

rulerunner[®] framework

Release 0.9

- User guide -

Author: Derk Rösler

Document version: 1

revision no. 202

Table of contents

1 What is it all about?.....4

 1.1 Target group of this document.....4

 1.2 Basic idea from a business perspective.....4

 1.3 What is the rulerunner® framework?.....5

 1.4 Rulerunner® is free.....5

2 Installation of the rulerunner® framework.....6

 2.1 Prerequisites.....6

 2.2 Installation using abapGit.....6

 2.3 Installation using SAP Transport Requests.....6

3 Using rulerunner® framework.....7

 3.1 Concept overview.....7

 3.2 Customizing the rulerunner® framework.....8

 3.2.1 Basics.....8

 3.2.2 Assigning BRFplus functions to Event_Types/Resultgroups.....10

 3.2.3 Implementing Customer Rulesets.....10

 3.3 Execution of rules using the rulerunner® framework.....11

 3.3.1 Synchronous execution of business rules.....11

 3.3.1.1 Basics.....11

 3.3.1.2 Execution in ABAP.....11

 3.3.1.3 Execution in SAP BW Transformations.....12

 3.3.2 Asynchronous execution of rules.....12

 3.3.2.1 Basics.....12

 3.3.2.2 Adding events.....13

 3.3.2.3 Scheduling the execution of stored events.....13

 3.3.2.4 Adding rulerunner® events within BRFplus functions.....13

 3.3.2.5 Consuming rulerunner® events in ABAP.....13

 3.3.2.6 Consuming rulerunner® events in a SAP BW scenario.....15

 3.3.3 Accessing rulerunner® parameters within BRFplus functions.....17

 3.4 Persistence vs. on the fly – generic OData service.....17

 3.5 Rulerunner® metadata conversion and BRFplus Result Data Object.....20

 3.6 Rulerunner® context handover.....21

4 SAP BW specifics.....23

 4.1 General.....23

 4.2 When should we use rulerunner® and BRFplus.....23

 4.3 Complex BW delta scenarios and rulerunner® asynchronous execution mode.....23

 4.4 Detection of DTP simulation.....24

 4.5 Detection of delta DTP's.....24

5 Enhancements.....25

 5.1 BRFplus DB Lookup expression -enhancement.....25

 5.2 Jump to BRFplus from ABAP Editor -enhancement.....26

 5.3 BRFplus For All Entries In – enhancement.....28

6 Future improvements.....30

 6.1 Cross function context handover.....30

7 Appendix.....31

 7.1 Development Packages.....31

- 7.1.1 Package ZRULERUNNER.....31
- 7.1.2 Package ZRULERUNNER_BW.....31
- 7.1.3 Package ZRULERUNNER_MAIN.....31
- 7.1.4 Package ZRULERUNNER_ODATA.....31
- 7.2 BW Datasource for event data.....31
- 7.3 Class zcl_rulerunner.....32
 - 7.3.1 Public method ADD_EVENT.....32
 - 7.3.2 Public method DELETE_EVENTS.....33
 - 7.3.3 Public method MOVE_DATA_SOURCE_TO_TARGET.....33
 - 7.3.4 Public method PROCESS_EVENT_DIRECTLY.....34
 - 7.3.5 Public method PROCESS_MULTIPLE_EVENTIDS.....34
 - 7.3.6 Public method PROCESS_STORED_EVENTS.....35
 - 7.3.7 Public method SELECT_FOR_ALL_ENTRIES_IN.....36
 - 7.3.8 Public methodSHOW_RULERUNNER_CUSTOMIZING.....37
- 7.4 Class zcx_rulerunner.....37
- 7.5 Dictionary objects.....37
 - 7.5.1 Table ZRULERUN_DELTA.....37
 - 7.5.2 Table ZRULERUN_EVENTS.....38
 - 7.5.3 Table ZRULERUN_PLOG.....38
- 7.6 Message class.....39

1 What is it all about?

1.1 Target group of this document

The *rulerunner*® framework connects the world of SAP Datawarehousing (SAP BW) with the SAP Business Rule Management software SAP BRFplus.

Therefore you should have a profound knowledge in the following areas:

- SAP BW and/or SAP ERP
- SAP Business Rule Framework BRFplus
- ABAP programming language

Please note:

This document does not explain neither SAP BW nor ABAP nor BRFplus.

Please note:

Although this document has a strong focus on SAP BW, the *rulerunner*® framework can be used on an SAP ERP System (on ABAP) as well.

1.2 Basic idea from a business perspective

Business decisions are made on the basis of Key Performance Indicators (KPI).

These KPI's are calculated by an analytical software.

But if decision makers or business analysts ask about the calculation logic of a KPI, then usually an IT expert has to be consulted, because the **logic** is **hidden** somewhere in the analytical software.

We consider the calculation of **KPI's** as a mission critical task.

For decision makers and business analysts that means:

- KPI calculation logic should be defined in an **understandable** way (no ABAP)
- KPI calculation logic should be directly **accessible** (in the software, not in a separate doc)
- KPI calculation documentation must be **up-to-date** (no outdated PDF docs)
- Lifecycle of calculation logic must be **traceable**

Therefore:

- In order to fulfill these requirements the business logic must be defined in a **Business Rule Management System**.
- SAP provides the **Business Rule Framework BRFplus**® for that purpose. BRFplus® is available on most of the existing SAP Netweaver Systems. Having SAP ERP or BW licences, BRFplus can be used **free of charge**.
- The *rulerunner*® framework offers an easy way to bundle the power of SAP BW/ERP® and SAP BRFplus®.

1.3 What is the *rulerunner*[®] framework?

The *rulerunner*[®] framework:

- is a framework for easy and customizable execution of BRFplus functions in a SAP BW and ERP context.
- enhances the SAP Business Rule Plus Framework (BRFplus) by adding more flexibility.
- allows synchronous and asynchronous execution of BRFplus functions.
- is intended to be used in mass data scenarios, especially in SAP BW.
- supports enhanced delta-load scenarios in SAP BW.

Defining business logic in a Rule Management System means:

- higher transparency of business logic
- better understandability of complex algorithms for business analysts, they may even change it themselves
- ability to redesign business logic more easily
- auditable change management of business logic

Advantages using *rulerunner*[®] framework

The framework adds more flexibility when embedding business rules into business processes using the event customization concept, it is very easy to replace or add more business rules w/o changing the business process itself.

1.4 *Rulerunner*[®] is free

Rulerunner[®] is publicly available on <https://github.com/rulerunner/rulerunner4ABAP> .

Rulerunner[®] can be used free of charge.

Contributors to the project are very welcome.

2 Installation of the *rulerunner*® framework

2.1 Prerequisites

Rulerunner® is developed on a ABAP 7.5 stack.

Rulerunner® runs on BW/4 HANA (on premise)

Installation on a 7.3 system requires minor changes in the source code after import.

SAP Business Rule Framework BRFplus must be active on your system.

2.2 Installation using abapGit

The source code of *rulerunner*® is available at <https://github.com/rulerunner/rulerunner4ABAP>

The source code can be installed onto your system by means of the abapGit tools. Please refer to <https://docs.abapgit.org/> for details.

Please note:

Due to current limitations of the abapGit project, the content of the *rulerunner*® packages is **not complete** (e.g. BRFplus objects and BW datasources are missing). In order to work with *rulerunner*® framework, you need to import the BRFplus application manually via a XML Import. The required XML file BrfplusApplicationRulerunner201901181421.xml is available in the main folder of the master branch.

Please ask for the SAP Standard Transport Request files for the complete content of the *rulerunner*® packages (Installation using SAP Transport Requests).

2.3 Installation using SAP Transport Requests

Please send an email to info@dkrsolutions.com . We will send you the SAP Standard Transport Request files.

3 Using rulerunner® framework

3.1 Concept overview

The rulerunner® framework encapsulates the execution of SAP BRFplus business rules as shown in [Figure 1: rulerunner® Overview](#)

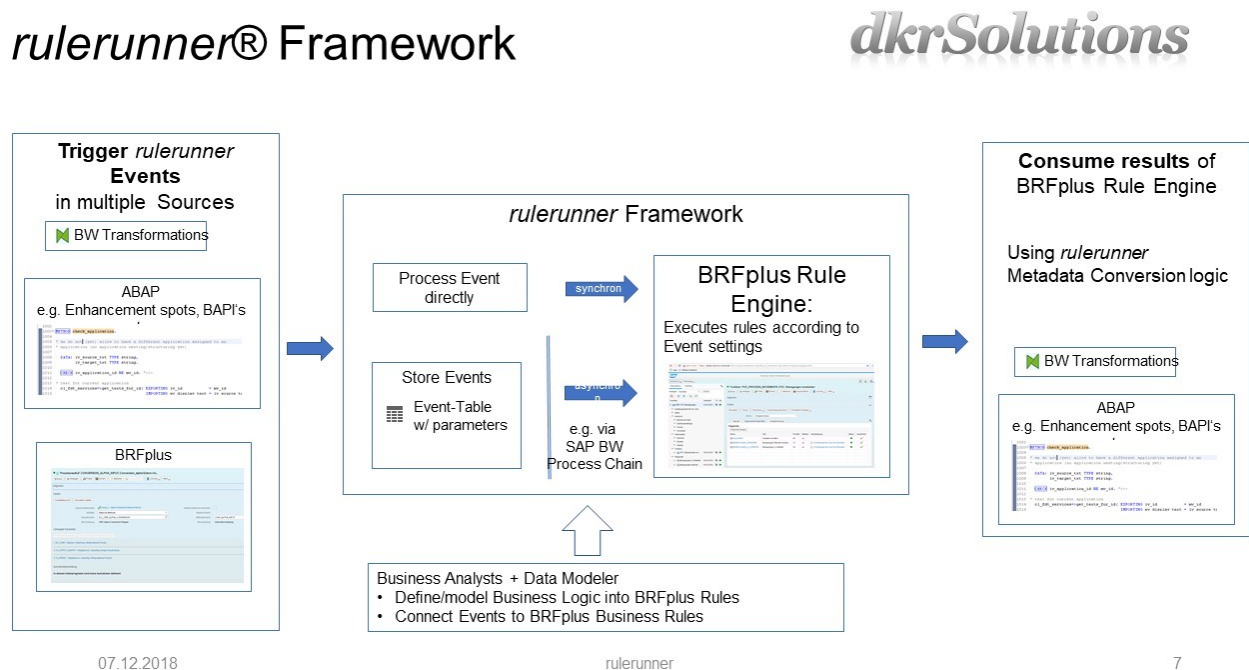


Figure 1: rulerunner® Overview

With the rulerunner® framework BRFplus business rules are not executed directly. The rulerunner® framework decouples the execution of BRFplus functions from the invoking/consuming ABAP code. It even allows to disconnect the invoking ABAP code from the consuming ABAP code by providing an asynchronous execution mode.

The execution of business rules is triggered by providing the parameter **Event_Type** and other optional parameters to the rulerunner® framework. The rulerunner® Event_Type is required to determine which BRFplus functions must be executed.

From a consumption perspective the results of the BRFplus business rules can be further specified by providing the parameter **Resultgroup**. The rulerunner® Resultgroup defines a subset of BRFplus functions that should be executed.

The rulerunner® framework uses a BRFplus decision table in order to assign BRFplus business rules (defined by application and function name) to the rulerunner® Event_Type and Resultgroup. This is shown in [Figure 2: Customizing BRFplus functions](#)(for details see chapter [Customizing the rulerunner® framework](#))

Table Contents

IV_EVENT_TYPE	RESULTGROUP	APPLICATION_NAME	FUNCTION_NAME
=HOSPITAL_CASE ☺	MOVEMENTS ☺	ORR_POC_BEWEGUNGEN ☺	POC_PROCESS_MOVEMENTS ☺
=HOSPITAL_CASE ☺	MOVEMENTS ☺	ORR_POC_BEWEGUNGEN ☺	POC_PROCESS_MOVEMENTS_DUMMY ☺
=HOSPITAL_CASE ☺	... ☺	ORR_POC_BEWEGUNGEN ☺	FCT_GIBT_ES_NICHT ☺
=SIMPLE_COMP_CODE ☺	... ☺	ORR_POC_BEWEGUNGEN ☺	POC_GET_COMP_CODE ☺
=ODATA_KEYVALUE_TEST ☺	... ☺	ORR_POC_BEWEGUNGEN ☺	ODATA_TEST ☺

Figure 2: Customizing BRFplus functions

Synchronous mode:

If rulerunner® framework is used in synchronous mode (i.e. triggering the execution and consuming the results happen at the same time) only one call is required (see chapter [Synchronous execution of business rules](#)). Parameters Event_type (mandatory) and Resultgroup (optional) are provided with this call.

Asynchronous mode:

In asynchronous mode an invoker triggers an event by calling the rulerunner® framework providing the parameter **Event_Type** and other optional parameters. This event is added to the rulerunner® event queue (for details see chapter [Asynchronous execution of rules](#)). The event queue is stored in a database table.

The stored events can be consumed later by multiple consumers (for details see chapters [Consuming rulerunner® events in ABAP](#) and [Consuming rulerunner® events in a SAP BW scenario](#)). Each consumer may specify what BRFplus functions should be executed by providing the parameter **Resultgroup**.

The rulerunner® framework automatically tries to convert the results of the BRFplus business rules into the target data format.

When executing rulerunner® events the caller can provide additional parameters as simple key-value pairs.

These parameters can be used within the BRFplus business rules (see chapter [Accessing rulerunner® parameters within BRFplus functions](#)).

3.2 Customizing the rulerunner® framework

3.2.1 Basics

The rulerunner® framework uses only BRFplus to customize the behaviour of the framework. The BRFplus application “RULERUNNER” contains a BRFplus function “**get_eventtype_metadata**” (see [Figure 3: rulerunner® BRFplus application](#)) which provides all necessary settings.

Please note: BRFplus application “RULERUNNER” is part of the [Package ZRULERUNNER_MAIN](#).

The BRFplus function “**get_eventtype_metadata**” contains 4 rulesets that can be used directly. Thanks to the BRFplus extension concept these rulesets can be replaced by customer rulesets.

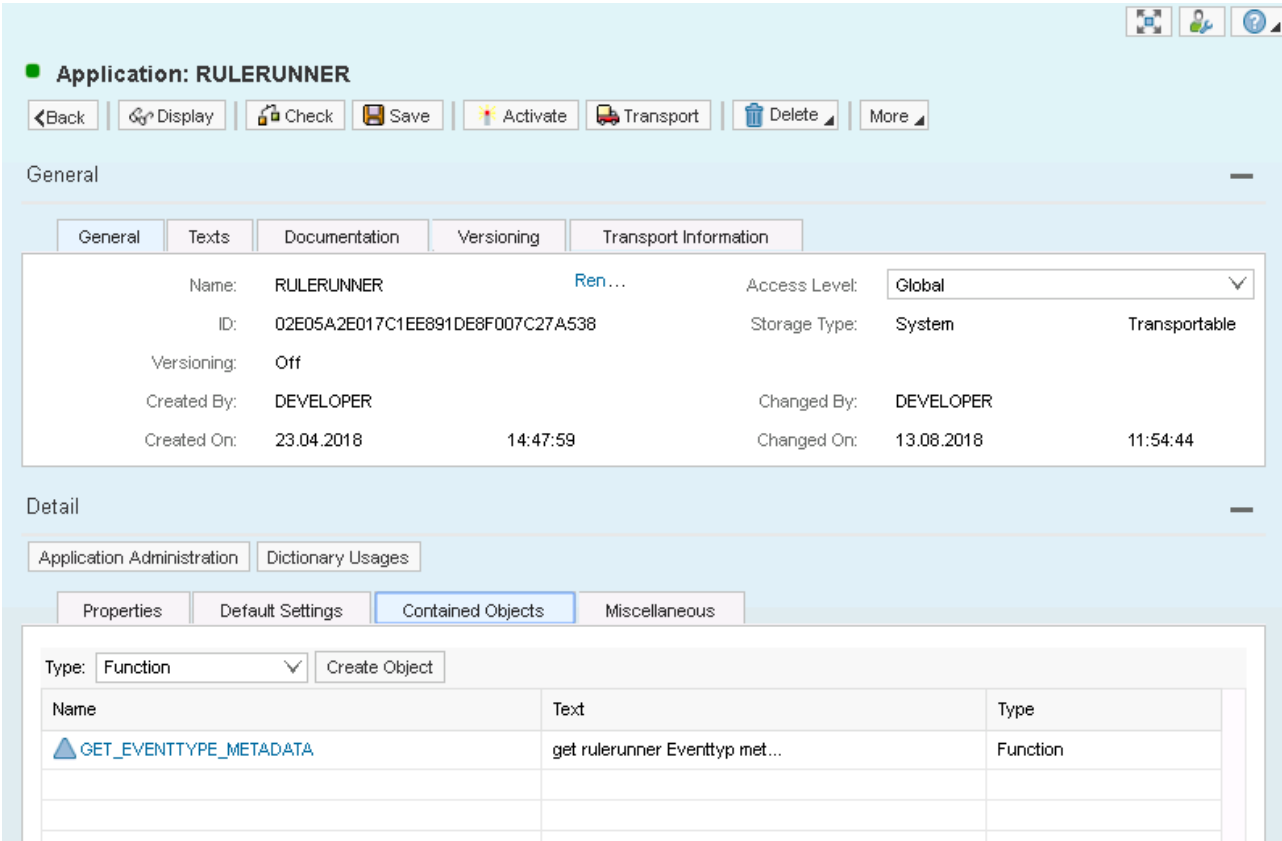


Figure 3: rulerunner® BRFplus application

In order to simplify the customization process, we created a BRFplus Catalog, see [Figure 4: rulerunner® BRFplus Catalog](#).

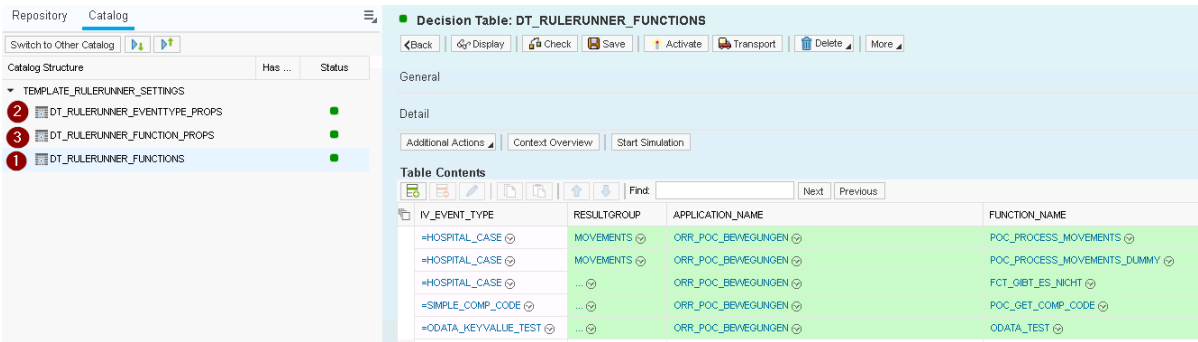


Figure 4: rulerunner® BRFplus Catalog

Catalog Element 1 “DT_RULERUNNER_FUNCTIONS“ is the place where rulerunner® parameters **Event_Type** and **Resultgroup** are assigned to BRFplus application and functions.

Catalog Elements 2 and 3 can be used to assign additional properties to the Event_Types and BRFplus functions. These properties are not used at this point in time.

3.2.2 Assigning BRFplus functions to Event_Type/Resultgroups

The rulerunner® parameter **Event_Type** classifies an event (from the perspective of the invoker) whereas the rulerunner® parameter **Resultgroup** defines a subset of BRFplus functions that must be executed from a Business Rule consumer perspective

The rulerunner® framework uses a BRFplus **decision table** in order to assign BRFplus business rules (defined by application and function name) to the rulerunner® Event_Type and Resultgroup
This is shown in Figure 5: rulerunner® BRFplus function assignment)

Table Contents

IV_EVENT_TYPE	RESULTGROUP	APPLICATION_NAME	FUNCTION_NAME
=HOSPITAL_CASE	MOVEMENTS	ORR_POC_BEWEGUNGEN	POC_PROCESS_MOVEMENTS
=HOSPITAL_CASE	MOVEMENTS	ORR_POC_BEWEGUNGEN	POC_PROCESS_MOVEMENTS_DUMMY
=HOSPITAL_CASE	...	ORR_POC_BEWEGUNGEN	FCT_GIBT_ES_NICHT
=SIMPLE_COMP_CODE	...	ORR_POC_BEWEGUNGEN	POC_GET_COMP_CODE
=ODATA_KEYVAL_IF_TEST	...	ORR_POC_BEWEGUNGEN	ODATA_TEST

Figure 5: rulerunner® BRFplus function assignment

Please note:
Whenever a rulerunner® event is triggered the parameter **Event_Type** is **mandatory**.
If there is no BRFplus function assigned to that Event_Type (Figure 5: rulerunner® BRFplus function assignment) then nothing will happen.
The parameter **Resultgroup** is **optional**. It can be used to differentiate between different consumers of the events.

3.2.3 Implementing Customer Rulesets

You might want to enhance our assignment template in BRFplus function “GET_EVENTTYPE_METADATA”, especially when hundreds or thousands of Event_Types must be handled. E.g, it is a good practice to split the assignments into different Decision Tables (e.g. one Decision Table per Division or Department).
To do so, you can define additional rulesets and register them to this function (see Figure 6: rulerunner® BRFplus function get_eventtype_metadata). These customer rulesets should have a Priority >10 and < 99. If the customer rulesets set the context parameter GV_RUN_TEMPLATES to FALSE, then the 3 TEMPLATE* rulesets (w/ priority = 99) will not be processed anymore.

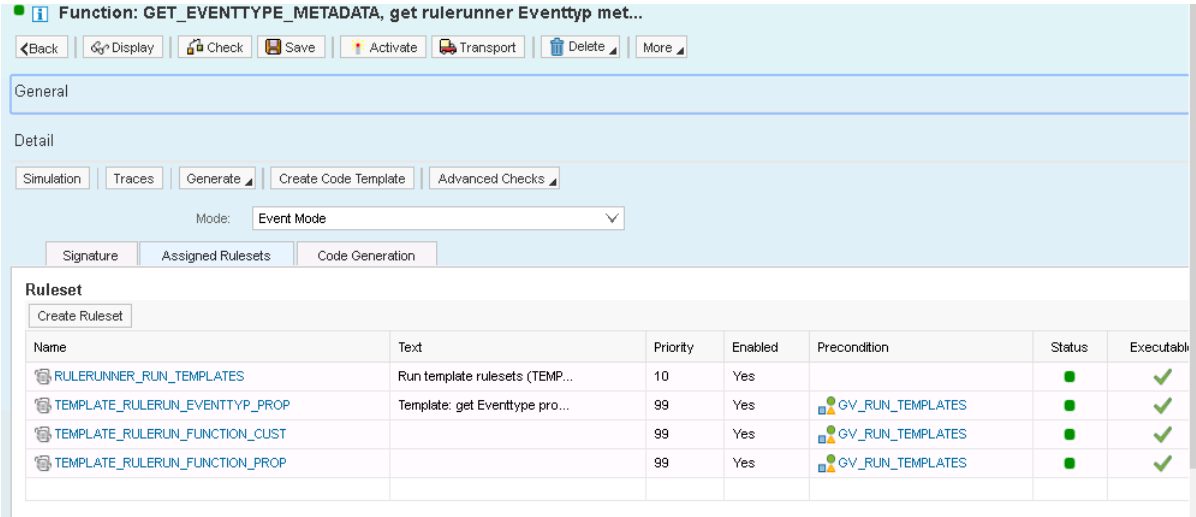


Figure 6: rulerunner® BRFplus function get_eventtype_metadata

3.3 Execution of rules using the rulerunner® framework

3.3.1 Synchronous execution of business rules

3.3.1.1 Basics

Synchronous execution means, that the invoker wants the business rules to be executed immediately. For this case rulerunner® framework provides the simple method “process_event_directly” whether used in an ABAP or in a SAP BW context.

3.3.1.2 Execution in ABAP

The rulerunner® framework provides a simple method “process_event_directly”.
Example:

```
call method zcl_rulerunner=>process_event_directly
exporting
    iv_event_type      = 'YOUR_EVENT_TYPE'
*   it_parameters      = 'optional list of parameter'
    iv_parameter_1_key = 'parameter1-Key'
    iv_parameter_1_value = 'Value1'
    iv_parameter_2_key  = 'parameter 2'
    iv_parameter_2_value = 'Value 2'
*   iv_parameter_3_key  =
*   iv_parameter_3_value =
*   iv_resultgroup      = 'optional resultgroup'
*   it_resultgroups     = 'optional list of resultgroups'
importing
    ev_returncode      = lv_returncode
    eo_result_data      = lt_result_package.
```

Please note, that the parameter **iv_event_type** is **mandatory**.
For details about the method, please refer to chapter [Public method PROCESS_EVENT_DIRECTLY](#).

If you provide the optional `iv_parameter*` or `it_parameters`, then these values can be accessed from within the BRFplus rule definition, please refer to chapter [Accessing rulerunner® parameters within BRFplus functions](#).

3.3.1.3 Execution in SAP BW Transformations

With rulerunner® framework it is very easy to use SAP BRFplus in order to calculate business data. The synchronous execution method “**process_event_directly**” is used in a BW Transformation as well. The coding from chapter [3.3.1.2 Execution in ABAP](#) must be inserted into Start-, End- or Expert Routine of a BW Transformation.

Please note: The Result Data Object of the BRFplus functions should be very similar to the target of your BW Transformation. The framework automatically tries to convert the Result Data Object into the Result Package of your BW Transformation (for details see chapter [3.5 Rulerunner® metadata conversion and BRFplus Result Data Object](#)).

Example: Coding in an End Routine of a BW Transformation

```

loop at RESULT_PACKAGE assigning <result_fields>.

    call method zcl_rulerunner=>process_event_directly
      exporting
        iv_event_type           = 'HOSPITAL_CASE'
        it_parameters           =
        iv_parameter_1_key      = 'OHC_INSTITU'
        iv_parameter_1_value    = <result_fields>-hc_institu
        iv_parameter_2_key      = 'OHC_PATCASE'
        iv_parameter_2_value    = <result_fields>-hc_patcase
        iv_parameter_3_key      =
        iv_parameter_3_value    =
        iv_resultgroup          =
        it_resultgroups         =
      importing
        ev_returncode           =
        eo_result_data           = lt_result_package_new.

endloop.

RESULT_PACKAGE = lt_result_package_new.

```

3.3.2 Asynchronous execution of rules

3.3.2.1 Basics

In asynchronous mode an invoker triggers an event by calling the rulerunner® framework providing the parameter **Event_Type** and other optional parameters. This event is added to the rulerunner® event queue (for details see chapter [Public method ADD_EVENT](#)). The event queue is stored in a database table.

The stored events can be consumed later by multiple consumers (for details see chapters [Consuming rulerunner® events in ABAP](#) and [Consuming rulerunner® events in a SAP BW](#)

[scenario](#)). Each consumer may specify what BRFplus functions should be executed by providing the parameter **Resultgroup**.

3.3.2.2 Adding events

The rulerunner® framework provides the method **Add_Event** to add events to the event queue. For details please refer to chapter [Public method ADD_EVENT](#).

Example:

```
call method zcl_rulerunner=>add_event
exporting
  iv_event_type      = 'Your Event Type'
  iv_suppress_exceptions = ''
*   it_parameters     = 'optional list of parameters'
  iv_parameter_1_key = 'parameter 1'
  iv_parameter_1_value = 'value 1'
  iv_parameter_2_key = 'parameter 2'
  iv_parameter_2_value = 'value 2'
*   iv_planned_execution_timestamp = '20200101000000'
  iv_bw_dtp_request = request
*   iv_bw_dtp_delta_mode = lv_updatemode
*   importing
*   ev_returncode      =
```

Please note:

Within a user session rulerunner® framework buffers all added events (via **Add_Event()**).

If an event with identical parameters (Event_Type, parameters and planned execution timestamp) has already been added within this user session, then the new event will be ignored.

3.3.2.3 Scheduling the execution of stored events

[Public method ADD_EVENT](#) has the optional import parameter

“iv_planned_execution_timestamp”.

If this parameter is supplied with a timestamp in the future, then the event will only be processed if the actual timestamp (i.e. when execution/consumption of the event takes place) is greater or equal to this parameter.

If import parameter “iv_planned_execution_timestamp” is not supplied, then the Planned Execution Timestamp is set to the actual timestamp when calling the ADD_EVENT method.

3.3.2.4 Adding rulerunner® events within BRFplus functions

The BRFplus provides an expression type Procedure Call. You can use `zcl_rulerunner=>add_event()` directly with this expression type.

3.3.2.5 Consuming rulerunner® events in ABAP

Stored rulerunner® events can be consumed by using method **process_stored_events()** (for details see chapter [7.3.6 Public method PROCESS_STORED_EVENTS](#)).

This method is intended to be used in a non SAP BW environment, e.g. in an ERP System.

Process_stored_events() provides a variety of calling options, that are described hereafter:

Packetised Mode:

The number of stored events is unknown, when calling this method. In order to avoid memory overflows or other performance issues you can call this function in packetised mode

`iv_run_packetised='X'`. That means that you have to call this function multiple times until parameter `ev_no_more_data` equals to `'X'`. You can specify the number of events that are processed with parameter `iv_package_size`. Please note that the number of result records in `eo_result_data` depends on the BRFplus functions that are assigned to the events.

Delta Mode:

It is a common requirement to process only events that have not been processed so far. In our case this is not easy to determine, because rulerunner® events can be consumed by multiple recipients. These recipients define their requirements (i.e. which BRFplus functions should be executed) by providing the parameter Resultgroup (see chapter [Assigning BRFplus functions to Event_Types/Resultgroups](#)). In order to determine whether a combination of event and Resultgroup has been processed yet, we would have to analyze the processing log (see [Table ZRULERUN_PLOG](#)).

That would be way to slow and not accurate as updating the processing log can be suppressed. Instead we implemented a delta mechanism that stores the timestamp when a combination of Eventtype/Resultgroup was processed lastly. These timestamps are stored in [Table ZRULERUN_DELTA](#). Each time method **process_stored_events()** is called in Delta Mode (`iv_delta_mode = ,D'`) the last execution timestamp from [Table ZRULERUN_DELTA](#) is set as the lower limit of the planned execution time. With this timestamp all events in [Table ZRULERUN_EVENTS](#) that have a planned execution timestamp greater then that will be processed.

Please set parameter `iv_update_delta_timestamp` to `'X'` to update the timestamp in [Table ZRULERUN_DELTA](#) to the execution time. Note that if parameter `iv_test_mode` is set to `'X'` then delta timestamps will not be updated.

Processing Log:

[Table ZRULERUN_PLOG](#) stores the execution timestamp for each Event_Type, Resultgroup and BRFplus function-ID (see [Figure 1: rulerunner® Overview](#)).

Data Browser: Table ZRULERUN_PLOG Select Entries						110
<div>Check Table...</div>						
	CLIENT	EVENTID	RESULTGROUP	FUNCTIONID	TST_PROCESSED	SKIPPED
	001	7264	MOVEMENTS	02E05A2E017C1EE887A54D2CA7C603FB	20.181.204.101.629	
	001	7264	MOVEMENTS	02E05A2E017C1EE897CD8210043E1644	20.181.204.101.629	
	001	7266	MOVEMENTS	02E05A2E017C1EE887A54D2CA7C603FB	20.181.204.101.629	
	001	7266	MOVEMENTS	02E05A2E017C1EE897CD8210043E1644	20.181.204.101.629	
	001	7268	MOVEMENTS	02E05A2E017C1EE887A54D2CA7C603FB	20.181.204.101.629	
	001	7268	MOVEMENTS	02E05A2E017C1EE897CD8210043E1644	20.181.204.101.629	
	001	7270	MOVEMENTS	02E05A2E017C1EE887A54D2CA7C603FB	20.181.204.101.629	
	001	7270	MOVEMENTS	02E05A2E017C1EE897CD8210043E1644	20.181.204.101.629	

Figure 7: rulerunner® processing log example

Please note: The processing log will only be updated if parameter `iv_update_processing_log` is set to 'X'.

When *rulerunner*® events are processed the system checks if events have identical parameters. If so, only the first event is processed and all other events are skipped. For skipped events the field “skipped” equals to 'X' in the corresponding record of [Table ZRULERUN_PLOG](#).

If parameter `iv_repeat_processing` is set to 'X' then *rulerunner*® framework will process all BRFplus functions that are assigned to the Event_Type/Resultgroup (see [Assigning BRFplus functions to Event_Types/Resultgroups](#)). Otherwise it will process only those BRFplus functions that have not been processed yet.

This is quite useful, if an event has been processed yet and afterwards new BRFplus functions have been added to the Event_Type/Resultgroup Customizing. In this case you can execute only those new BRFplus functions by executing **PROCESS_STORED_EVENTS()** with `iv_delta_mode = ''` and `iv_repeat_processing = ''`.

If you don't need a processing pog, please execute **PROCESS_STORED_EVENTS()** with `iv_update_processing_log = ''` and `iv_repeat_processing = 'X'`.

3.3.2.6 Consuming rulerunner® events in a SAP BW scenario

When used within a SAP BW datamodel, BRFplus functions are used to calculate data that are stored in or consumed by an BW Infoprovider.

All stored *rulerunner*® events can be loaded into the SAP BW PSA (Persistent Staging Area). The PSA contains a list of event-ID's (together with other data from table [7.5.2 Table ZRULERUN_EVENTS](#)).

From there the events can be distributed into the target data providers via BW Transformations.

Package [7.1.2 Package ZRULERUNNER_BW](#) contains the SAP BW Datasource (ODP and Classic) ZRULERUNNER_EVENTS (see [Figure 8: BW Datasource for stored rulerunner® events](#)).

The datasource provides events in delta update mode. Delta relevant events are determined via the Planned Execution Timestamp (see chapter [3.3.2.3 Scheduling the execution of stored events](#))

RSDS ZRULERUNNER_EVENTS A4HODPEX -> ODSO ISMOVKFG	DQQIIAQ3QCX52LK0PE58FKN98WIVKN1G	Change
▼ rulerunner: Event Data	ZRULERUNNER_EVENTS	Change
• ZRULERUNNER_EVENTS ODP Delta	ZPAK_7KOIWDZ65OXKECAYXEXIAN3WA	Execute

Figure 8: BW Datasource for stored rulerunner® events

Once the events are loaded into the PSA, they may be consumed by multiple BW Transformations. An example dataflow is shown in [Figure 9: BW: Transformation from ZRULERUNNER_EVENTS to an Infoprovider](#).

▼ rulerunner: Event Data	ZRULERUNNER_EVENTS
• ZRULERUNNER_EVENTS ODP Delta	ZPAK_7KOIWDZ65OXKECAYXEXIAN3WA
▼ Data Flow Upwards	_DATAFLOW_UPWARDS
▶ ZRULERUNNER_EVENTS / A4HODPEX -> ISMOVKFG from PSA	DTP_0640Z5QFFV102V10HMDAR5LZ8
▼ RSDS ZRULERUNNER_EVENTS A4HODPEX -> ODSO ISMOVKFG	DQQIIAQ3QCX52LK0PE58FKN98WIVKN1G
• Movement Keyfigures	ISMOVKFG

Figure 9: BW: Transformation from ZRULERUNNER_EVENTS to an Infoprovider

Datasource ZRULERUNNER_EVENTS provides the stored event-ID's to the BW Transformations. These event-ID's are consumed in the Start-, End- or Expert Routine of the BW Transformations. In order to process all events of a list of event-ID's rulerunner® provides the method **PROCESS_MULTIPLE_EVENTIDS()** ([Public method PROCESS_MULTIPLE_EVENTIDS](#)).

Example:

Transformation “ZRULERUNNER_EVENTS / A4HODPEX -> ISMOVKFG from PSA” uses an Expert Routine. The Expert Routine is very straight forward and looks like this:

```
METHOD expert_routine.
*=== Segments ===

FIELD-SYMBOLS:
  <SOURCE_FIELDS>      TYPE _ty_s_SC_1.

DATA:
  RESULT_FIELDS        TYPE _ty_s_TG_1.

*$$$ begin of routine - insert your code only below this line      *-*
... "insert your code here
call method zcl_rulerunner=>process_multiple_eventids
exporting
  it_table_with_eventid = SOURCE_PACKAGE
*  iv_update_processing_log = 'X'
  iv_repeat_processing = 'X'
  iv_package_size = 1000
  iv_resultgroup = 'MOVEMENTS'
*  it_resultgroups =
importing
  eo_result_data = RESULT_PACKAGE.
```



```
*$*$ end of routine - insert your code only before this line *-*  
ENDMETHOD.
```

3.3.3 Accessing rulerunner® parameters within BRFplus functions

All parameters that have been provided while calling a rulerunner® event can be accessed within the BRFplus function.
All you need to do is to include the BRFplus table GT_RULERUNNER_CONTEXT into the context of your BRFplus function, see [Figure 8: BW Datasource for stored rulerunner® events](#).

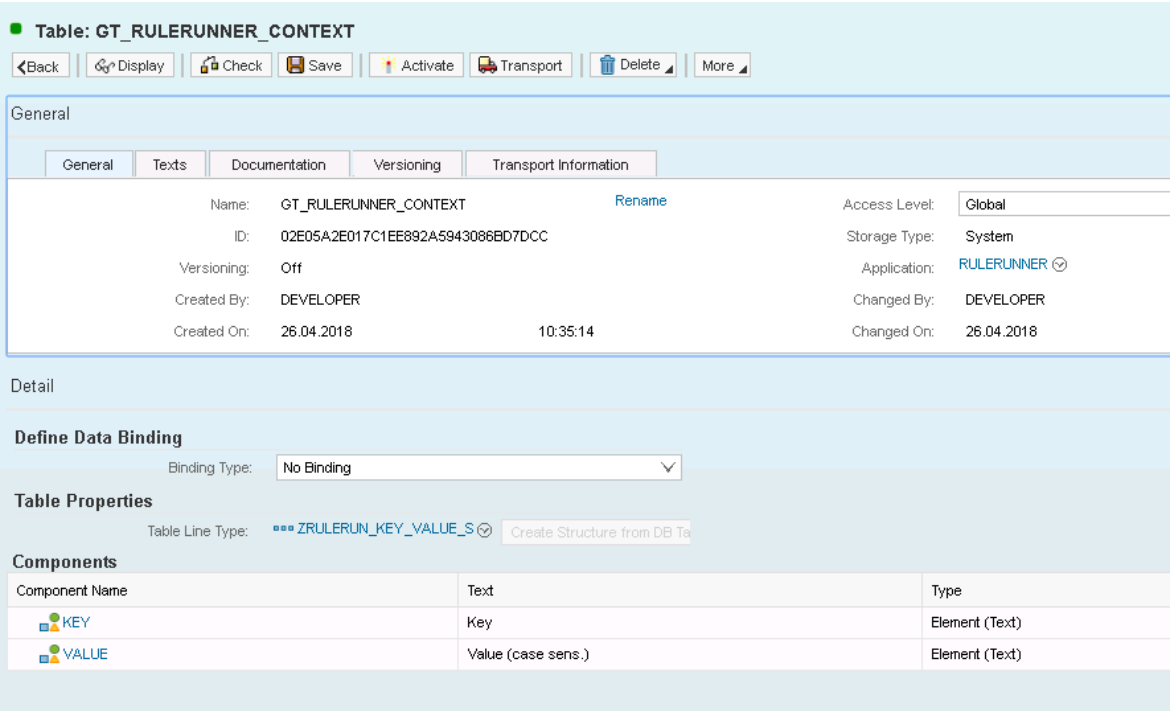


Figure 10: rulerunner® BRFplus context table
Table GT_RULERUNNER_CONTEXT is part of the RULERUNNER application (see [Figure 3: rulerunner® BRFplus application](#)).
Rulerunner® framework will automatically transfer all parameters into the context table.

3.4 Persistence vs. on the fly – generic OData service

Please note:
The generic rulerunner® OData service is still work in progress. At this point in time your BRFplus function must return a key-value result table in order to work properly.
Databases like HANA become more and more powerful.
Therefore complex calculations can be executed on the fly instead of storing the results in a BW Infoprovider or a database table.
With rulerunner® framework and BRFplus you can define business logic that can be consumed in either way. The results of the same logic can be stored in a DB or used on the fly.
In order to consume BRFplus functions on the fly an OData service must be created.
Rulerunner® framework provides a generic OData service (see also chapter [7.1.4Package ZRULERUNNER_ODATA](#)). To work properly an appropriate OData service data model has been defined. The data model should work independently of the Result Data Objects of the BRFplus

functions.

The data model uses a key value type Entity Set, see [Figure 11: rulerunner OData Service: Data Model](#).

Please note:

The OData service is still work in progress. The current data model is a flat structure without associations. All import parameters for the execution of *rulerunner*® methods have to be provided using the fields with prefix "IV".

The result data are provided in the fields "KEY" and "VALUE". The BRFplus functions must provide a result structure or table containing the fields "KEY" and "VALUE".

Only up to 3 key/value parameters and 1 resultgroup can be passed.

This Data Model should be re-engineered in the future.

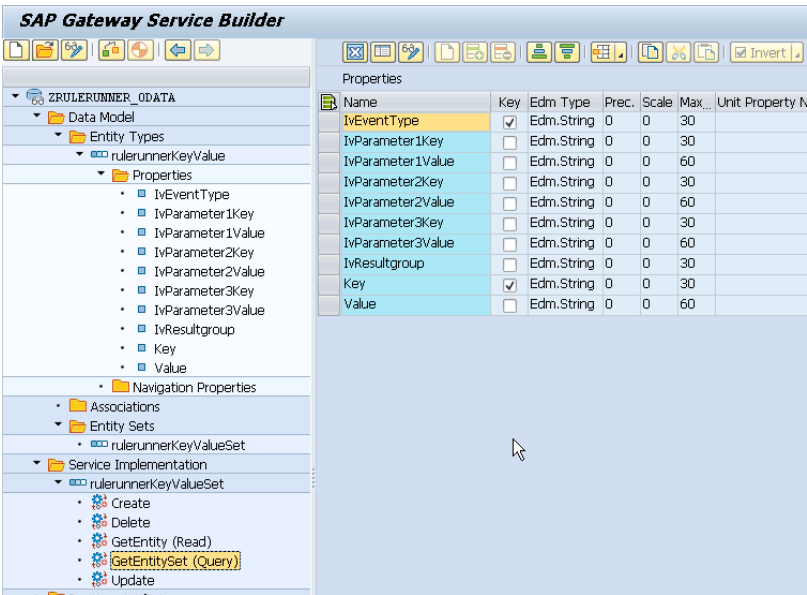


Figure 11: rulerunner OData Service: Data Model

The implementation of the GetEntitySet service is very straight forward. It makes use of the rulerunner® process_event_directly method (7.3.4 Public method [PROCESS_EVENT_DIRECTLY](#)), see [Figure 12: rulerunner OData Service: GetEntitySet implementation.](#)

```
loop at it_filter_select_options assigning <ls_filter>.
* all filters should be provided with option=EQ and sign = I
  read table <ls_filter>-select_options assigning <ls_filter_sel> index 1.
  case <ls_filter>-property.
    when 'IvEventType'.
      lv_event_type = <ls_filter_sel>-low.
    when 'IvParameter1Key'.
      lv_parameter_1_key = <ls_filter_sel>-low.
    when 'IvParameter1Value'.
      lv_parameter_1_value = <ls_filter_sel>-low.
    when ''.
      lv_parameter_2_key = <ls_filter_sel>-low.
    when 'IvParameter2Value'.
      lv_parameter_2_value = <ls_filter_sel>-low.
    when 'IvParameter3Key'.
      lv_parameter_3_key = <ls_filter_sel>-low.
    when 'IvParameter3Value'.
      lv_parameter_3_value = <ls_filter_sel>-low.
    when 'IvResultgroup'.
      lv_resultgroup = <ls_filter_sel>-low.
  endcase. " <ls_filter>-property.
endloop. "at it_filter_select_options assigning <ls_filter>.

[check] lv_event_type is not initial.

* -----
* Step: process rulerunner event directly
zcl_rulerunner=>process_event_directly(
  exporting
    lv_event_type      = lv_event_type
    it_parameters      =
*   iv_parameter_1_key = lv_parameter_1_key
*   iv_parameter_1_value = lv_parameter_1_value
*   iv_parameter_2_key = lv_parameter_2_key
*   iv_parameter_2_value = lv_parameter_2_value
*   iv_parameter_3_key = lv_parameter_3_key
*   iv_parameter_3_value = lv_parameter_3_value
*   iv_resultgroup      = lv_resultgroup
*   it_resultgroups     =
  importing
    ev_returncode      =
*   eo_result_data      = et_entityset
  ).
```

Figure 12: rulerunner OData Service: GetEntitySet implementation

3.5 Rulerunner® metadata conversion and BRFplus Result Data Object

BRFplus functions provide a Result Data Object (Actions are not supported by rulerunner®). After being executed, the Result Data Object contains the data (Figure 13: BRFplus Result Data

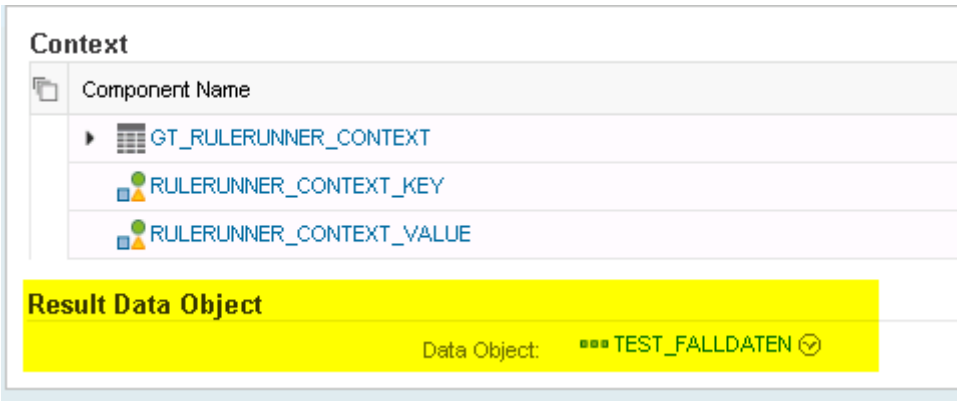


Figure 13: BRFplus Result Data Object

Object).

With rulerunner® the results of all BRFplus functions are transferred automatically into the target data object.

The target data object is supplied with the parameter **eo_result_data** (type any) when the consuming method is called ([Public method PROCESS_EVENT_DIRECTLY](#), [Public method PROCESS_MULTIPLE_EVENTIDS](#), [Public method PROCESS_STORED_EVENTS](#)).

Rulerunner® will analyse the metadata of the target data object and **all** Result Data Objects (multiple BRFplus functions may be executed at once, see [3.2 Customizing the rulerunner® framework](#)).

Then it tries to create a field mapping between the source data (BRFplus Result Data Objects) and the target data object.

Currently rulerunner® is able to convert automatically:

- structure to structure
- structure to table
- table to table
- table to structure

Currently rulerunner® is able to convert automatically the nested BRFplus date/time/timestamp format into a non-nested format (and vice versa).

The fieldmapping is create by a simple “source-fieldname = target-fieldname” approach.

This logic could be extended in future releases.

Please note:

The **result data** of each BRFplus Result Data Object **are always added** to the target data object.

Currently there is no way of overwriting or deleting data in the target data object.

3.6 Rulerunner® context handover

The rulerunner® framework supports the mass execution of rules.

In SAP BW it is a very common scenario to execute business logic multiple times, e.g. in a loop over the resultpackage in an Endroutine of a BW Transformation.

Each BRFplus function usually has some context data objects. These context objects contain data according to the requirements of the business logic.

The context data objects may contain data, that are also relevant for the next calls of the BRFplus function. For example the context may contain masterdata that have to be read from database.

In order to avoid redundant data operations (e.g. database reads) rulerunner® framework automatically has a feature called **context handover**.

If a consuming method ([Public method PROCESS_EVENT_DIRECTLY](#), [Public method PROCESS_MULTIPLE_EVENTIDS](#), [Public method PROCESS_STORED_EVENTS](#)) is called **multiple** times, then for each BRFplus function **all context data objects are handed over**.

Please note:

When designing your BRFplus functions you must be aware, that all context data objects may contain data due to the context handover. This data come from the last call of the BRFplus function.

Please note:

Context handover is currently only supported for the same BRFplus function. There is no context handover between different BRFplus functions.

4 SAP BW specifics

4.1 General

Chapter [SAP BW specifics](#) discusses some special aspects when using *rulerunner*® framework and BRFplus in combination with SAP BW.

4.2 When should we use *rulerunner*® and BRFplus

As described in chapter [1.2 Basic idea from a business perspective](#) the intention is to make the business logic much more transparent.

It is not required to implement every single logic with *rulerunner*® framework and BRFplus.

We suggest to use *rulerunner*® when:

- Business data are not transferred 1:1 but require some kind of calculations in SAP BW
- Masterdata need to be harmonized between heterogeneous sources, especially with legacy systems.
- Organizational structures are modified when processing data (e.g. cost centers, company codes, divisions, etc.)

When it comes to data cleansing while staging data, it might not be necessary to use *rulerunner*®.

4.3 Complex BW delta scenarios and *rulerunner*® asynchronous execution mode

With regard to complex calculations in SAP BW a common pattern can be found.

Usually the processing logic (especially in the Business Data Layer) is build in a way that only new or changed data must be processed, the so called Delta Update mode.

Whenever a calculation requires data from different BW sources (transactional or masterdata) it becomes very complicated to determine, which delta records must be processed. It is even more complicated to implement an appropriate BW model that respects all deltas in the different BW sources.

Sometimes this problem is ignored or solved by a full load from time to time.

With *rulerunner*® asynchronous execution mode we can address this problem in a very different way.

If a calculation logic is based on data from different BW sources, then each change of a record in the BW sources is considered as an *rulerunner*® event. These events must be registered in the *rulerunner*® event queue via the `ADD_EVENT` method ([3.3.2.2 Adding events](#)). The registration has to be implemented in all relevant Transformations where the deltas occur.

At the end, all delta records are stored in the *rulerunner*® event queue (only keys as parameters).

The calculation of the business logic is triggered by the events stored in *rulerunner*® then.

This is described in chapter [3.3.2.6 Consuming rulerunner® events in a SAP BW scenario](#) in detail. Therefore a transformation from the *rulerunner*® datasource to the target infoprovider has to be created. In order to consume multiple *rulerunner*® Event Types you can define an appropriate Resultgroup (see [3.2 Customizing the rulerunner® framework](#)) that comprises different Event Types.

Rulerunner® asynchronous execution mode facilitates a streamlined way to process complex delta scenarios. But how to deal with Initialisations or Full Loads?

We do not recommend to use *rulerunner*® asynchronous execution mode for Full loads or Delta Initialisations. It is better to create a Transformation from one of the BW sources into the target Infoprovider. The transformation uses the same *rulerunner*® Eventtype and Resultgroups like the transformation from *rulerunner*® datasource to the target Infoprovider.

4.4 Detection of DTP simulation

The `ADD_EVENT` ([3.3.2.2 Adding events](#)) method can be used in BW Transformations in order to store *rulerunner*® events. Data Transfer Processes (DTP) are used to start the process of loading data into a target.

If a DTP is started in Simulation mode than the data are not transferred to the target.

In this case it might be necessary to avoid storing *rulerunner*® events.

This can be done by adding import parameter `iv_bw_dtp_request = request` to the signature of method `ADD_EVENT`. The variable `request` exists only within a BW transformation and contains the number of an DTP instance/request. In case of a Simulation DTP it contains the specific value 'DTPR_SIMULATION' that enables *rulerunner*® to automatically detect a simulation.

4.5 Detection of delta DTP's

The `ADD_EVENT` ([3.3.2.2 Adding events](#)) method can be used in BW Transformations in order to store *rulerunner*® events. Data Transfer Processes (DTP) are used to start the process of loading data into a target.

It can be relevant to add events only in case that the DTP is executed in delta mode. Therefore the `ADD_EVENT` method has the import parameter `iv_bw_dtp_delta_mode = lv_updatemode`. The variable `lv_updatemode` can be determined with the following lines of code:

```
data: lv_updatemode type RSBKUPDMODE.  
lv_updatemode = p_r_request->get_updmode( ).
```

This code works only within a BW Transformation (i.e. Start-,End-,Export or Field-routines).

5 Enhancements

5.1 BRFplus DB Lookup expression -enhancement

For SAP BW objects like (e.g.) DSO or Infoobjects the database tables are generated by SAP BW. These generated tables may contain references to other tables. This happens quite often in case that currencies or units are involved.

When accessing these tables via the BRFplus DB Lookup expression these reference fields lead to errors like this (*Figure 14: BRFplus DB Lookup: Error due to reference fields*):

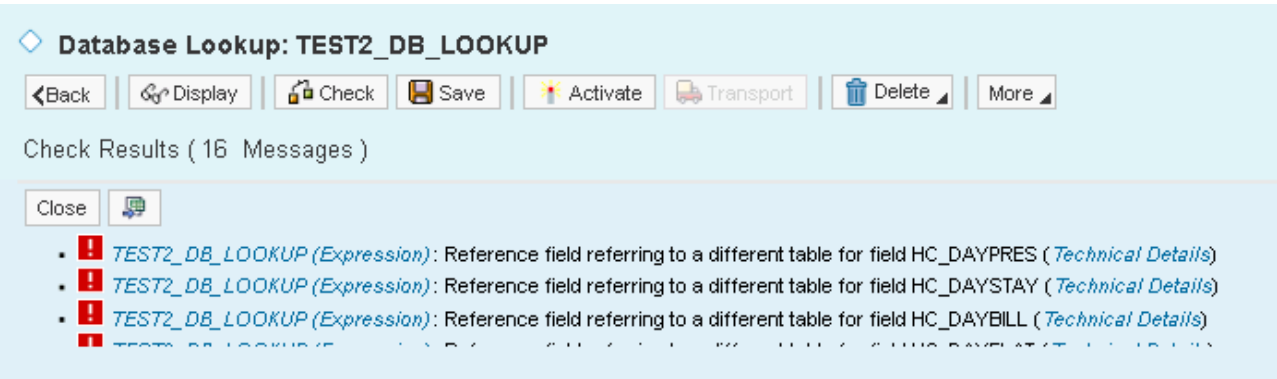


Figure 14: BRFplus DB Lookup: Error due to reference fields

In order to suppress these error messages implement the following static implicit enhancement

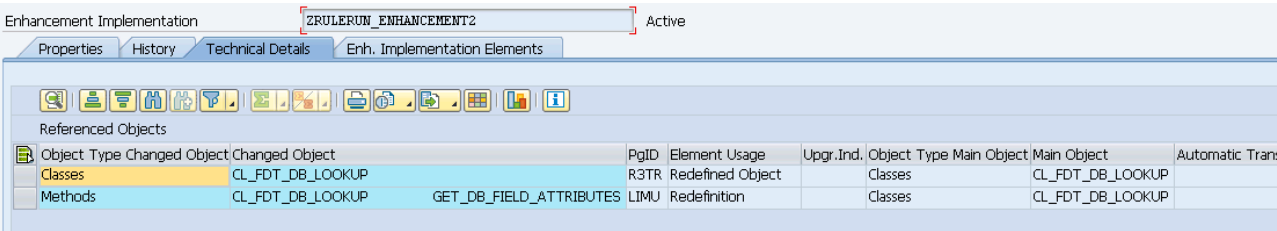


Figure 15: BRFplus DB Lookup: RefField enhancement Metadata

```

Method GET_DB_FIELD_ATTRIBUTES Active
172 eo_tmp_element->if_fdt_data_object-set_ddic_binding( iv_ddic_typename = <ls_db_field_buffer>-ddic_typename ).
173 ELSE.
174     "DDEV
175     * Default Element with no DDIC knowledge (ONLY UPPER CASE, Field Length, etc.).
176     eo_tmp_element->set_element_type( <ls_db_field_buffer>-element_type ).
177     IF <ls_db_field_buffer>-element_type EQ if_fdt_constants=>gc_element_type_timepoint.
178         eo_tmp_element->set_element_type_attributes(
179             iv_timepoint_type = <ls_db_field_buffer>-timepoint_type ).
180     ENDIF.
181 ENDIF.
182
183
184 *$*Start: (1)-----$*SE:(1) Class CL_FDT_DB_LOOKUP, A
185 ENHANCEMENT 1 ZRULERUN_ENHANCEMENT2. "active version
186 * rulerunner: Ignore reference tables and fields
187 CLEAR ev_reffield.
188 CLEAR ev_reftable.
189 ENDENHANCEMENT.
190 *$*End: (1)-----$*
191 ENDMETHOD.
192

```

Figure 16: BRFplus DB Lookup: RefField enhancement coding

Please note:

The enhancement will not provide any reference table and field.

5.2 Jump to BRFplus from ABAP Editor -enhancement

Please note:

This enhancement works only with a classic SAP GUI Code Editor like SE80.

In an ABAP Editor there is a build in navigation path to the object-definition when double-clicking it.

When double-clicking a rulerunner® method call, then the standard navigation opens up the source code of the rulerunner® method.

For an analyst this behaviour is not appropriate, because the business logic is not defined inside the source code of the rulerunner® method but in the BRFplus function.

Therefore you can implement an enhancement, that opens up the BRFplus Workbench whenever you double-click on the class name "ZCL_RULERUNNER" in an ABAP editor, e.g. in a BW Transformation.

The enhancement will open up BRFplus Workbench with the rulerunner® BRFplus function

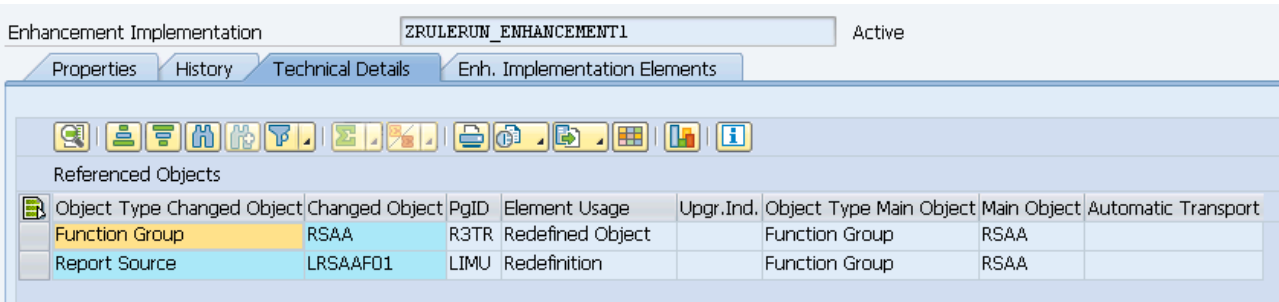
get_eventtype_metadata (see chapter [3.2.1 Basics](#)).

To do so, please add an implicit enhancement to ABAP Include LRSAAF01 at the beginning of FORM user_command_0100 (see [Error: Reference source not found](#))

Add the following lines of code:

```
IF fcode = 'WB_SELECT_OBJECT'.
.
*   get target object
CALL METHOD editor->get_navigation_object
EXPORTING
*       P_CONTEXT      =
        p_operation    = swbm_c_op_display
IMPORTING
        p_wb_request   = wb_request.
    "if target object = rulerunner then call BRFplus Workbench in a
Webbrowser
    IF wb_request->object_name = 'ZCL_RULERUNNER'.
        zcl_rulerunner=>show_rulerunner_customizing( ).
    ENDIF.

ENDIF.
```



The screenshot shows the 'Enhancement Implementation' window for 'ZRULERUN_ENHANCEMENT1'. The 'Technical Details' tab is active, displaying a table of 'Referenced Objects'.

Object Type	Changed Object	PgID	Element Usage	Upgr.Ind.	Object Type Main Object	Main Object	Automatic Transport
Function Group	RSAA	R3TR	Redefined Object		Function Group	RSAA	
Report Source	LRSAAF01	LIMU	Redefinition		Function Group	RSAA	

Figure 17: ABAP Editor enhancement: Metadata

```

ABAP Editor: Display Include LRSAAF01
Include LRSAAF01 Active

53   ENDIF.
54   ENDFORM.
55   " set fcode
56   *-----*
57   * Form user_command_0100
58   *-----*
59   *
60   *
61   FORM user_command_0100.
62   *$$$-Start: (1)-
63   ENHANCEMENT 2 ZRULERUN_ENHANCEMENT1. "active version
64   *
65   IF fcode = 'WB_SELECT_OBJECT'.
66   *
67   * get target object
68   CALL METHOD editor->get_navigation_object
69   EXPORTING
70   *
71   p_context =
72   p_operation = swbm_c_op_display
73   IMPORTING
74   p_wb_request = wb_request.
75   "if target object = rulerunner then call BRFplus Workbench in a Webbrowser
76   IF wb_request->object_name = 'ZCL_RULERUNNER'.
77     zcl_rulerunner=>show_rulerunner_customizing( ).
78   ENDIF.
79
80   ENDIF.
81   ENDENHANCEMENT.
82   *$$$-End: (1)-
83   DATA: l_program_state TYPE REF TO cl_wb_program_state,
84         l_wb_request_set TYPE swbm_wb_request_set,
85         l_error TYPE werror,

```

Please note:

The method `zcl_rulerunner=>show_rulerunner_customizing()` can open up any BRFplus object by adding the import parameter `iv_brf_component_id`. You need to pass the ID of a BRFplus object. By default the method opens the rulerunner® BRFplus function `get_eventtype_metadata`.

5.3 BRFplus For All Entries In – enhancement

In BRFplus the Database Lookup expression lacks a “Select for all entries in “ feature.

With the current release rulerunner® provides a static method *Public method* `SELECT_FOR_ALL_ENTRIES_IN` that can be used in BRFplus directly.

Procedure Call: For all entries in - Test

Back

Display

Check

Save

Activate

Transport

Delete

More

General

Detail

Context Overview

Start Simulation

Result Data Object:ET_ISMOVKFG

Call Type:Static Method

Class Name:ZCL_RULERUNNER

Description:RuleRunner

Use Interface Method:

Interface Name:

Method Name:SELECT_FOR_ALL_E

Description:No Description

Mapped Parameters

Add Parameter

Show Unsupported Parameters

1. ET_RESULT_DATA - Optional , Exporting (No Description)

Show Details

2. IT_FOR_ALL_ENTRIES_TABLE - Mandatory , Importing (No Description)

Show Details

3. IV_FOR_ALL_ENTRIES_TABLENAME - Mandatory , Importing (No Description)

Hide Details

Component Name	Description	Assigned Value	Move Type
IV_FOR_ALL_ENTRIES_TAB...	No Description	/BIO/SHC_PATCASE	Move Value

4. IV_SELECT_FROM_TABLE_NAME - Mandatory , Importing (No Description)

Hide Details

Component Name	Description	Assigned Value	Move Type
IV_SELECT_FROM_TABLE_...	No Description	/BIC/AISMOVKFG00	Move Value

5. IV_WHERE_FIELD_1_DB - Mandatory , Importing (No Description)

Show Details

6. IV_WHERE_FIELD_1_FOR_ALL - Mandatory , Importing (No Description)

Show Details

7. IV_WHERE_FIELD_2_DB - Optional , Importing (No Description)

Show Details

8. IV_WHERE_FIELD_2_FOR_ALL - Optional , Importing (No Description)

Show Details

Figure 18: BRFplus Select For All Entries In - Enhancement

Please provide mapped parameters 1 to 6 according to [Figure 18: BRFplus Select For All Entries In - Enhancement](#).

These parameters are mandatory.

You may provide up to 4 additional pairs of fieldnames (iv_where_field_x_db + iv_where_field_x_for_all).

Please note:

You **cannot** assign **fixed values** to a database field as a **where condition**. Only data in the internal table can be used as where conditions.

6 Future improvements

6.1 Cross function context handover

Context handover functionality can be implemented in a way that BRFplus context objects that are used in multiple BRFplus functions are handed over from function to function.

7 Appendix

7.1 Development Packages

7.1.1 Package ZRULERUNNER

This package is the Superpackage for the rulerunner® framework.

7.1.2 Package ZRULERUNNER_BW

Package contains all objects that are specific to SAP BW functionality. At present the package contains a standard and a ODP datasource for stored rulerunner® events.


7.1.3 Package ZRULERUNNER_MAIN


This package contains all the basic data- and programming objects.
The ABAP codeline resides within the static class ZCL_RULERUNNER.



7.1.4 Package ZRULERUNNER_ODATA


This package contains a prototype of a generic OData service.
In the SAP Gateway Service Builder (SEGW) the project ZRULERUNNER_ODATA has been created. The package contains the definition of the service and all generated development artifacts.

7.2 BW Datasource for event data

DataSource  ZRULERUNNER_EVENTS rulerun:Event data

Source System  A4HODPEX A4H ODP (SAP Extraction)

Version  Active  Compare with...

Active Version Executable  Edited Version









General Info.

Extraction

Proposal

Fields

Preview

Field Attributes															
Pos.	Field	Descript.	D...	T..	InfoObjec...	Data type	Length	Deci...	Exter...	C..	K..	Con...	Format	SS ...	Cur
1	EVENTID	Event ID		<input checked="" type="checkbox"/>		NUMC	18	0	18	<input type="checkbox"/>	<input checked="" type="checkbox"/>		Inter...		
2	EVENTTYPE	Event Type		<input checked="" type="checkbox"/>		CHAR	30	0	30	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
3	TST_CREATED	Created-Tim...		<input checked="" type="checkbox"/>		DEC	15	0	16	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
4	TST_PLANNED	Planned Exe...		<input checked="" type="checkbox"/>		DEC	15	0	16	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
5	PARAHASH	Hash Value		<input checked="" type="checkbox"/>		CHAR	40	0	40	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
6	PARALENGTH	Length of P...		<input checked="" type="checkbox"/>		INT2	5	0	6	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
7	ODQ_CHANGEMODE	Change Mod...		<input type="checkbox"/>		CHAR	1	0	1	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		
8	ODQ_ENTITYCNTR	Number of ...		<input type="checkbox"/>		DEC	19	0	20	<input type="checkbox"/>	<input type="checkbox"/>		Inter...		

This SAP BW Datasource for event data reads data from database [Table ZRULERUN_EVENTS](#). It uses the generic extraction mode and is delta capable. The relevant database column for delta extractions is field “TST_PLANNED”.

7.3 Class zcl_rulerunner

7.3.1 Public method ADD_EVENT

Adds an event to the rulerunner® event-table.

If parameter *iv_suppress_exceptions* is set to 'X' no exceptions will be thrown.

Exceptions might be thrown in case the list of parameters is too large.

Signature:

```
class-methods add_event
importing
!iv_event_type                                type zrulerun_evtyp
        <!-- can be any string (up to 30 characters) -->
!iv_suppress_exceptions                        type abap_bool
        <!-- set to 'X' means that no exception is thrown
        Exceptions might be thrown in case the list of
        parameters is too large for database -->
!it_parameters                                type zrulerun_key_value_t optional
        <!-- optional table with key/value pairs that
        should be transferred to BRFplus functions-->
!iv_parameter_1_key                           type any optional
        <!-- can be any string (up to 30 characters) -->
!iv_parameter_1_value                         type any optional
        <!-- represents the value that belongs to key 1
        can be any string up to 60 chars-->
!iv_parameter_2_key                           type any optional
!iv_parameter_2_value                         type any optional
!iv_parameter_3_key                           type any optional
!iv_parameter_3_value                         type any optional
!iv_planned_execution_timestamp               type zrulerun_timestamp_pla
                                                optional
        <!-- timestamp in short format (yyyymmddhhmmss)
        events will only be processed in case that
        the actual time > this timestamp -->
!iv_bw_dtp_request                            type tyv_bw_request optional
        <!-- SAP BW DTP Request Number
        In a BW Transformation the Req.No is stored in
        variable REQUEST (iv_bw_dtp_request = request)
        is used to identify a Simulation Request-->
!iv_bw_dtp_delta_mode                         type tyv_bw_upd_mode optional
        <!-- is used to identify delta DTP's
        can be derived by the following code:
        data: lv_updatemode type RSBKUPDMODE.
        lv_updatemode = p_r_request->get_updmode( ).
        iv_bw_dtp_delta_mode = lv_updatemode-->
exporting
!ev_returncode                                type syst-subrc
        <!-- value 0 = success -->
!es_event_data                                type zrulerun_events_extended_s
        <!-- returns the record inserted into event table -->
raising zcx_rulerunner
.
```


Please note:

Key and Value fields are transferred and stored as they are.

No conversions happen.

Please take care of leading spaces and other pitfalls.

7.3.2 Public method DELETE_EVENTS

Deletes records from event and event-log tables.

Signature:

```
class-methods delete_events
importing
    iv_delete_unprocessed_events type abap_bool
        <!-- unprocessed (yet) events
            will only be deleted in case that
            this parameter is set to 'X' -->
    iv_delete_future_events      type abap_bool
        <!-- events with a planned timestamp in the future
            will only be deleted in case that
            this parameter is set to 'X' -->
    iv_test_mode                  type abap_bool default 'X'
    it_range_event_id             type tyt_range_event_id
        <!-- range table with event ID's to be deleted -->
    it_range_resultgroups        type tyt_range_resultgroups
        <!-- range table of Resultgroup to be deleted -->
    it_range_event_types         type tyt_range_event_types
        <!-- range table of event Types to be deleted -->
    it_range_timestamp_created    type tyt_range_timestamp
        <!-- range table of Timestamps Created to be deleted-->
    iv_timestamp_planned_max      type zrulerun_timestamp_pla
        <!-- events up to this Timestamp Planned are deleted-->
    it_range_timestamp_processed type tyt_range_timestamp
        <!--range table of Timestamps Processed to be deleted
            -->
    iv_display_messages           type abap_bool default 'X'
        <!-- Messages are displayed-->
exporting
    et_messages                  type tyt_messages
        <!-- Table of messages created during
            processing -->
raising zcx_rulerunner
.
```

7.3.3 Public method MOVE_DATA_SOURCE_TO_TARGET

Method moves data from a source to a target.

Data types are casted as decribed in chapter [3.5 Rulerunner® metadata conversion and BRFplus Result Data Object](#)

Signature:

```
class-methods move_data_source_to_target
importing
```

```

        io_source_data type any
exporting
        eo_target_data type any
raising   zcx_rulerunner.

```

7.3.4 Public method PROCESS_EVENT_DIRECTLY

Executes BRFplus functions in synchronous mode (for details see chapter [3.3.1 Synchronous execution of business rules](#)).

Signature:

```

class-methods process_event_directly
importing
    !iv_event_type          type zrulerun_evtyp
                                <!-- can be any string (up to 30 characters) -->
    !it_parameters          type zrulerun_key_value_t optional
                                <!-- optional table with key/value pairs that
                                    should be transferred to BRFplus functions-->
    !iv_parameter_1_key     type any optional
                                <!-- can be any string (up to 30 characters) -->
    !iv_parameter_1_value   type any optional
                                <!-- represents the value that belongs to key 1
                                    can be any string up to 60 chars-->
    !iv_parameter_2_key     type any optional
    !iv_parameter_2_value   type any optional
    !iv_parameter_3_key     type any optional
    !iv_parameter_3_value   type any optional
    !iv_resultgroup         type zrulerun_resultgroup optional
                                <!-- optional resultgroup, see also chapter
                                    Error: Reference source not found;
                                    can be any string up to 60 chars-->
    !it_resultgroups        type tyts_resultgroups optional
                                <!-- table of resultgroup, see also chapter
                                    Error: Reference source not found;
                                -->
exporting
    !ev_returncode          type syst-subrc
    !eo_result_data         type any
                                <!-- any data object, retrieves the BRFplus results
                                    see chapter
                                    Error: Reference source not found-->
raising   zcx_rulerunner.

```

7.3.5 Public method PROCESS_MULTIPLE_EVENTIDS

Method is intended to be used in SAP BW Transformations. It processes a list of existing EVENT ID's, that have been stored in rulerunner® Error: Reference source not found before (see [Adding events](#)). The list of EVENT ID's must be passed to the method. See also [3.3.2.6 Consuming rulerunner® events in a SAP BW scenario](#) .

Please note:

The method needs only the event ID. All other data (e.g. parameters) of an event are beeing read from the database.

Please note:

The method does **not check** the event's **planned execution** timestamp.

That has to be done when getting the event ID's.

Signature:

```
class-methods process_multiple_eventids
importing
    it_table_with_eventid type any table
        <!-- table must contain field EVENTID
        the EVENTID's should exist in rulerunner® table
        ZRULERUN_EVENTS, if so they are processed -->
    iv_update_processing_log type abap_bool default 'X'
        <!-- if 'X' then processing log is stored in
        Error: Reference source not found -->
    iv_repeat_processing type abap_bool
        <!-- 'X' = processing BRFplus functions per event
        that have been processed yet-->
    iv_package_size type I
        <!-- defines, how many EVENT ID's are read and
        processed in one package-->
    iv_resultgroup type zrulerun_resultgroup optional
        <!-- optional resultgroup, see also chapter
        Error: Reference source not found;
        can be any string up to 60 chars-->
    it_resultgroups type tyts_resultgroups optional
        <!-- table of resultgroup, see also chapter
        Error: Reference source not found;
        -->
exporting
    eo_result_data type any
        <!-- any data object, retrieves the BRFplus results
        see chapter
        Error: Reference source not found-->
raising zcx_rulerunner.
```

7.3.6 Public method PROCESS_STORED_EVENTS

The Method processs stored rulerunner® events.

The events must have been added to the rulerunner® event queue using [Public method ADD_EVENT](#) see ([Adding events](#)).

Signature:

```
class-methods process_stored_events
importing
    !iv_package_size type tyv_package_size
        <!-- defines, how many EVENT ID's are read and
        processed in one package-->
    !iv_run_packetised type abap_bool
        <!-- see chapter Error: Reference source not found Error: Reference
source not found
        -->
```

```

!iv_timestamp_planned_from type zrulerun_timestamp_pla
    <!-- lower limit of a planned timestamp of an event;
    used as a filter of events -->
!iv_timestamp_planned_to type zrulerun_timestamp_pla
    <!-- upper limit of a planned timestamp of an event;
    used as a filter of events -->
!iv_event_type type zrulerun_evtyp
    <!-- can be any string (up to 30 characters) -->
!it_event_type_range type tyt_range_event_types optional
    <!-- range table of event Types -->
!iv_resultgroup type zrulerun_resultgroup optional
    <!-- optional resultgroup, see also chapter
    Error: Reference source not found;
    can be any string up to 60 chars-->
!it_resultgroups type tyts_resultgroups optional
    <!-- table of resultgroup, see also chapter
    Error: Reference source not found;
    -->
Error: Reference source not found type abap_bool
    <!-- if 'X' then processing log is stored in
    Error: Reference source not found -->
iv_repeat_processing type abap_bool
    <!-- if 'X' then processing log is stored in
    Error: Reference source not found -->
iv_update_delta_timestamp type abap_bool
    <!-- see chapter Error: Reference source not found Error: Reference
source not found-->
iv_delta_mode type tyv_delta_mode
    <!-- see chapter Error: Reference source not found Error: Reference
source not found-->
iv_test_mode type abap_bool
    <!-- see chapter
    Error: Reference source not found Error: Reference
source not found-->
exporting
!et_event_id type tyt_event_id
    <!-- table with EVENT master data
    ( but w/o parameters ) -->
!eo_result_data type any
    <!-- any data object, retrieves the BRFplus results
    see chapter
    Error: Reference source not found-->
!ev_no_more_data type abap_bool
raising zcx_rulerunner.

```

7.3.7 Public method SELECT_FOR_ALL_ENTRIES_IN

This method is to be used directly in BRFplus.

It provides an ABAP OPEN SQL “For All Entries In” functionality inside a BRFplus function.

For details see chapter [5.3 BRFplus For All Entries In – enhancement](#).

Signature:

```
class-methods select_for_all_entries_in
importing
    iv_select_from_table_name      type any
    it_for_all_entries_table       type any table
    iv_for_all_entries_tablename   type any
    iv_where_field_1_db           type any
    iv_where_field_1_for_all       type any
    iv_where_field_2_db           type any optional
    iv_where_field_2_for_all       type any optional
    iv_where_field_3_db           type any optional
    iv_where_field_3_for_all       type any optional
    iv_where_field_4_db           type any optional
    iv_where_field_4_for_all       type any optional
    iv_where_field_5_db           type any optional
    iv_where_field_5_for_all       type any optional
exporting
    et_result_data                type any table
raising
    zcx_rulerrunner.
```

7.3.8 Public method SHOW_RULERUNNER_CUSTOMIZING

Method opens up the BRFplus Workbench. For details see chapter [5.2 Jump to BRFplus from ABAP Editor -enhancement](#).

Signature:

```
class-methods show_rulerunner_customizing
importing
    iv_brf_component_id type zrulerun_functionid optional
    <!-- BRFplus Technical Object ID;
    if supplied then the BRFplus object opens up-->
raising
    zcx_rulerrunner.
```

7.4 Class zcx_rulerrunner

Global exeption class used throughout rulerunner® framework.

7.5 Dictionary objects

7.5.1 Table ZRULERUN_DELTA

Contains delta timestamps per EVENT_TYPE and Resultgroup, see chapter [3.3.2.5 Consuming rulerunner®events in ABAP](#).

Dictionary: Display Table

Transparent Table: ZRULERUN_DELTA Active

Short Description: Eventid: Delta Timestamps

Attributes | Delivery and Maintenance | Fields | Input Help/Check | Currency/Quantity Fields

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
EVENTTYPE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_EVTYP	CHAR	30	0	Event Type
RESULTGROUP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_RESULT...	CHAR	30	0	ZRULERUN_RESULTGROUP
TST_PLANNED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_TIMEST...	DEC	15	0	Planned Execution-Timestamp(short)

7.5.2 Table ZRULERUN_EVENTS

Table contains rulerunner® events that have been added to the rulerunner® event queue via [Public method ADD_EVENT](#).

Dictionary: Display Table

Transparent Table: ZRULERUN_EVENTS Active

Short Description: Events

Attributes | Delivery and Maintenance | Fields | Input Help/Check | Currency/Quantity Fields

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3	0	Client
EVENTID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_EVTID	NUMC	18	0	Event ID
EVENTTYPE	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_EVTYP	CHAR	30	0	Event Type
TST_CREATED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_TIMEST...	DEC	15	0	Created-Timestamp
TST_PLANNED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_TIMEST...	DEC	15	0	Planned Execution-Timestamp(short)
TST_PROCESSED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_TIMEST...	DEC	15	0	Processed-Timestamp
PARAHASH	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_PARAHA...	CHAR	40	0	Hash Value
PARALENGTH	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_PARALE...	INT2	5	0	Length of Parameter-BLOB
PARAJSON	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_PARAJS...	LCHR	2000	0	Parameter in JSON format

7.5.3 Table ZRULERUN_PLOG

Contains the rulerunner® Processing Log per EVENT_ID, Resultgroup and BRFplus function_ID. Is only updated if import parameter `iv_update_processing_log` is set to 'X' when calling [Public method PROCESS_STORED_EVENTS](#) or [Public method PROCESS_MULTIPLE_EVENTIDS](#). For details see chapter [3.3.2.5 Consuming rulerunner®events in ABAP](#).

Dictionary: Display Table

←

→

Technical Settings Indexes... Append Structure...

Transparent Table ZRULERUN_PLOG Active

Short Description RuleRunner: Event-Processing Timestamps

Attributes Delivery and Maintenance Fields Input Help/Check Currency/Quantity Fields

Srch Help Predefined Type

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
CLIENT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDT	CLNT	3		0Client
EVENTID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_EVTID	NUMC	18		0Event ID
RESULTGROUP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_RESULT...	CHAR	30		0ZRULERUN_RESULTGROUP
FUNCTIONID	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZRULERUN_FUNCTI...	CHAR	32		0Function ID (BRF)
TST_PROCESSED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_TIMEST...	DEC	15		0Processed-Timestamp
SKIPPED	<input type="checkbox"/>	<input type="checkbox"/>	ZRULERUN_SKIPPED	CHAR	1		0Skipped

7.6 Message class

Message class ZRULERUNNER_MSG contains all messages.