BRFplus Application Exits

Applies to:

Business Rule Framework plus, shipped with NetWeaver 700 Enhancement Package 2 and NetWeaver 720.

Summary

The Business Rule Framework plus (BRFplus) inherently incorporates many possibilities to adapt business rules to custom needs. There is, however, always a limit in the functional scope that standard software can cover. One way to overcome such unavoidable boundaries with BRFplus is the concept of application exits.

The paper gives an introduction on how the capabilities of the Business Rule Framework plus can be enhanced by using application exits.

Author: Thomas Albrecht

Company: SAP AG

Created on: July 25, 2009

Last update: July 25, 2012

Author Bio

Thomas Albrecht is a developer in the team of Business Rule Framework plus. Since joining SAP in 2003, he has worked in and driven several development projects in ERP Travel Management with focus on architecture. In 2008, he joined the BRFplus development team.

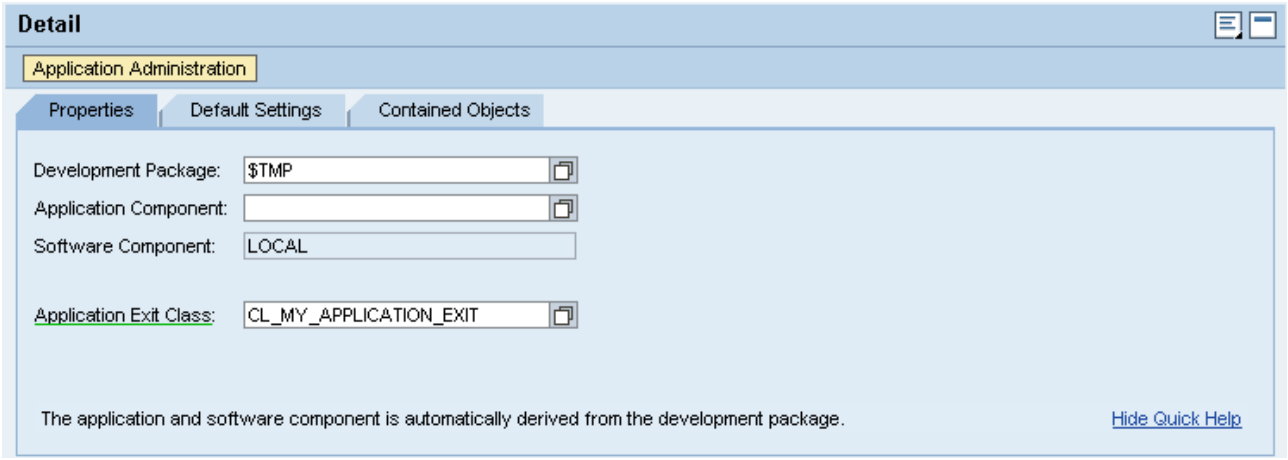
Table of Contents

What is an Application Exit?

In BRFplus, all objects like functions, rulesets, expressions, data objects, etc. are created within an application. The application is considered to represent a logical unit of rule definitions, similar e.g. to a development project in common programming environments.
The application defines some standard settings for all of its contained objects, like the storage type and package assignment. With the concept of application exits, BRFplus allows to extend its common features in the scope of an application and its contained objects.

For example, it is possible to restrict the access to BRFplus objects with a customer-specific authorization-check mechanism with a finer granularity. The validity of data entered in an expression may need to be cross-checked against some external data sources. Or one simply wants to be notified in case a BRFplus object is saved so that some dependent adaptations or other post-processing can be started.

An Application Exit is technically based on an ABAP class that implements the specific interface `IF_FDT_APPLICATION_SETTINGS`. You can assign one such class for each application, using method `IF_FDT_APPLICATION~SET_SETTINGS_CLASS()`. This may of course also be achieved via the workbench UI.



General Implementation

The interface `IF_FDT_APPLICATION_SETTINGS` defines a set of static methods – the actual Application Exits. For each method/exit a corresponding boolean attribute `GV_<method name>` is defined by the interface. The attribute is used to indicate whether the interface-method is actually **implemented** and the exit shall be used. **An exit-method is called by the BRFplus framework only, if the corresponding boolean attribute has been set to `true`.** Setting the attributes to **`true`** is best done inside a static class constructor, as in the following example:

```
METHOD class_constructor.
  if_fdt_application_settings~gv_authority_check = abap_true.
  if_fdt_application_settings~gv_get_element_values = abap_true.
ENDMETHOD.
```

Depending on the triggering event and the called method's interface, the application exit may influence the behavior of BRFplus' object handling. Since the currently executed program exits the realm of BRFplus in this instant, it is in the responsibility of the programmer to avoid any kind of runtime errors.

Available Application Exits

The following application exit methods are currently available in the interface:

Application Exit Method	Description
AUTHORITY_CHECK	If an object is accessed or shall be manipulated, BRFplus performs an authority check first. This exit allows to enhance or even completely replace the default authority checks of BRFplus.
CHECK	This method is called whenever a BRFplus object is checked for consistency. Additional checks may be implemented within this exit. The exit also affects the possibility to activate an object. (see below)
ACTIVATION_VETO	Before a BRFplus object can be activated, it needs to be in a consistent state and is therefore checked accordingly. With the activation veto exit it is possible to prohibit the activation also due to other aspects like external dependencies, that might even only be of temporary nature.
CHANGE_NOTIFICATION	This exit sends a notification if an object is changed. This might e.g. be used for an automatic update of a dependent BRFplus object.
SAVE_NOTIFICATION	This exit simply gives a notification, if an object is saved. This might e.g. be used for an automatic update of a dependent BRFplus object.
GET_ELEMENT_VALUES	With this exit it is possible to define a dynamically retrieved list of valid values for a data element in BRFplus.
GET_FORMULA_FUNCTIONALS	With this application exit the functions that are available for formula expressions can be set. It is possible to restrict the list of standard functions but also to add custom defined functions .
GET_CHANGEABILITY	With this application exit the changeability of customizing objects can be altered.
GET_CALENDAR	This application exit can define the usage of a specific calendar for date and time related operations. The exit is currently not relevant for NW70x releases. It is only used for special application platform systems based on NW710 or higher.

For all application exits, exceptions of type CX_FDT_SYSTEM can be propagated or created, in case a severe error occurs. This will either result in a failure, e.g. in case of the CHECK application exit; or the application exit will simply be ignored, like in the case of application exit CHANGE_NOTIFICATION.

A `CX_FDT_SYSTEM` exception is also automatically raised, if BRFplus detects recursive calls to one application exit for the same object (more than 3 times). It should e.g. be avoided to change the object passed to a CHANGE_NOTIFICATION exit, which would cause yet another call of the same application exit.

Let‘s have a look into the details of each application exit now, including its signature definition.
BRFplus uses some naming conventions for method parameters. You should be aware of the following, most basic conventions:

- **Importing parameter names start with an i**
- **Exporting parameter names start with an e**
- **Changing parameter names start with a c**
- **Returning parameter names start with an r.**

Application Exit AUTHORITY_CHECK

Access to BRFplus objects can currently be restricted per application on the level of the object type and the current activity type. Object types are e.g. functions, expressions, data objects, etc. The activity type includes the display, change, creation, deletion and activation of an object.

Standard authority checks

Technically, the authorization object FDT_OBJECT is used for the standard authorization checks. It includes the following fields:

- `FDT_APPL` – ID of the BRFplus application
- `FDT_OBJTYP` – the object type. The most important object types you find in the list below. All available object types are defined as constant definitions in the class interface `IF_FDT_CONSTANTS`, starting with `GC_OBJECT_TYPE_<…>`.
- `FDT_ACT` – The activity type. The current activity types are listed below. All available activity types are defined as constant definitions in the class interface `IF_FDT_CONSTANTS`, starting with `GC_ACTIVITY_<…>`.

Object Type	Value
Application	AP
Expression Type	ET
Expression	EX
Data Object	DO
Ruleset	RS
Function	FU
Catalog	CA

Activity Type	Value
Create	1
Change	2
Display	3
Delete	4
Activate	5

Programmatically the authorization check can be done with public method `IF_FDT_TRANSACTION~AUTHORITY_CHECK`, that is implemented by all BRFplus objects. It has the following signature:

- `IV_ACTIVITY` – activity type to be checked
- `IV_DEEP` – indicates whether or not the authority check is to be performed for all directly or indirectly referenced objects
- `RV_PASSED` – indicates whether the check was successfully passed

Exit Details

If the AUTHORITY_CHECK application exit is implemented, it is called **before** the standard authorization check is performed.

The importing parameters of method `IF_FDT_TRANSACTION~AUTHORITY_CHECK` are simply forwarded to the application exit, together with the ID of the object to check.

The result of the custom check is returned in a boolean parameter `ev_passed`. In addition, the standard authorization check mechanism can be skipped either completely or only for referenced objects.

A message may optionally be defined, that can be read from system structure `SYST` after the authorization check has been performed.

Signature

The AUTHORITY_CHECK application exit has the following signature:

- `IV_ID` – ID of the object to be checked
- `IV_ACTIVITY` – activity type to check authorization for
- `IV_DEEP` – perform the authorization check also on all directly or indirectly referenced objects
- `EV_PASSED` – indicate whether the check was passed successfully
- `ES_MESSAGE` – a message that can be read from system structure `SYST` afterwards
- `EV_SKIP_CHECK` – indicates whether standard checks shall be skipped
- `EV_SKIP_REFERENCED` – indicates whether standard checks for referenced objects shall be skipped

Application Exit CHECK

All objects in BRFplus include some standard checks that should guarantee that the object is consistently defined and thus can be processed at runtime without errors. An object can therefore only be activated if the checks have been successfully passed. Inconsistent objects may still be saved in inactive state.

The CHECK application exit allows the definition of additional checks, e.g. due to some customer-specific restrictions, cross references or relations to external data. It is executed when the standard checks have already been successfully passed. If the additional checks fail for an object, it cannot be activated (regardless of the result of the standard checks).

The application exit is intended only for checks that are based on the object's properties: Users defining the object should be able to correct the properties in case the check fails.
If the additional checks are not related to object properties and e.g. depend on some temporary constraints, they should rather be implemented in the ACTIVATION_VETO application exit.

Exit Details

The application exit's signature is quite simple. The ID of the object to check is passed as input. Also, the object's type is passed (function, expression, data object, etc.).

The application exit may or may not perform additional checks on the passed object. The outcome of the additional checks is a list of corresponding messages. If one of the messages is of type error ('E'), the check fails.

Signature

- `IV_ID` – ID of the object to be checked
- `IV_OBJECT_TYPE` – type of the object to be checked
- `ET_MESSAGE` – a list of messages, assembled by the additional checks

Application Exit ACTIVATION_VETO

When a BRFplus object is activated, it should be ready for processing. This is normally ensured by standard consistency checks or even additional custom checks (see CHECK application exit).

The ACTIVATION_VETO application exit allows to veto the activation of a BRFplus object for additional reasons. This may e.g. be some dependent system settings; or there might exist interdependencies to other BRFplus objects, even though they do not reference each other.
Note that for authorization related checks the AUTHORITY_CHECK exit with activity type ACTIVATE should be used.

Exit Details

The application exit's design is similar to the one of the CHECK application exit: The ID and type of the object to activate is passed. The exit allows however to veto the activation without a specific message.

If an exception occurs, it is treated like a veto and the activation fails.

Signature

- `IV_ID` – ID of the object to activate
- `IV_OBJECT_TYPE` – type of the object to activate
- `EV_VETO` – prohibits the activation if set to `true`
- `ET_MESSAGE` – a list of optional messages in case the activation is vetoed

Application Exit CHANGE_NOTIFICATION

This application exit is called to notify whenever an object's property is changed. There is also the special case that all previously done changes are discarded, either explicitly or if the object is unlocked (dequeued) without prior saving.

The implementation of the application exit might for example adjust depending objects. Discarding of changes would need to be propagated, including unlocking of the dependent objects.
Be aware, however, that you cannot rely on changing and saving other BRFplus objects, because they might already be locked externally.

Exit Details

The ID and type of the object that was changed is passed. An additional flag is set in case the change was actually a discarding of previous changes.

If an exception occurs it is simply ignored.

Signature

- `IV_ID` – ID of the changed object
- `IV_OBJECT_TYPE` – type of the changed object
- `IV_CHANGES_DISCARDED` – indicates whether all changes of the object have been discarded

Application Exit SAVE_NOTIFICATION

This application exit is called to notify whenever an object is saved. Be aware that in the application exit implementation, you must not alter the object itself! Rather implement additional checks in the CHECK or ACTIVATION_VETO exits for that purpose.
Also you may not rely on changing or saving other objects, because they might e.g. be locked externally. Eventually you need to consider that changes on used objects might not get refreshed accordingly in the UI when you change them in the background.

Exit Details

The ID and type of the object that was saved is passed.

Signature

- `IV_ID` – ID of the saved object
- `IV_OBJECT_TYPE` – type of the saved object

Application Exit GET_ELEMENT_VALUES

Standard data element definitions in BRFplus allow to specify a static list of valid values, including description short texts for the values. In case the element is bound to a DDIC[1] type or a GDT[2], the value list is derived automatically from the respective object. DDIC types may e.g. have a set of domain values or reference a value table.

For non-bound data elements, this application exit allows to dynamically retrieve a list of valid values, e.g. via the selection of some database table entries.

The application exit may be called for different purposes. The most simple case is to check if a specific element has a value list at all. Sometimes only some specific entries of the value list are required, e.g. to retrieve the short text descriptions. Last but not least the complete value list might be requested.

Exit Details

Each time the value list of an element is accessed, the application exit is called with the ID of the element. With an exporting parameter the exit first of all should indicate if it is applicable for the specific element (i.e., the exit implementation must keep a list of those data objects which are covered by the exit and set the parameter accordingly).

If the application exit is applicable, an optionally passed timestamp can be used in case the element is version-dependent and thus time-dependent. If the parameter is not passed, the latest version/current time should implicitly be used.
For some specific use cases, additional information about the runtime context may be passed, such as the object that requests the value list for the element. E.g., The value list might then be reduced for specific scenarios.

The application exit may indicate that the element does not have a value list to be checked at all. Otherwise, it should check if the value list is actually requested, due to performance reasons. If so, it should return the complete list of values and texts by default.

By passing an optional range selection for values and short texts, the caller may request only specific entries of the value list. In this case, the returned value list must be filtered by the exit accordingly.

To allow a performant implementation, the filtering is not automatically done in a generic way. E.g., if the value list is retrieved from a database table, the filtering could already be embedded into an according SQL-SELECT statement.

Signature

- `IV_ID` – Element ID
- `IV_TIMESTAMP` – optional timestamp for version-dependent elements
- `ITR_VALUE` – optional range of values for filtering
- `ITR_TEXT` – optional range of description texts for filtering

---

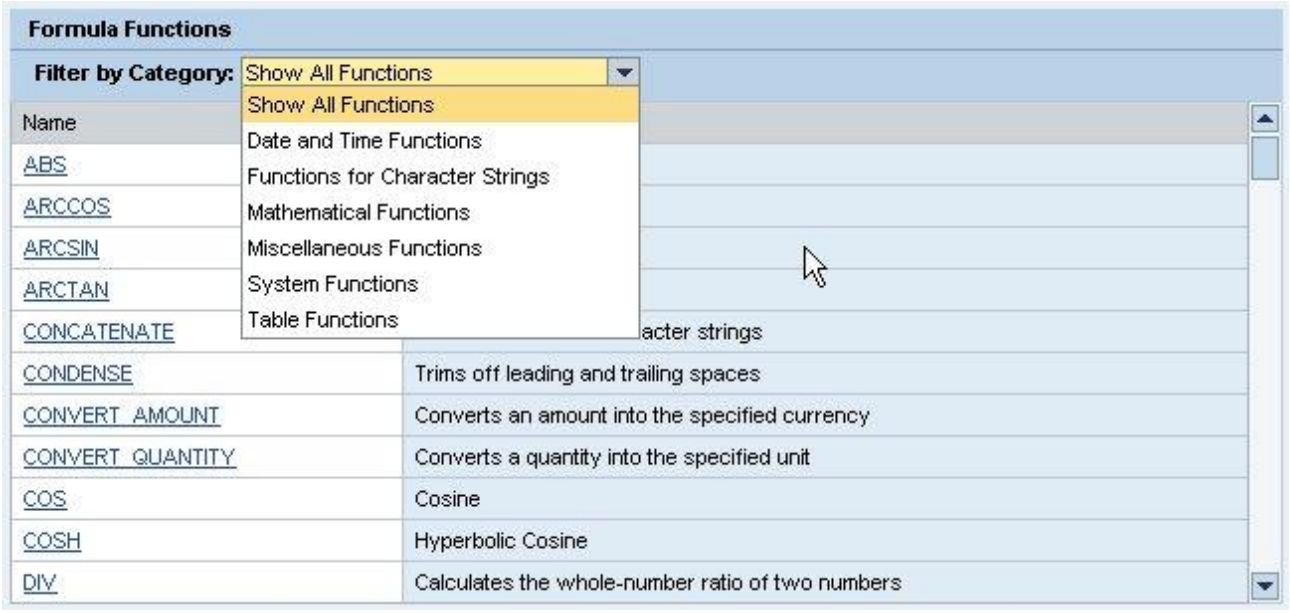
[1] ABAP **D**ata **Dic**tionary

[2] **G**lobal **D**ata **T**ype in Enterprise Service Repository

- IV_LANGU – optional language to use for description texts
- IO_SELECTION[3] – optional context for the value selection
- EV_APPLICABLE – indicate whether or not the application exit is applicable for the element
- EV_NO_CHECKLIST – indicates that no value list is available
- ET_VALUE – the actual (filtered) value list

Application Exit GET_FORMULA_FUNCTIONALS

The standard formula expression type comes along with a predefined set of functions, that can be used within formula expressions by default. The functions are assigned to various categories, as can be seen in below screenshot snippet of the formula expression UI.



With the application exit it is possible to influence the list of available functions in general but also for specific formula expressions only. The implementation can filter out standard functions and add new, customer-defined functions. These may be assigned to an own category.

[3] The parameter IO_SELECTION is mainly used in conjunction with UI interaction, and especially in the case of decision table definitions. The parameter is of type CL_FDT_SELECTION_PARAMETER that currently provides the following set of parameters:

- MV_EXPRESSION_TYPE_ID – type of the expression requesting the element's value list
- MO_CALLER_OBJECT – a reference to the using element
- MS_DECISION_TABLE_ATTRIBUTES – in case the calling object is a decision table, further information like the currently edited cell position (row/column) is provided.

Further new attributes may be included into CL_FDT_SELECTION_PARAMETER in the future to cover additional requirements.

Exit Details

The ID of the currently handled formula expression is passed to the application exit. This allows the implementation to consider single formula instances.

A prefilled list of standard functions and function categories is passed as changing parameters. The list thus becomes completely editable within the application exit.

There's a paper "How to Create Formula Functions" available, that describes the creation of custom formula functions in detail. It contains also a step-by-step example how to create an according application exit. See the *Related Content* section.

If an exception occurs it is ignored. The list of available functions is not altered.

Signature

- `IV_ID` – ID of the formula expression
- `CT_FUNCTIONAL_DEFINITION` – list of available functions to be adapted
- `CT_FUNCTIONAL_CATEGORY` – list of available function-categories to be adapted

Application Exit GET_CHANGEABILITY

The changeability exit allows overruling the client settings for the changeability of **customizing** objects in customer systems. Also, the automatic recording of changes on transport requests can be adapted. However, when transportable customizing objects become editable in this way, changes may get lost with the next transport from a customizing system or a client copy.

Exit Details

For each object that shall be changed, BRFplus sends a request to the Change and Transport System (CTS) to determine whether the system allows changes and possibly requires recording on a transport request. Usually, only the object's application must be checked, because it determines the storage type of the object. As opposed to this, for customizing objects, the default settings can be overridden with the exit method also on object level.

If (objects of) a BRFplus application are not changeable, changing parameter `CV_CHANGEABLE` is initially `ABAP_FALSE`, and parameter `CT_MESSAGE` typically contains one or multiple error messages. In order to make the object changeable, `CV_CHANGEABLE` must be set to `ABAP_TRUE` and error messages must be deleted from `CT_MESSAGE`.

Signature

- `IV_APPLICATION_ID` – ID of the application, in case the exit class is used for multiple applications
- IV_ID - ID of the object that shall be changed.
- `CV_CHANGEABLE` – Indicates whether the default changeability shall be overruled
- `CV_CHANGE_RECORDING` – Indicates whether the default change recording settings shall be overruled
- `CT_MESSAGE` – Allows appending or deleting messages in connection with the changeability settings.

Application Exit GET_CALENDAR

This exit can define the usage of a specific UDTM[4] calendar instance for date and time specific operations. It may thus e.g. override standard settings for the local timezone, first day of the week, etc.

[4] Universal Date Time Model

The application exit is currently only relevant for NW71x or higher releases. It is also only used for special application platform systems.

Exit Details

The application exit simply may return the UUID of a desired calendar instance. The calendar instance is of type `/ISDT/CL_UDTM_CALENDAR` and will be retrieved by calling method `/ISDT/CL_UDTM~GET_CALENDAR()`. Check this class and method for further details.

Signature

`EV_CALENDAR_UUID` – ID of the calendar instance to use.

Related Content

BRFplus – The Very Basics

Using BRFplus with Third-Party Rules Engine

Carsten Ziegler, About Business Rules

Carsten Ziegler, BRFplus a Business Rule Engine written in ABAP

Carsten Ziegler, Important Information for Using BRFplus

Rajagopalan Narayanan, Business Rules and Software Requirements

Rajagopalan Narayanan, Seven Tips for Your First Business Rules Project

Rajagopalan Narayanan, Real World Return of Investment Scenarios with Business Rules Management

Rajagopalan Narayanan, Five Reasons to Build Agile Systems Using Business Rules Management Functionality

Rajagopalan Narayanan, How Business Rules Management Functionality Helps Tariff Plans Management in Transportation and Shipping

Rajagopalan Narayanan, Getting Started with Business Rules Management

Online Help

How to create Formula Functions“ by Carsten

Copyright