fields. This means that right from the start you can prevent rules that make no sense in a certain context from being executed.

For example: Application developer can even come up with a new field in Rules to specify the Priority on which the order of execution of rules will depend.

Rule Sets

Grouping of rules that belong together from a business point of view, and that are distributed over several events.

You can use rule sets to map simple units for managing rules. For example, you can jointly activate or deactivate the rules of a rule set.

The rules of a rule set have the same attributes as the rules that are directly assigned to the events (see Rule).

You can define a gate expression for a rule set. If the rule set is active, you can use the gate expression to control whether the rule set should be applied. You can define a gate expression for the individual rules in a rule set.

Actions

- Actions are BRF objects that start some activity as part of Rule Execution
- They form the “THEN“ part of the Rule (Refer to Picture 1)
- Standard BRF Actions can be of the following types:
  - Execute Function Module / Method
  - Message Output
- The content of an action is usually application specific and is implemented with SAP standard tools.
  - Start a specific workflow/Raise Event

Expressions

Expressions are BRF objects that return a definite result. Boolean Expressions form the “IF” part of the Rule.

- **Expressions are freely definable within the BRF framework without any modifications.**
  - SAP can provide new expression types without interfering existing functionality
  - Customers can define their own expression types if very specific expressions are needed
- **New expression types need to be implemented in two places by ABAP development.**
  - Maintenance class: Which data have to be entered for an expression? Which return values are expected?
  - Runtime class: Which checks, computations etc. are needed when the expression is evaluated?
- **New expression types can have the same properties like standard expression types**
  - Nesting
  - Using the event context. Etc.
- **Expresson types**

Expression type is determined by the implementing class. This also defines how an expression is calculated. These expression types are provided in SAP standard.

Note: “Implementing class” is more of a technical term. It is used synonymously for expression type (and also action type) in the sense of “implementing class of an expression”.

Below listed are few of the standard expression types delivered by SAP. You can also create your own expression types.

1. **Constant (Implementing Class 0CN001)**

The constant is the trivial expression type. Apart from the value and the typing, no other information is required.

2. **Simple value request (field of a structure)**

Use this expression type if the required value is in a structure in the application. For example, if you want to determine the current user in the SAP system, the system accesses the *uname* field in the SYST structure (sy-uname).

The application must only contain an access function module that makes the structure available to the BRF. The BRF then extracts the particular field itself.

3. **Field of a line of an internal table (0TB001)**

This expression type operates in the same way as the “simple value request (field of a structure)” expression type.

In addition, you must specify the corresponding table line using a SELECT mechanism: SELECT <FIELD> FROM <ITAB> WHERE <CONDITION>.

<ITAB> must be available via an access function module (data source), and <FIELD> must be a component of the internal table <ITAB>.

4. **Call function module/method (0CF001)**

Use this expression type if no other expression types in the BRF are suitable for the purpose in mind.

A special expression can be implemented in a function module. You can also transfer parameters (such as other expressions) to the function module. In theory, you can realize any value request and any operation in this way. You can use module BRF_CALL_FUNCTION_TEMPLATE as a copy template.

Recommendation: However, it is recommended that you follow this procedure only for expressions that are required infrequently, as the special features of the function that might be behind the function module cannot be considered. Maintenance of the expression in the BRF is unable to consider these special features.

With frequently used expressions, you should implement both the runtime class and the maintenance class, in other words add own expression types.

5. **Three-operand arithmetic (03O001)**

Using this expression type, which has a *Boolean* result type, you can link three Boolean expressions logically with one another. In addition to the simple links “A AND B AND C” as well as “A OR B OR C”, you can also use the compounded links “(A AND B) OR C” as well as “A AND (B OR C)”.

You can also negate each A, B, C expression.

6. **Simple formula (0FR001)**

Sometimes it is rather tedious to derive facts from existing expression types. Instead, use the simple formula.

With the aid of the simple formula you can perform simple arithmetical and logical calculations that consider the sequence of the operators and compounding.

7. **SAP formula interpreter (0FB001)**

This expression type operates in the same way as the *simple formula* expression type. However, as this expression type uses the SAP formula editor/formula interpreter, it is far more powerful than the *Simple Formula*.

8. **Tag Expression**

Tags are the one Order Data Expression for Reading Data from One Order Context. The customizing to set the tag hierarchy will be done in maintenance views CRMV_BRF_TAG and CRMV_ BRF_TAG_AS

9. **Truth table ("bit pattern recognition") (0PM001)**

Use this expression type in the environment of configurable status administration. All the incoming expressions in a truth table have a Boolean result type. A truth table always delivers a scaleable return value.

The following example describes how a truth table functions:

Return Value	Expression 1	Expression 2	Expression 3
3	TRUE	*	FALSE
1	TRUE	TRUE	*
4	*	*	*

The system calculates expressions 1 to 3 at runtime. Then the system checks each line one by one to see if the bit pattern matches the current line.

- If the bit pattern matches, the return value of the corresponding line is given as the result of the expression. All other lines are ignored.
- If the bit pattern does not match, the system checks the next line.
- If the bit pattern does not match any line at all, the behavior is not defined. For this reason you should explicitly introduce a last line in which each column contains an asterisk (*).

An asterisk means that the value of the expression of the corresponding column is irrelevant. For example, you get the return value 3 with TRUE/TRUE/FALSE and also with TRUE/FALSE/FALSE. In this case, the value of expression 2 is irrelevant.

Examples:

- (TRUE, TRUE, FALSE): Return value 3, as the **first** line already matches.
- (TRUE, TRUE, TRUE): Return value 1, as the **second** line already matches.
- (FALSE, FALSE, TRUE): Return value 4, as only the **third** line matches.

Recommendation:

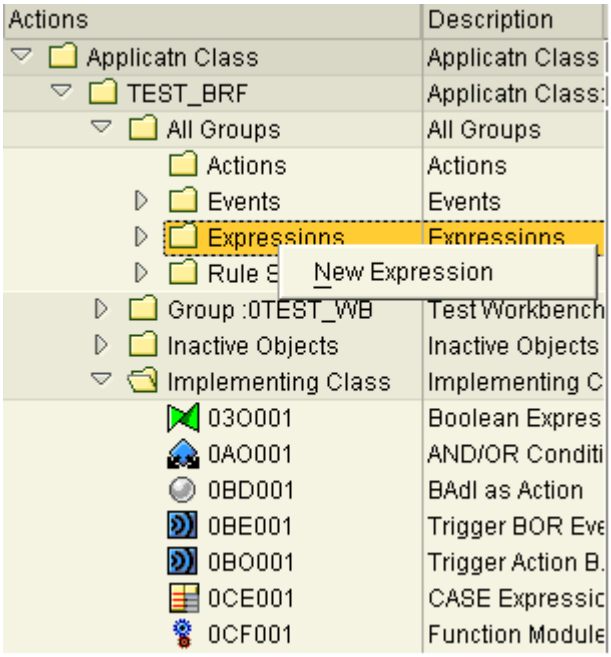
If you are not sure whether you have explicitly checked all possible combinations, add a last line that contains an asterisk (*) for each expression and that returns a value. In the area of status calculation, this value could have the meaning "unexpected status“.

Transaction codes used in BRF

Table below shows the few important transactions used in BRF.

Transaction	Description
BRF	Business Rule Framework - Workbench
BRFAPL01	Create Application Class
BRFAPL02	Change Application Class
BRFAPL03	Display Application Class
BRFEVT01	Create Event
BRFEVT 02	Change Event
BRFEVT 03	Display Event
BRFEXP01	Create Expression
BRFEXP02	Change Expression
BRFEXP03	Display Expression
BRFACT01	Create Action
BRFACT02	Change Action
BRFACT03	Display Action
BRFIMC01	Create Implementation Class
BRFIMC02	Change Implementation Class
BRFIMC03	Display Implementation Class
BRFRLS01	Create Rule Set
BRFRLS02	Change Rule Set
BRFRLS03	Display Rule Set
BRF_OVERVIEW	Displays overview of BRF objects

Creation of the BRF objects via the Transaction code **BRF** is shown in the below picture



Picture 3: You can even create the BRF objects (Events, Rules, Rules sets and Actions) from the BRF transaction by right clicking on the respective folders

Database tables which will hold the BRF objects

The table below gives the information about few database tables which play important role in BRF.

DataBase Table	Description
TBRF000	BRF: Application Class
TBRF000T	BRF: Application Class Text
TBRF042	BRF: Text GUIDs for BRF Objects
TBRF100	BRF: Application Class (Customer-Specific)
TBRF110	BRF: Event
TBRF110T	BRF: Event - Text
TBRF120	BRF: Technical Context
TBRF150	BRF: Expression
TBRF170	BRF: Actions
TBRF180	BRF: Implementing Class
TBRF200	BRF: Where-Used List of BRF Objects in BRF Objects
TBRF210	BRF: Assignment of Event, Expression, Action

Few special actions/settings on BRF objects

Grouping of BRF objects

You can group BRF objects (events, rules, expressions or actions) together in a group to sort or filter them. You can configure the groups as you wish.

To create a grouping, proceed as follows:

1. Call transaction BRF
2. Choose Utilities → Object Groups.
3. This takes you to the Display View "BRF: Groups": Overview screen.
4. Choose  with the quick info text Display -> Change.
5. Choose New Entries.
6. Create your group.

History management

You can perform history management on the BRF objects that are delivered, meaning that you can record and track chronological changes to a BRF object.

The option provided activates the history management for all objects in the current application class.

Once you have activated history management, you are not able to deactivate it.

If individual implementing classes of an application class do not support history management, there is no history management for the BRF definitions that refer to these implementing classes.

Copy of implementation classes

This makes you copy all the application-neutral BRF objects (in other words, all expression types and the action types delivered by SAP) available to your application class.

Procedure

1. Call up transaction SE38.
2. In the *Program* field, enter the report name BRF_COPY_IMPL_CLASSES.
3. Choose  with the quick info text *Execute*.
4. In the *Implementing Class* field, enter an asterisk (*).

   This means that you are selecting all standard BRF objects.

5. In the *Application Class* field, enter the name of your application class.
6. Choose  with the quick info text *Execute*.

Example: Refer to Picture 3: Shows the expression types and action types placed under the folder *Implementing Class.*

Calling Events

In this step you will call a BRF event in your application. To do so, you have to create a function module to wrap the call of the event.

Procedure

1. Implement a function module.

In doing so, use function module BRF_DEMO_RAISE_EVENT_SAMPLE as a copy template.

2. If necessary, change the function module that you have copied.

You can use the copied function module without changing it.

Info: Function module BRF_DEMO_RAISE_EVENT can also be used.

3. Call this function module in your application.

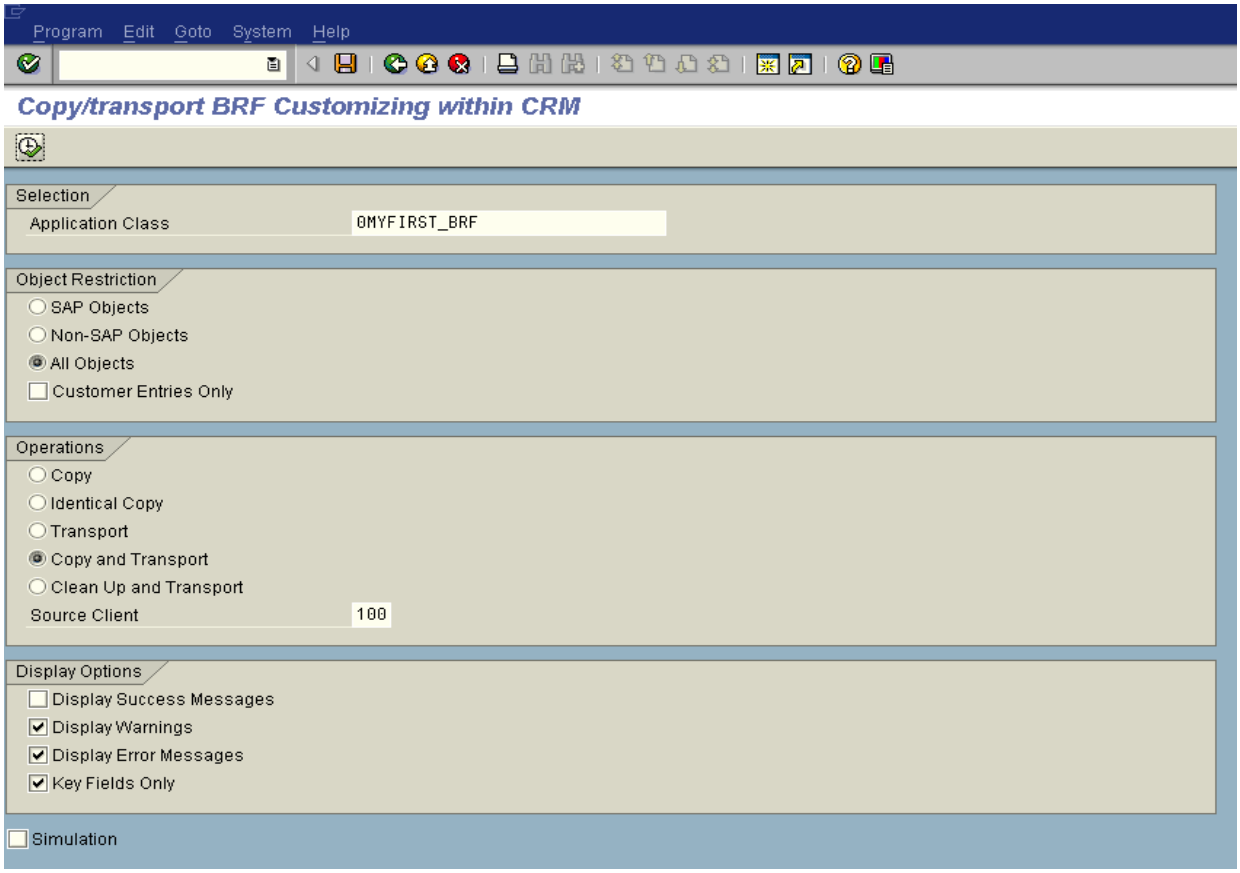
Result

You have called an event in your application. Before you can define rules for this event, you must register the event with the BRF or you have to create the Event in the BRF. The creation has been discussed in the later units.

Transporting BRF objects within SAP CRM

It is possible to copy and/or transport BRF object from one client to another client of SAP CRM. Since the BRF application class will be cross-client object, the system will copy the objects based on the application class name.

Select the target system. Go to transaction SPRO-> Customer Relationship Management -> Industry-Specific Solutions -> Public Sector -> Business Rule Framework -> Transport and/or Copy BRF Objects within SAP CRM



Specify the Application class name which has to be copied or transported. Also specify the source client number from which you have to copy.

The operations that can be performed are Copy, Transport, Copy and Transport, Clean up and Transport.

Simulation mode has to be switched off to make the changes in the database.

Note: Beware while specifying the source client number and the selection of operations. Your wrong move can erase the whole BRF objects for the specified application class.

Creation of your own BRF Objects

Application Class

Create an application class using the transaction BRFAPL01

Details on an application class are stored in client-dependent tables. The following properties can be maintained:

- Application Class name : 0MYFIRST_BRF
- Short text: Short description of Application class
- Trace class: The BRF offers the option of executing a trace for diagnosis purposes. The trace functionality is realized by an ABAP-OO class that implements the interface IF_TRACE_BRF. The BRF standard trace class CL_TRACE_BRF implements this interface. It is always used by default. If the functionality of the standard trace class is not sufficient, you can derive this or implement the interface directly, and enter your own trace class here instead of the standard trace class.
- Implementing tree class: Transaction BRF comprises a navigation tree on the left side and a maintenance area on the right side of the screen. The navigation tree, including its appearance and operating mode, is realized in class CL_WB_TREE_DEFAULT_BRF. If no alternative class is specified, this class is used – even if the field is left blank. You do have the option of using another implementation of the tree class, for example, if the standard appearance or operating modes are unsuitable for a specific application class. You can enter each class that implements the interface IF_WB_BROWSERTREE_BRF as an alternative to the standard class.

As a next step you can go to transaction BRF and create the BRF object with similar name as Application class 0MYFIRST_BRF.

Events

Event maintenance in the BRF standard (BRFEVT01) comprises administrative data with short text and the following properties:

- Event : 0EVENT_SAMPLE
- Implementation Class: The implementation class will contain the Maintenance class and Implementation class.
  - Implementation class: Runtime class for the current BRF object. With the BRF object, this might be an event, expression, action, rule or rule set.
  - Maintenance Class: This is mainly used for the maintenance of sub-screens.

The table below gives information of the corresponding basis Implementation /Maintenance classes:

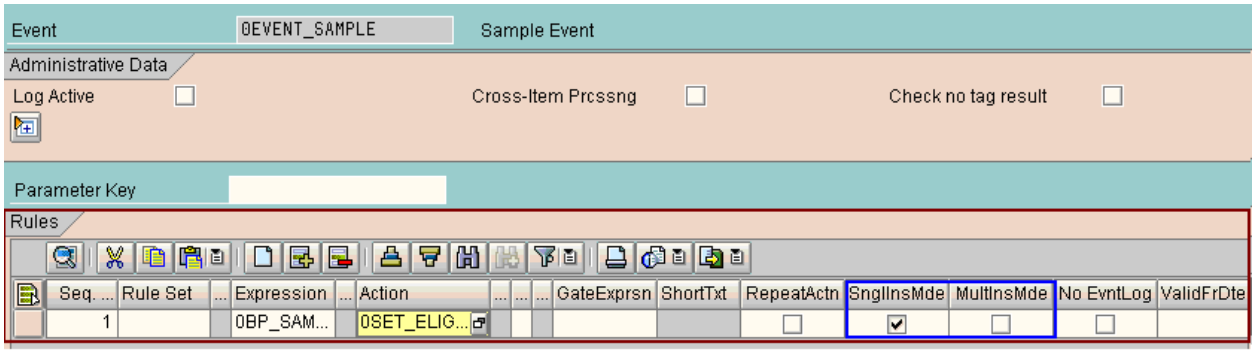
BRF Objects	Implementation Class	Maintenance Class
Expressions	CL_EXPRESSION_BASE_BRF	CL_EXPRESSION_BASE_MNT_BRF
Actions	CL_ACTION_BASE_BRF	CL_ACTION_BASE_MNT_BRF
Events	CL_EVENT_BASE_BRF	CL_EVENT_BASE_MNT_BRF
Rules		CL_RULES_BASE_MNT_BRF
Rule Sets		CL_RULE_SET_BASE_MNT_BRF

Note: User can come up with their own implementation or maintenance classes by using above mentioned classes as super-class. By doing so, user can control the behavior of BRF objects.

- Context: A BRF event is the carrier of the context. In the “Check Invoice Item“ event, the context is the invoice item. However, the BRF does not know how the context of invoice item is presented in the technical sense. To be able to at least name the context, you can use transaction BRF in the BRF and choose *Utilities -> Contexts* to declare contexts and enter one of them. You can then also check whether an expression that can be assigned to a context is used in the correct event.
- Controller class: The controller class realizes a rule. Rules are presented by class CL_CONTROLLER_BRF. This class is used by default if no other class is entered. If a different operating mode is required, you must realize another class and enter this. Like CL_CONTROLLER_BRF, this class must implement interface IF_CONTROLLER_BRF.
- Expression buffer: This setting enables you to control expression buffering. It can have the following values:
  - Expression buffering is active: An expression is calculated just once for each BRF event. The event is buffered and is then available to other BRF objects. The life span of the buffer is restricted to the life span of the event.
  - Expression buffering per rule line is active: The life span of the buffer is restricted to a rule line. If the same expression is needed in two rule lines of the same event, it is calculated twice.
  - Expression buffering is inactive. No buffering takes place.
- Determine assignment classes (rule definition classes): The BRF saves the rules in tables TBRF210 and TBRF310. However, the BRF cannot force the applications to store the rules in the tables mentioned. Many applications have to store the rules in tables of the application, because of the different key format or different attributes, for example. The determine assignment classes offer the option of determining rules for a specific event from any source and to convert them to a format that can be processed further by the BRF. The standard delivery classes CL_DET_ASSIGNMENT_RL_BRF and CL_DET_ASSIGNMENT_RS_BRF are the rule definition classes for rules in the event and for rules per event in rule sets that implement interface IF_DETERMINE_ASSIGNMENT_BRF. An application that saves its rules itself implements one or more such rule definition classes and adds them to the list of determine assignment classes or replaces the standard class(es).

Rules

Rules will be defined within the event main screen (Picture 4).



Picture 4: The Rule has been attached to the Event 0EVENT_SAMPLE. When expression returns result TRUE, the action will be executed.

Important notes on Rules:

1. Can define any number of rules for each Event.
2. Rules are also called as Subscriptions.
3. Rules can be differentiated as Real rule or False rule based on the expression and the action it contains.

    - Rule consisting of an **expression** and an (abstract) **action** ("Real rule")

      This is “real rule” is the most common one.

      It has the following appearance: if <expression> = TRUE then <action>. In other words, if a condition defined by the <expression> delivers the result TRUE, the assigned <action> is executed.
    - Rule that does not contain an **expression**, but an **action** (“False rule”)

      If you do not store an expression, the action is always executed as soon as the event is triggered.
    - Rule that contains an **expression**, but not an **action** (“False rule”)

      The expression is always calculated, but no action is executed.

      This rule only makes sense in the following case:

      If the application that calls the event expects a result to be returned.

Description of components involved in Rules maintenance:

1. Expression – name of expression. Expressions are BRF objects that return a definite result.
2. Action – name of action. Actions are BRF objects that start some activity as part of Rule Execution
3. Gate expression - Determines at runtime whether a rule really should be processed.
4. Single Instance mode - Apply Rule in Single Instance Mode (Dialog Mode)
5. Multi Instance mode - Execute Rule in Multi-Instance Mode (Batch Mode)
6. Rule Processing Termination - Rules processing can be terminated when there is a specific result of the relevant condition. This means that no other rules can be calculated at runtime.

Rule Processing Termination	Description
0	No Termination
1	Termination If Condition Is True
2	Termination If Condition Is False

7. Start/End Date and Start/End Time of validity of Rule - These specifications are static. This means that they are features of the rule that you define when you configure the rule.
8. Reference date and time - In the *Reference Date* and *Reference Time* fields you can store a BRF expression with result type D (for the reference date) and/or with result type T (for reference time).

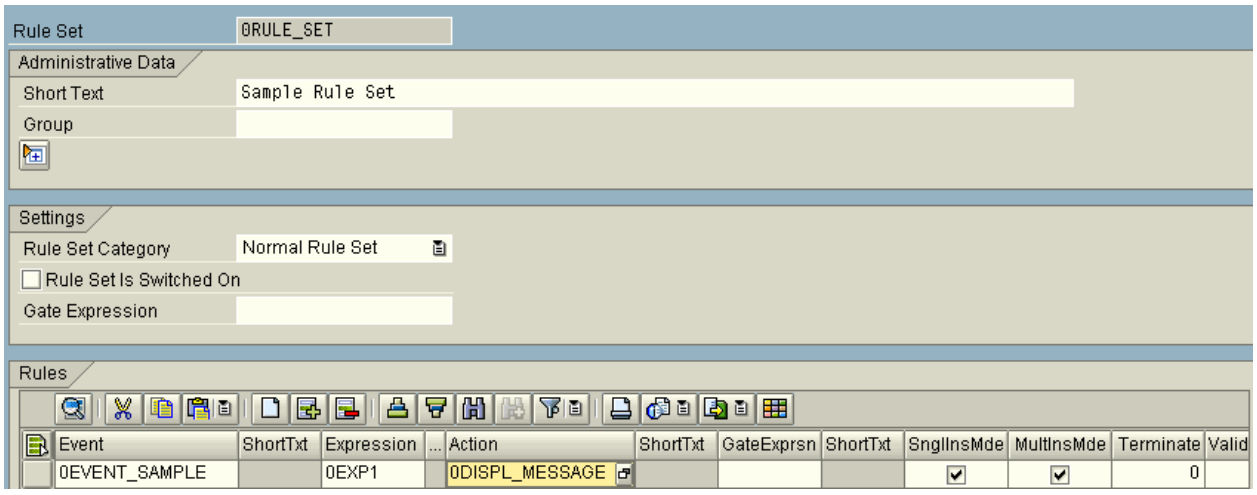
Rule Sets

Transaction BRFRLS01 is used to create a rule set.

Components involved in Rule set maintenance:

1. Rules in the rules sets can be activated or deactivated together.
2. Event – name of the event
3. Expression – name of the expression
4. Action – name of the action
5. Gate expression - Gate expression determines at runtime whether a rule really should be processed. You can only define an expression as a gate expression if it delivers a Boolean value (return value with category B).
6. Single Instance mode - Apply Rule in Single Instance Mode (Dialog Mode)
7. Multi Instance mode - Execute Rule in Multi-Instance Mode (Batch Mode)
9. Start Date/End Date/Start Time/End Time of validity of Rule set - These specifications are static. This means that they are features of the rule that you define when you configure the rule.
8. Reference date and time - In the *Reference Date* and *Reference Time* fields you can store a BRF expression with result type D (for the reference date) and/or with result type T (for reference time).

If the expression name is not entered, the rule is processed normally.



Actions

Transaction BRFACT01 is used to create actions.

Actions can be of following types

Implementation Class	Short Text
0CRMBD001	CRM: Call of a Business Add-in
0CRMEV001	CRM: Execute BRF Event
0CRMMS001	CRM: Simple Message Output
0CRMST001	Set CRM Status
0FM001	Run Function Module
0MS001	Message Output via BDT (Only makes sense to use in BDT Environment)

Note: The function modules and BAdi methods used in BRF should have the common interface.

The message executing action may also contain the expressions. So this in turn executes an expression attached to it.

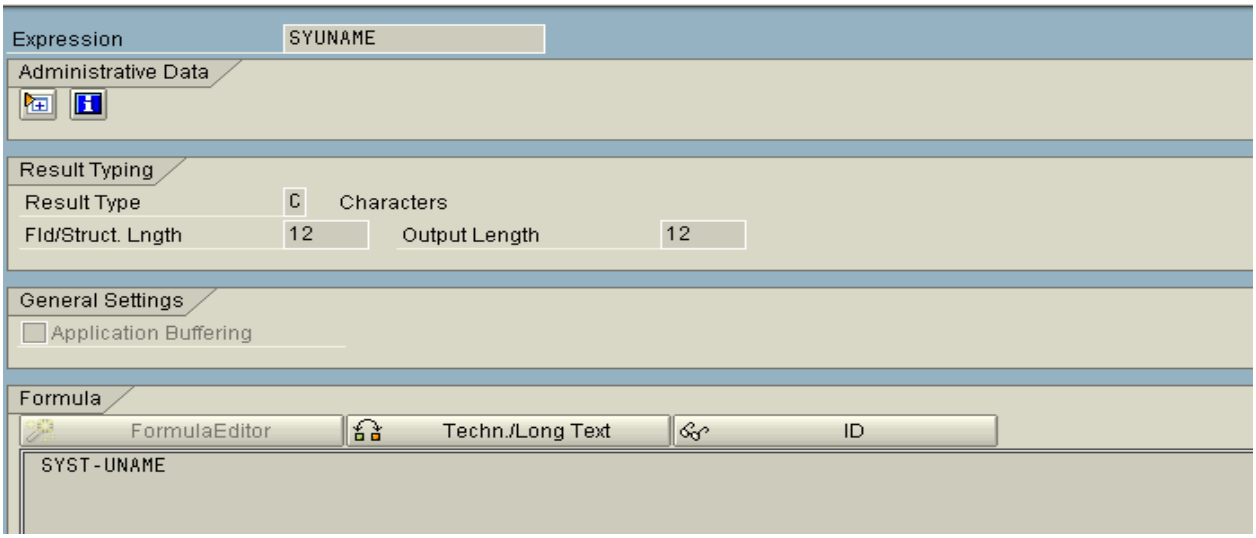
Expressions

Transaction BRFEXP01 is used to create BRF expressions.

Expressions can be constant, case expressions, Boolean expressions, function module BAdi, formula interpreter, simple formulas, truth table, ranges or structure fields etc. Each of these types will have a separate implementation class like shown in below table:

Implementation Class	Short Text
03O001	Boolean Expression with Three Variables
0AO001	AND/OR Conditions of Boolean Expressions
0CE001	CASE Expression
0CF001	Function Module BAdi as Expression
0CN001	Constant
0DI001	Date interval
0FB001	SAP Formula Interpreter
0FR001	Simple Formulas
0PM001	Truth Table
0RE001	Range Expression
0RF001	Reference-Like Expression
0RV001	Random Number Generator
0SM001	Field of a Structure/Simple Value Request
0TB001	Field of a Line of an Internal Table/Select-Like Value Request

An example of formula expression is shown in below picture:



BRF at Runtime

This chapter explains the runtime behavior of all the BRF objects.

Architecture of the BRF at Runtime

The runtime environment of the BRF is a framework that is realized in ABAP-OO (ABAP Object-Oriented).

Before a class can participate in the BRF at runtime, you must implement an interface.

You can implement this interface in one of the following ways:

- Directly

  However, this usually means extensive implementation effort.
- Indirectly

  Derive the interface from a suitable basis class.

  “Suitable” in this context means that you derive the interface for an event from an event basis class, for an expression from an expression basis class, and for an action from an action basis class.

  Consider the following:
  - Each BRF runtime class implements the interface IF_RULE_COMPONENT_BRF.
  - In addition, events implement the interface IF_EVENT_BRF, expressions the interface IF_EXPRESSION_BRF, and (concrete) actions the interface IF_ACTION_BRF.
  - The relevant basis classes CL_EVENT_BASE_BRF, CL_EXPRESSION_BASE_BRF and CL_ACTION_BASE_BRF implement the relevant interfaces themselves.

A complete division of the software layers **object** and **database** has not been realized.

A GUI (Graphical User Interface) layer does not occur at runtime.

Calling an Event in the coding

Using a sample source text (BRF_DEMO_RAISE_EVENT_SAMPLE) from the ”Flight Demonstration“ example, the following describes how you call an event in your application.

Note: Only the main parts of the source text of BRF_DEMO_RAISE_EVENT_SAMPLE are mentioned.

The event call is split into five parts:

1. In the first part, the data declaration takes place.

   Sample source text:

```
* Step 1: data declaration
 DATA:
        lo_controller     TYPE REF TO cl_controller_brf,
        lo_event          TYPE REF TO cl_event_base_brf. "if_event_brf.
```

2. In the second part, you create a controller object for the current application class.

   Sample source text:

```
* Step 2: get an instance of the BRF controller
  CALL METHOD cl_controller_brf=>get_controller
    EXPORTING
      iv_applclass  = iv_applclass
    IMPORTING
      eo_controller = lo_controller
    EXCEPTIONS
      OTHERS        = 1.
```

3. In the third part, you request the controller object obtained in point 2 to provide an event object with event ID IV_EVENT.

   Sample source text:

```
* Step 3: get an instance for the BRF-event
  CALL METHOD lo_controller->get_event
    EXPORTING
      iv_event = iv_event
    IMPORTING
      eo_event = lo_event
    EXCEPTIONS
      OTHERS   = 1.
```

4. In the fourth part, the event is called.

   The method PROCESS_EVENT of the CONTROLLER object is called, and the current event object is provided.

   Sample source text:

```
* Step 4: process the event
  CALL METHOD lo_controller->process_event
    EXPORTING
      io_event = lo_event
    EXCEPTIONS
      OTHERS   = 1.
```

5. In the fifth part, both the controller and the event are returned. The existing references are deleted implicitly upon completion of the function module.

   Sample source text:

```
* Step 5: free objects
  CALL METHOD lo_controller->free.
```

```
CALL METHOD lo_event->free.
```

After processing of the event, and before the FREE methods, you can also perform one of the following activities:

- You can pick up collected messages from the event object
- Depending on the implementation, you can transfer to the application other return values stored in the event during event processing.

Note: Take a look at function module BRF_DEMO_RAISE_EVENT.

Expression buffering

Within an event, you can re-use several expressions. The expression buffer saves each calculated expression so that when requested again, the expression is available immediately and does not have to be calculated completely from scratch again.

Variants of Reuse

The following different reuse variants exist:

- Variant 1:

  The expression is already in the expression buffer for the current event and has a valid result.
- Variant 2:

  The system did not find any object that has the same expression type as the requested expression. This object is then reused and calculated again.
- Variant 3:

  The expression is created for the first time and stored in the expression buffer since the system found no matching expression or object that could be reused.

Definition of Expression Buffering

In event maintenance you define whether expression buffering should be active.

You have the following options:

- Expression buffering is active

  This means that all expressions in an event are reused (if this is possible).

  At the end of the event, the results of the expressions are invalidated, but the objects are reused in the events that follow.
- Expression buffering per rule line is active

  This means that all expressions are only reused in a rule line.

  At the end of each rule line, the results of the expressions are invalidated, but the objects are reused in the rule lines and events that follow.
- Expression buffering is inactive

  This means that the expression buffer has been deactivated. No expressions are saved in the expression buffer.

Runtime Class for Expressions

If you want to realize an own expression type, you must also implement a runtime class in addition to the maintenance class and the associated sub-screen.

Note: It is recommended that you implement your runtime class on the basis of basis class CL_EXPRESSION_BASE_BRF.

If you implement your runtime class on the basis of CL_EXPRESSION_BASE_BRF, you only have to redefine the following methods:

- INITILIZE

  The INITIALIZE method has the following effect:

  - It gets the following information from table TBRF130:
    - what type of link of the three operands is involved
    - which operands are involved
  - It calls its super method so that the following information can be imported:
    - The required information from table TBRF150 to table MS_BRF150 (table of all expressions and their joint features)
    - The required information from table TBRF260 to table MS_BRF260 (table of all operands)

  Method can be enhanced to fulfill the requirement.

- PROCESS_SPECIFIC

  At the start, the GET_RESULT method for all sub-expressions is called. The basis class has already provided for their calculation (PROCESS_SUBEXPRESSION method).

Note: It is recommended that you perform a calculation only for the operands that are really needed. For example, if the result is available prematurely, no further operands need to be calculated.

Redefine the PROCESS_SUBEXPRESSION method.

The further procedure depends on the concrete requirements regarding functionality of the class. In class CL_EXPRESSION3_BRF, three operands are linked according to the customizing setting, and the result is determined.

The result variables are then filled accordingly, and the result is set to “Valid” to avoid repeated calculation.

The other methods have already been implemented by the basis class. This means that you do not have to redefine them.

Note: In the description of the methods INITIALIZE and PROCESS_SPECIFIC, class CL_EXPRESSION3_BRF (for the expression type Three Operand Arithmetic) serves as an example. This class is derived from basis class CL_EXPRESSION_BASE_BRF. You can use class CL_EXPRESSION3_BRF to calculate the logical link of up to three Boolean expressions.

Runtime Class for Actions

If you want to realize an own action type, you must also implement a runtime class in addition to the maintenance class and the associated sub-screen. In principle, this is done in the same way as with the runtime class for expressions. This means that you must also redefine the following methods:

- INITIALIZE

  The method calls the INITIALIZE method of the super-class so that the following information can be imported:

  1. The required information from table TBRF150 to table MS_BRF150 (table of all expressions and their joint features)
  2. The required information from table TBRF260 to table MS_BRF260 (table of all operands)

  Method can be enhanced to fulfill the requirement.

- PROCESS_SPECIFIC

The PROCESS_SPECIFIC method implements the specific behavior of the runtime class according to the settings imported in Method INITIALIZE.

Note: You can use class CL_AC_FUNCTION_BRF as a template for your own development. This class realizes the action type Execute Function Module (implementing class 0CF001).

Runtime Class for Events

At runtime you can use the standard runtime class for BRF events, namely CL_EVENT_BRF. This means that an own implementation is not necessary.

However, if you want to provide your event or event class with context information that is not known to the BRF, you must implement an own runtime class for events.

Consider the following points:

- Rule Definition

  Rule definition involves determination of rules that should be processed in the current event. In the BRF, the determined rules are stored as follows:

  - Rules (without rule sets) in table TBRF210
  - Rules from rule sets in table TBRF310

  Until now, the rules were determined by the DETERMINE_ASSIGNMENT method of runtime class Events.

  As of Release 700, rules are determined by a new, much more dynamic mechanism.

  Rule definition looks like this as of Release 700:

  Rule definition takes place in the following classes that implement interface IF_DETERMINE_ASSIGNMENT_BRF:

  - Class CL_DET_ASSIGNMNT_RL_BRF for rules (without rule sets)
  - Class CL_DET_ASSIGNMNT_RS_BRF for rules from rule sets

Note: Take a look at these two classes in your system.

  These two classes form the basis for you being able to parameterize rules further. For example, in the Claims Management system, the insurance line of business is one such parameter. You can therefore specify that a rule should only be processed in the line of P&C insurance, but not in the line of health insurance.

  The insurance line of business is a parameter that the BRF does not know. You therefore have to save rules that are only valid for a certain line of insurance in tables of the application (in this case, in the *SAP Claims Management* application).

  SAP Claims Management therefore contains the class DETERMINE_ASSIGNMENT.

- Event with context Information

If you want to provide the event with context information, you must define an interface. This interface must contain at least one method of any name that accepts the context information (in this example, the interface IF_BRF_EVENT_FLIGHT with the method SET_CONTEXT).

You can include the context information in the interface as a member variable (structure, table, and object). The advantage of this is that the context can be (read) accessed without the need to explicitly specify a read method (MS_FLIGHT_CONTEXT).

The own event class (in this example, CL_EVENT_FLIGHT) is derived from the basis class CL_EVENT_BASE_BRF. In addition, it implements the previously defined interface (IF_BRF_EVENT_FLIGHT).

Sample source text:

```
METHOD if_brf_event_flight~set_context.
  mv_flight_context = iv_flight_context.
ENDMETHOD.
```

Glossary

ABAP

Advanced Business Application Programming. SAP's own programming language for developing application programs.

ABAP OO

ABAP Object Orient Programming. SAP's own object oriented programming language for developing application programs.

ALV

ABAP List View

Action

Actions are BRF objects that start some activity as part of Rule Execution

Application class

Application class identifies an application that uses the BRF (Business Rule Framework).

BRF

The Business Rule Framework is an event-controlled runtime environment for processing rules.

BRF Objects

Events, rules, expressions or actions form the BRF Objects

Change Request

An information source in the Transport Organizer that records and manages all alterations made to Repository objects and Customizing settings during a development project.

Client

A client usually represents a company in an SAP system. This means that if an SAP system has several clients, then several companies can be represented and simultaneously active in that system. The client has a corresponding key field in the tables of the database for that SAP system. If you are logged on to a specific client, then you can only access data for that client. Clients therefore correspond to independent business entities.

Customizing

Customizing is the overall procedure for setting up one or more SAP systems. This procedure is directed toward adapting the standard, industry-specific SAP system functions to a company's particular business requirements. Customizing is obligatory both during the first installation and during an upgrade and is performed in the SAP system using the Implementation Guide (IMG).

Data element

ABAP Dictionary object that describes the data type and semantic meaning of a table field or structure field.

Database interface

Component of a work process that connects it to the database. The database interface translates Open SQL into database-specific SQL, thereby enabling communication with the database.

Domain

An ABAP Dictionary object that describes the technical attributes of a data element, such as data type, length and value range. You can group fields that have similar technical or business purposes under a single domain. All fields based on a domain are updated automatically when you change the domain. This guarantees the consistency of the fields

Event

Events may be considered the "Entry point/gateway" to Rule Evaluation and Execution in BRF

Expression

Expressions are BRF objects that return a definite result. Boolean Expressions form the “IF” part of the Rule.

Gate Expression

Gate Expression is a Boolean expression that determines at runtime whether a rule or an entire rule set really should be processed. A gate expression must always be Boolean. Other result categories are not permitted

PAI

Process After Input. PAI is a processing block in the screen flow logic that is executed after the screen is displayed. This processing block calls modules in ABAP programs and determines the processing that is required after a user action on the screen.

PBO

Process Before Output. Block of code that is processed after a screen is called but before it is actually displayed.

Repository

Central store for all ABAP Workbench development objects. The development objects stored in the SAP system Repository include: program objects, function group objects, Dictionary objects, Business Engineering objects and other objects.

Rule

Rules are assigned to an event and will be processed in the order they were created

Rule Set

Grouping of rules that belong together from a business point of view, and that are distributed over several events

SAP Easy Access

SAP Easy Access is the default initial screen in SAP systems. The left side of the screen contains a tree hierarchy of the menus available to you in the SAP system; you can use the right side of the screen to display a graphic, such as your company logo.

SAP GUI

SAP Graphical User Interface; medium that enables the user to exchange information with the computer. The user interface allows you to select commands, start programs, display files and execute other options by pressing function keys or pushbuttons, or by selecting menu options.

SAP transaction

An SAP transaction describes a logically complete action in an SAP system. From the user’s point of view, a transaction represents a unit (for example, creating a list of a certain type of customer, changing a customer’s address, creating a flight reservation for a customer, or executing a program).

Table

Tabular array of data in the ABAP Dictionary. A table consists of columns (data values of the same type) and rows (data records). Each record can be identified uniquely by one or more fields.

Table Control

Table control is an Area on a screen that enables you to enter and display one-line tabular data efficiently.

The data is processed in a loop at runtime. There can be several table controls on one screen and, as with sub screens, each has a unique name.

The main features of a table control are as follows:

- Vertical scroll bar

- Horizontal scroll bar
- One-line table (there is no equivalent of multiple-line loop blocks, as found in step loops)

Table controls are similar to step loops, but have greater functional scope. For this reason, they can be regarded as enhanced step loops.

Transaction Code

A transaction code (also known as a TCode) is a sequence of characters that identifies a transaction in the SAP system. A transaction code may contain up to 20 characters and must always begin with a letter. Permitted characters are letters from A to Z, numbers from 0 to 9, and the underscore. To call a transaction, enter the transaction code in the command field and choose Enter.

View

Virtual table that contains no data, but is an application-specific view of one or more tables in the ABAP Dictionary.

Related Content

BRF Objects

BRF

BRF Configuration for Sample Processes

Copyright