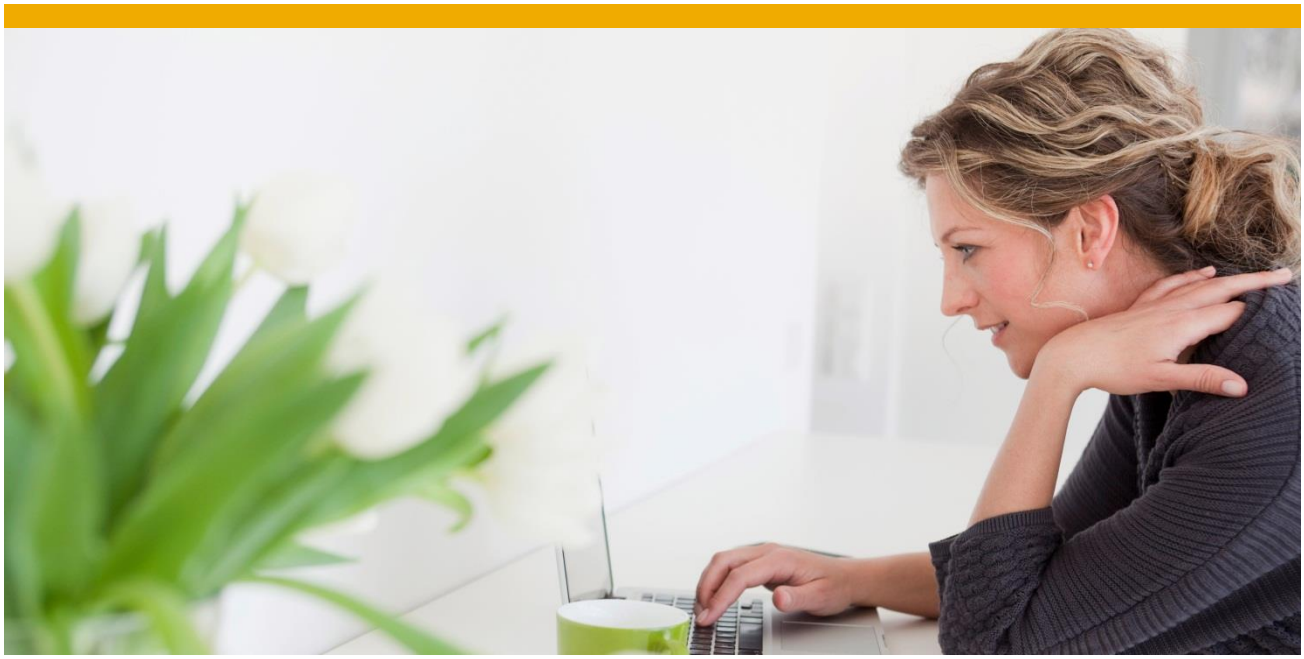


SAP Decision Service Management Test Case Tool

A brief introduction

TABLE OF CONTENTS

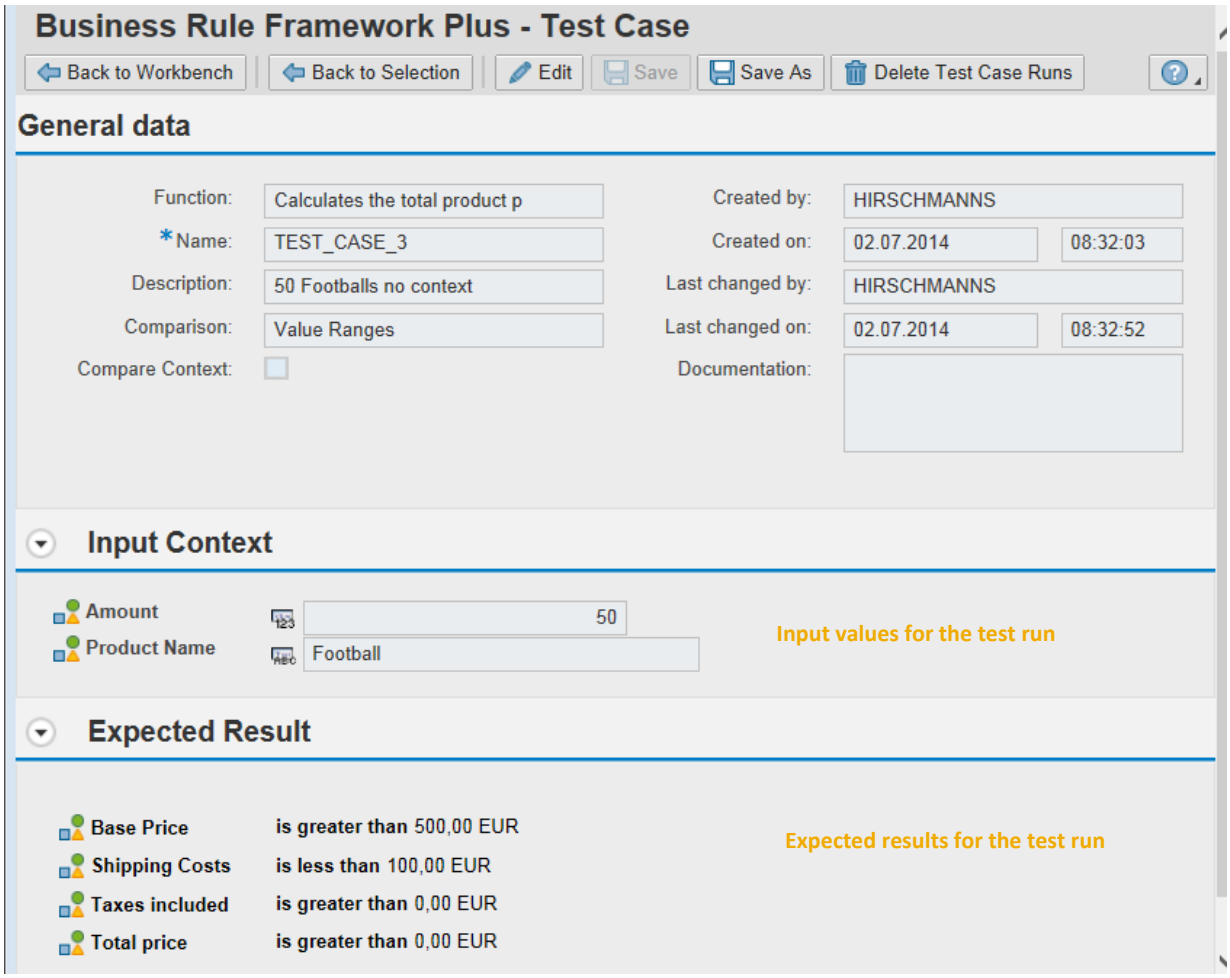
INTRODUCTION

SAP Decision Service Management (DSM) supports organizations in automating business decisions in their applications. Often, the decision making process is not trivial. Therefore, testing the rules that make the decisions is a must.

DSM already provides tools for testing decision services:

1. *Simulation:* Allows you to check the result a *decision service* returns for a given input. This is great to do a quick test of changes while modeling the business rules.
2. *Debugger:* Provides capabilities to step through the rules execution, inspect and change variables and pause on specific states. With these options, users can identify errors (“bugs”) in the business rules.

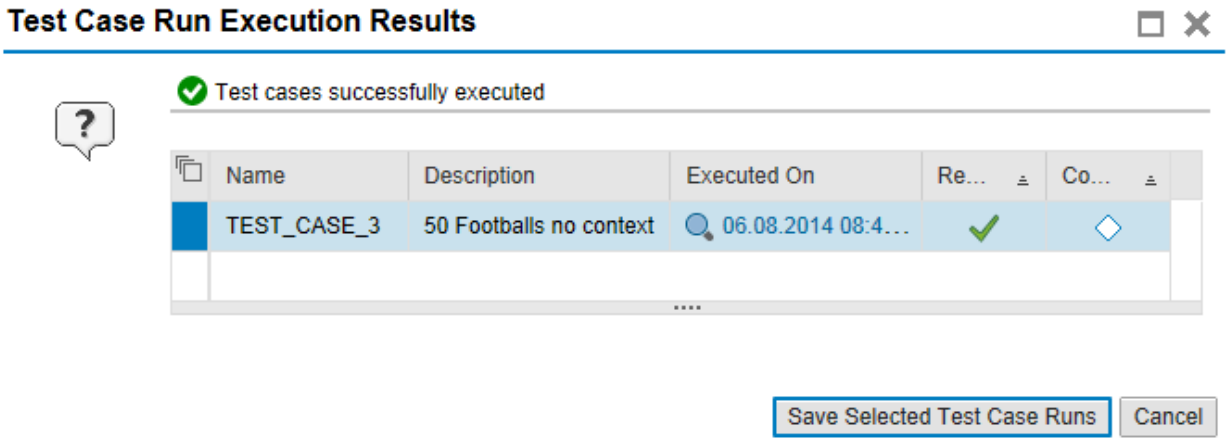
In both cases it is necessary to manually execute the test and understand inputs and results to assess if the decision service is running correctly for a particular combination of values. This can become a very time consuming exercise. The better approach is to define sets of test cases that get executed more easily or even regularly. This is where the Test Case Tool comes in handy. It provides the infrastructure to define inputs and expected outputs for decision services. It is very well integrated with DSM and the BRFplus workbench. With note 2056709 “Test Case Tool - Enhancements for test results (UI)” the latest set of features for the test case tool was released.



1 Test Case Details

Once a test case is created, it can be executed with a few mouse clicks. It is also possible to completely automate its execution with some ABAP code using the Test Case Tool's API. By doing so decision service testing can be performed continuously while rules are being changed without any additional effort. Furthermore, test cases provide a means of documentation so that the purpose of the test and the possible results are easy to understand even by persons other than the creator of the particular test case. Finally, the Test Case Tool does not only allow for comparing results of a decision service with a pre-defined expected result. Rather, it also allows the definition of result ranges and even custom comparison logic with the help of an ABAP class.

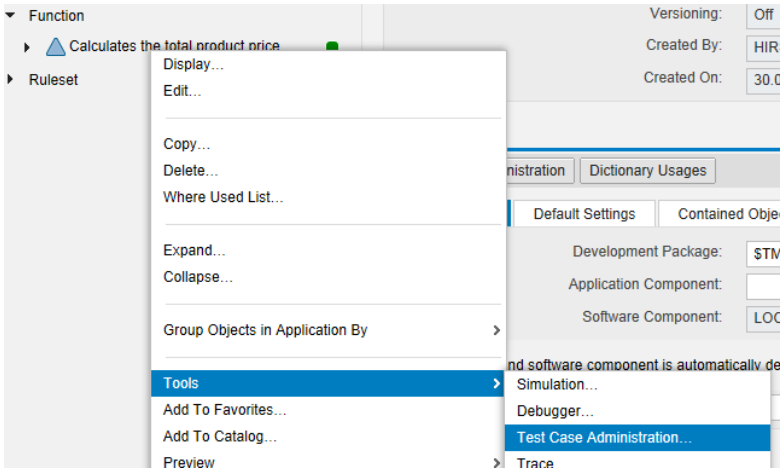
Each time a test case is executed a *test case run* will be created which can be saved in the database. Newer runs can be compared with older ones to see how rule modifications impact the results.



2 Overview Popup for a Test Case Run

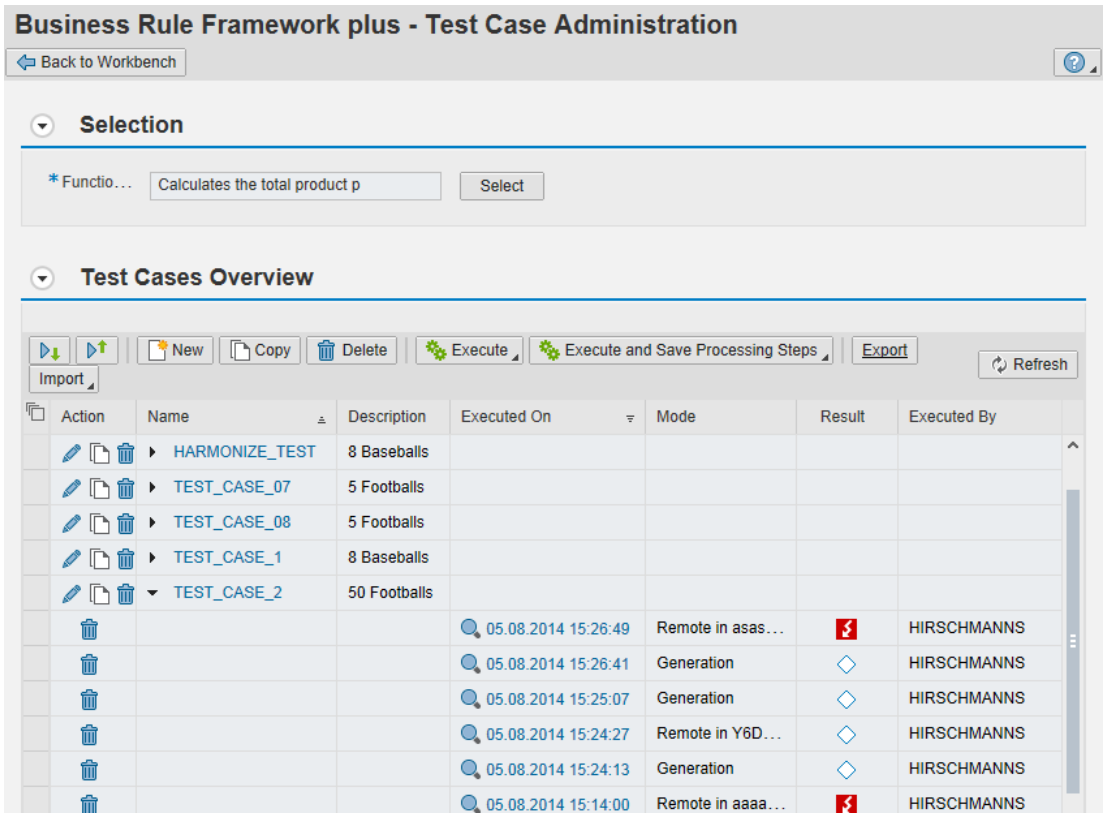
USING THE TEST CASE TOOL

With the test case tool it is possible to create and maintain test cases for decision services (BRFplus functions). Each test case is assigned to a function. In the BRFplus workbench the test case tool can be opened either from the *Tools* menu or via the corresponding item in a function's context menu in the navigation pane of the BRFplus workbench.



3 Open the Test Case Tool

The following screenshot shows the test case tool after startup.



4 The Main Screen

In the *Selection* section, the function (decision service) to work with can be chosen or changed by pressing the *Select* button. It is not necessary to navigate back to the workbench if you want to test another function.

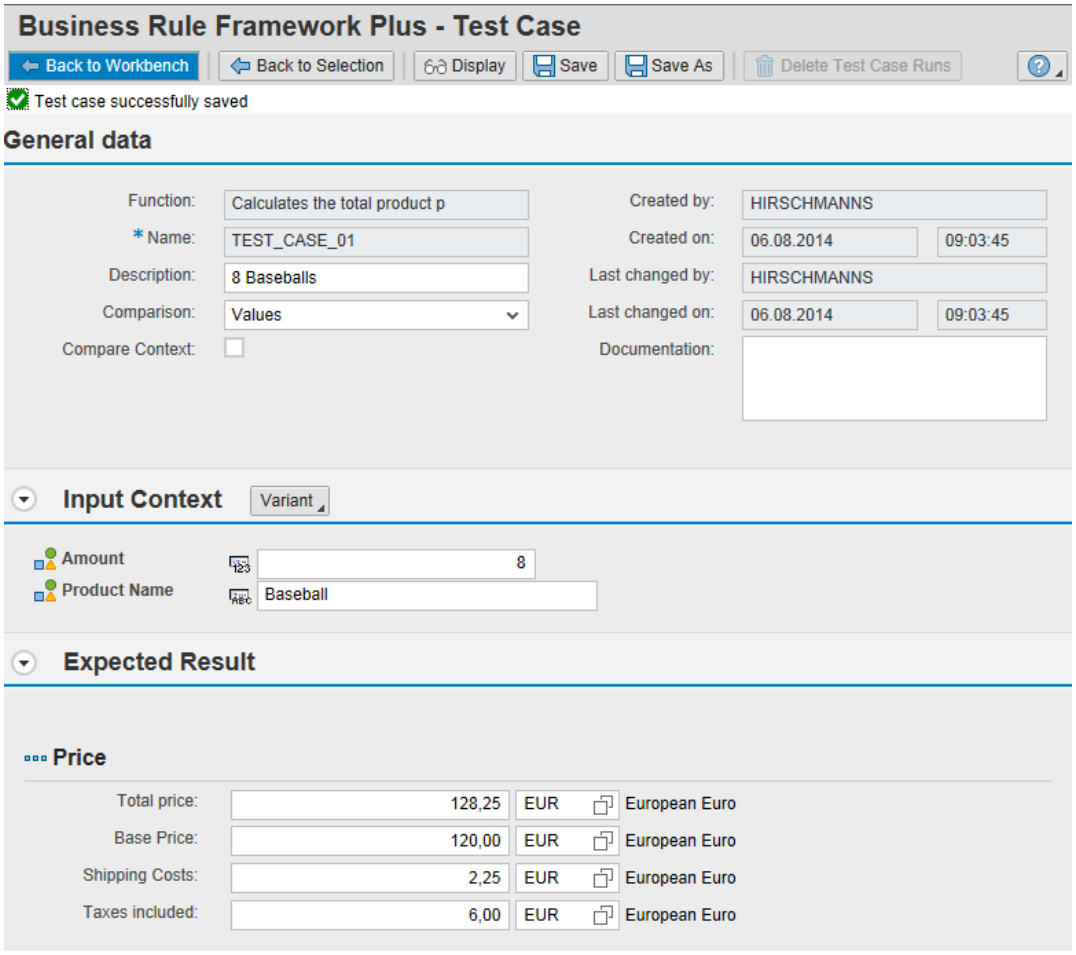
In the *Overview section*, a list of all test cases and test case runs is shown.
The following options are provided:

- Create a new test case
- Execute a test case
- Maintain a test case (copy, delete, modify)
- Import or export a test case

Each of these options is explained in more detail in the following chapters.

CREATING A NEW TEST CASE

After pressing the *New* button, the test case screen with all details is shown.



5 Test Case Details

The *General Data* Section

Test cases must have a name. Optionally they can also have a description and they can be documented. Not only the result but also the changed context can be included into a test case. Note, that some decision services may also update their context values. You can have the test case tool watch these values by ticking the *Compare Context* option.

A very important property is the comparison mode that defines how the result will be checked for correctness. The following modes are available:

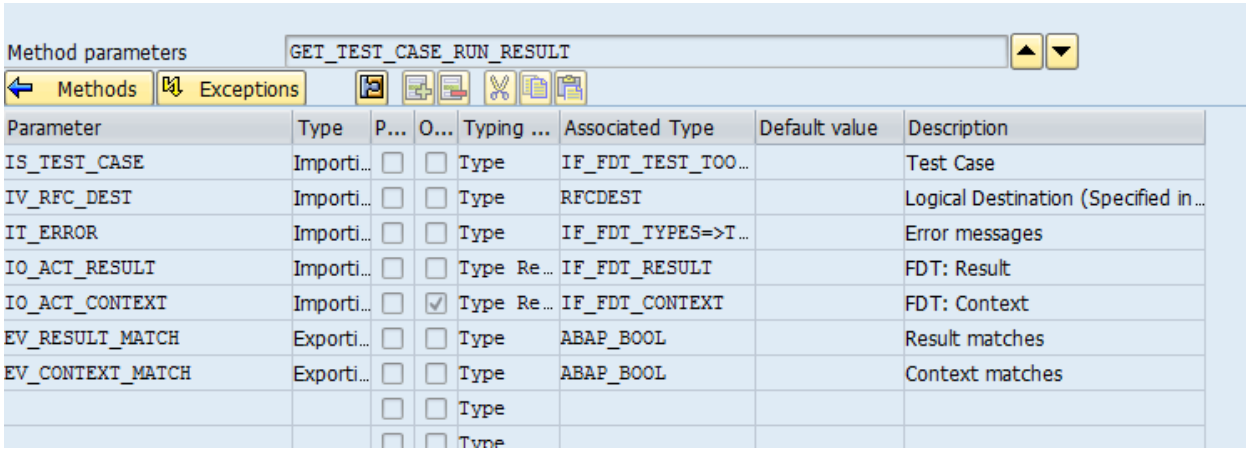
- Value
  - Compares each result/context element with a fixed value.
    For example: *'the element should equal 10'*
- Value Range
  - Compares each result/context element with a value range.
    For example: *'the element's value should be between 10 and 50'* or
    'the element's value should be less than 50'
- Exception:
  - Checks if an exception occurs while executing the function. A test case run can be successful when no exceptions occurs, or when an exception occurs.
- Custom Class

- It is possible to implement a comparison in an ABAP class. This is useful when the checks are very complex or dynamic.

The *Input Context* Section
The context element values are defined.

The *Expected Result /Context* Section
The content of the *'Expected Result'* and the '*Expected Context*' (appears only if *Compare Context* is checked) depends on the comparison mode:

- If '*Value'* or '*Value Range'* is chosen, there will be a list of all result/context elements, where the expected value/value range can be entered.
- If '*Exception*' *is chosen*, no context comparison is available. In this case, you can simply choose if you expect an exception, or not.
- If '*Custom-Defined'* is chosen, it is required to enter the name of an ABAP class implementing the interface *'IF_FDT_TEST_TOOL_EXITS'* for result comparison. The interface contains the method '*GET_TEST_CASE_RUN_RESULT'*.



Parameter	Type	P...	O...	Typing ...	Associated Type	Default value	Description
IS_TEST_CASE	Importi...	☐	☐	Type	IF_FDT_TEST_TOO...		Test Case
IV_RFC_DEST	Importi...	☐	☐	Type	RFCDEST		Logical Destination (Specified in ...
IT_ERROR	Importi...	☐	☐	Type	IF_FDT_TYPES=>T...		Error messages
IO_ACT_RESULT	Importi...	☐	☐	Type Re...	IF_FDT_RESULT		FDT: Result
IO_ACT_CONTEXT	Importi...	☐	☑	Type Re...	IF_FDT_CONTEXT		FDT: Context
EV_RESULT_MATCH	Exporti...	☐	☐	Type	ABAP_BOOL		Result matches
EV_CONTEXT_MATCH	Exporti...	☐	☐	Type	ABAP_BOOL		Context matches
		☐	☐	Type			
		☐	☐	Type			

6 Signature of the method

The result returned by the decision service is provided in variable *'IO_ACT_RESULT'*. In case *Context Comparison* was selected, variable '*IO_ACT_CONTEXT'* holds the values of the changed context.
The following demo implementation checks, if the result and the context both are greater than 5.

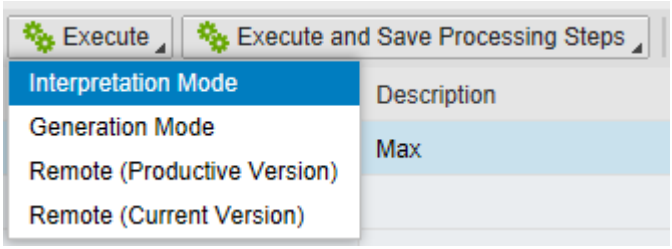
IF_FDT_TEST_TOOL_EXITS~GET_TEST_CASE_RUN_RESULT — Active

```
METHOD if_fdt_test_tool_exits~get_test_case_run_result.
  DATA: lv_res TYPE i.
  TRY .
      io_act_result->get_value( IMPORTING ea_value = lv_res ).
      IF lv_res > 5.
        ev_result_match = abap_true.
      ENDIF.
    CATCH cx_root .
      ev_result_match = abap_false.
  ENDTRY.

  IF io_act_context IS BOUND.
    TRY .
        io_act_context->get_value( EXPORTING iv_name = 'amount' IMPORTING ea_value = lv_res ).
        IF lv_res > 5.
          ev_context_match = abap_true.
        ENDIF.
      CATCH cx_root .
        ev_context_match = abap_false.
    ENDTRY.
  ENDIF.

ENDMETHOD.
```

7 Sample Implementation for a result check method

EXECUTING TEST CASES

The test case tool reuses the BRFplus simulation infrastructure . Consequently, the same simulation modes are available:



8 The execution modes

1. Interpretation Mode:
   In this mode the system uses the objects involved in the simulation by drilling down through their design-time definition. For a quick one-time simulation run, this can be faster than using generation mode.
2. Generation Mode:
   In this mode, if there is no source code available for the function, the system first generates source code for the objects involved in the simulation.
3. Remote (Productive Version):
   In this mode, the rules are executed in another system (a so-called managed system). For this, the rules are used as they exist in the managed system.
4. Remote (Current Version)
   In this mode, the rules, as they exist in the current system, are temporarily transmitted to and executed in the managed system.

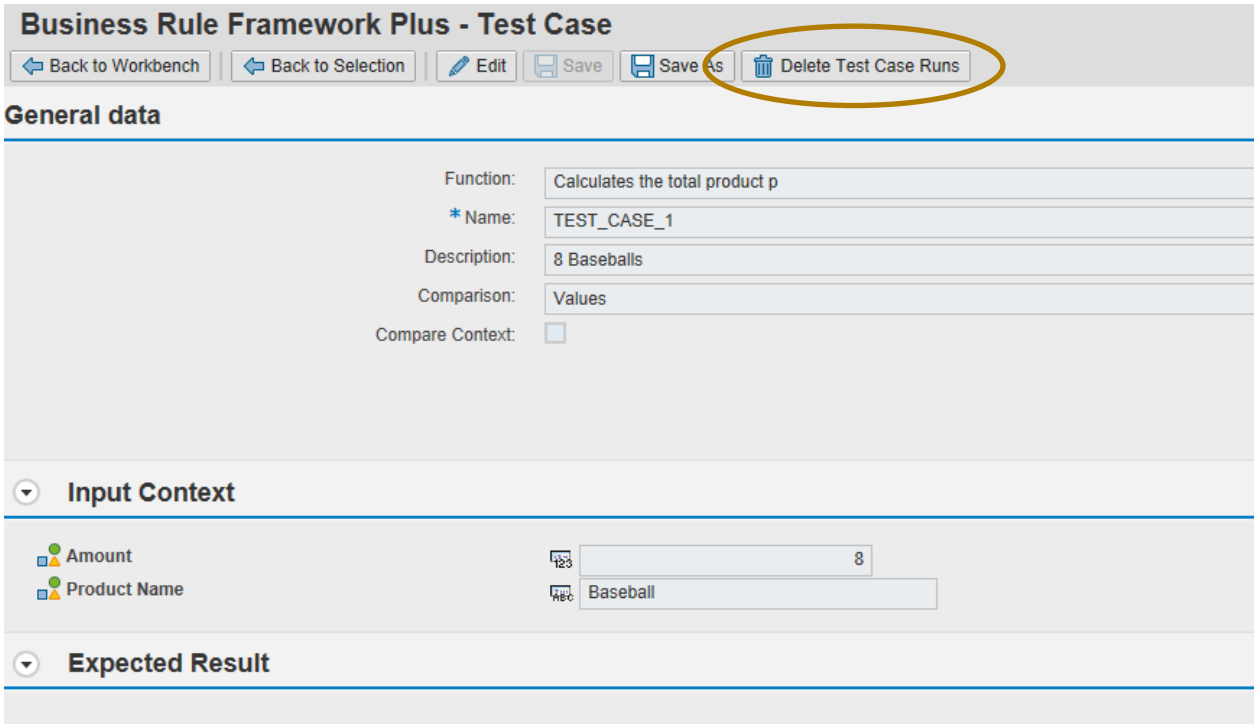
When a remote mode is used, the managed system has to be specified.

After execution, a popup shows the result of the test case (see figure 2). When context comparison is enabled, the test case will only be successful if both context and result comparison were successful.
By clicking on the execution date, it is possible to navigate to the detail view for test case runs.

MAINTAINING TEST CASES

To copy, delete, or modify a test case, simply use the icons in the first column of the related row.
Notice, that it is not possible to change a test case, if on e or more runs for that test case already exist.
If you want to modify a test case, which has already been executed, you need either to delete the runs or to copy the test case (so that you can modify the copy).

To delete all runs of a test case, press the modify-Button at the main screen. This will lead you to the detail view for test cases where the runs can be deleted.



9 Deleting all Test Case Runs

EXPORT/IMPORT OF TEST CASES

Test cases can be exported to a file. The file can then be uploaded (imported) subsequently. This is helpful when test cases have to be moved from one system to another or from one decision service (function) to another. When importing a test case for another function, matching parts of the context are identified based on data object names. These matching parts are taken over into the importing system. Note that an imported test case will replace an existing test case if their names are equal.

There are 2 ways to import a test case:

- If both functions are in the same system, it is possible to use the ‘Import from Function’ feature. The main screen provides a corresponding button to select the source and to start the ‘Import’.
- If the functions are in different systems, it is necessary to first ‘Export’ the test case to a zip file, which can then be imported again into the new function.

AUTOMATED TESTING

Automated testing offers the possibility to create, change, execute etc. test cases from within ABAP coding without using the test case tool UI. The interface *'IF_FDT_TEST_TOOL'* contains all available methods.

Interface IF_FDT_TEST_TOOL Implemented / Active

Properties | Interfaces | Attributes | Methods | Events | Types | Alias

Parameters | Exceptions | Properties

Method	Level	Description
CREATE_TEST_CASE	Instance Method	Create a test case
EXECUTE_TEST_CASE	Instance Method	Execute a test case
READ_TEST_CASE	Instance Method	Read a test case
UPDATE_TEST_CASE	Instance Method	Update a test case
READ_TEST_CASE_LIST	Instance Method	Get a list of test cases
COPY_TEST_CASE	Instance Method	Copy test case to another name
DELETE_TEST_CASE	Instance Method	Delete one or more test case(s)
DELETE_TEST_CASE_RUN	Instance Method	Delete one or more test case run(s)
EXECUTE_TEST_CASE_BUNDLE	Instance Method	Execute a list of test cases
READ_TEST_CASE_RUN	Instance Method	Read the test case runs of a test case
EXPORT_TO_ZIP	Instance Method	Export test cases into a ZIP-file
IMPORT_FROM_ZIP	Instance Method	Import test cases from a ZIP-file

10 Methods of interface IF_FDT_TEST_TOOL

The most important methods are probably EXECUTE_TEST_CASE and CREATE_TEST_CASE. The following section shows how to use them.

AUTOMATED TEST CASE EXECUTION

Method EXECUTE_TEST_CASE is used to automatically execute an existing test case. It has the following signature:

Parameters of Method EXECUTE_TEST_CASE

Methods | Exceptions | Properties

Parameter	Type	P...	O...	Typing Method	Associated Type
IV_FUNCTION_ID	Importing	☐	☐	Type	IF_FDT_TYPES=>ID
IV_NAME	Importing	☐	☐	Type	IF_FDT_TYPES=>NAME
IV_MODE	Importing	☐	☐	Type	STRING
IV_TRACE	Importing	☐	☑	Type	ABAP_BOOL
IV_RFCDEST	Importing	☐	☑	Type	RFCDEST
IV_PERSIST	Importing	☐	☑	Type	ABAP_BOOL
ES_RUN	Exporting	☐	☐	Type	S_TEST_CASE_RUN
ET_ERROR	Exporting	☐	☐	Type	IF_FDT_TYPES=>T_MESSAGE
		☐	☐	Type	

11 Signature of method EXECUTE_TEST_CASE

The importing parameters function id (IV_FUNCTION_ID), the name of the test case (IV_NAME) and the execution mode (IV_MODE) are mandatory. These values have to be provided in the ABAP code such as in the following example.

```
DATA:
  lo_test_tool    TYPE REF TO if_fdt_test_tool,
  lv_function_id TYPE fdt_uuid VALUE '005056A501951ED48089DC9214B75651',
  lv_test_case    TYPE fdt_name VALUE 'TEST_CASE_07',
  lv_mode         TYPE string VALUE if_fdt_test_tool=>gc_mode_local_generation.
```

12 Definition of mandatory importing parameters

The importing parameter IV_RFCDEST has to be filled with the RFC destination, if the function is located in a remote system.
IV_PERSIST has to be set to ABAP_TRUE to save the test case run result in the database. Afterwards the test case will show up in the test case tool. Otherwise, the result will be returned as well but it is up to the caller to decide where and how to save.
IV_TRACE has to be set to ABAP_TRUE to write and add execution trace to the test case. Also traces can be persisted.

The following code snippet shows how to get an instance of the test case tool object and how to execute a test case.

```
*   Get test tool
    lo_test_tool = cl_fdt_factory=>get_instance( )->get_test_tool( ).

*     Test case execution
    lo_test_tool->execute_test_case(
        EXPORTING iv_function_id = lv_function_id
                  iv_name        = lv_test_case
                  iv_mode        = lv_mode
*                  IV_TRACE       =
*                  IV_RFCDEST     =
*                  IV_PERSIST     =
        IMPORTING es_run         = ls_run
                  et_error       = lt_error ).
```

13 Execution of method EXECUTE_TEST_CASE

The result is returned in structure ES_RUN. Components RESULT_COMP and CONTEXT_COMP are both fields of the type IF_FDT_TEST_TOOL=> COMPARISON_MODE. The possible values for these fields are defined by the constants IF_FDT_TEST_TOOL=>GC_COMPARISON_*. For example: Component RESULT_COMP has the value IF_FDT_TEST_TOOL=>GC_COMPARISON_SUCCESS, if the returned values match the expected results.

AUTOMATED TEST CASE CREATION

To create test cases from ABAP code method *'CREATE_TEST_CASE'* can be used.

Parameters of Method: CREATE_TEST_CASE

Methods | Exceptions | Properties

Parameter	Type	P...	O...	Typing Method	Associated Type
IV_FUNCTION_ID	Importing	☐	☐	Type	IF_FDT_TYPES=>ID
IV_NAME	Importing	☐	☐	Type	IF_FDT_TYPES=>NAME
IV_DESCRIPTION	Importing	☐	☑	Type	IF_FDT_TYPES=>TEXT
IO_INPUT_CONTEXT	Importing	☐	☑	Type Ref To	IF_FDT_CONTEXT
IO_EXPECTED_CONTEXT	Importing	☐	☑	Type Ref To	IF_FDT_CONTEXT
IO_EXPECTED_RESULT	Importing	☐	☑	Type Ref To	IF_FDT_RESULT
IS_RESULT_OPTIONS	Importing	☐	☑	Type	S_RES_OPTIONS
IV_COMPARE_CONTEXT	Importing	☐	☑	Type	ABAP_BOOL
IV_DOCUMENTATION	Importing	☐	☑	Type	STRING
ET_ERROR	Exporting	☐	☐	Type	IF_FDT_TYPES=>T_MESSAGE

14 Signature of method CREATE_TEST_CASE

The input parameters function id (IV_FUNCTION_ID) and the name of the test case (IV_NAME) are mandatory. All other input parameters are optional and are needed depending on the use case. E.g., the expected context (IO_EXPECTED_CONTEXT) is only needed, if it shall be compared to the context after test case execution.

The following code snippet shows how to create a context object with values.

```
DATA:  lo_context        TYPE REF TO if_fdt_context.

       * get input context
       lo_context = cl_fdt_factory=>get_instance( )->get_function(
                      iv_id = lv_function_id )->get_process_context( ).

       * fill input context
       lo_context->set_value( iv_name = 'AMOUNT' ia_value = 5 ).
       lo_context->set_value( iv_name = 'PRODUCT_NAME' ia_value = 'FOOTBALL' ).
```

15 Fill input context

Afterwards the expected result has to be defined.

```
* get result object
  lo_result = cl_fdt_convenience=>get_function_result( iv_function_id = lv_function_id ).

* fill expected result
  ls_total_price-number    = '100'.
  ls_total_price-currency  = 'EUR'.
  lo_result->set_value( iv_name = 'PRICE-TOTAL_PRICE' ia_value = ls_total_price ).

  ls_base_price-number    = '80'.
  ls_base_price-currency  = 'EUR'.
  lo_result->set_value( iv_name = 'PRICE-BASE_PRICE' ia_value = ls_base_price ).

  ls_shipping_costs-number    = '15'.
  ls_shipping_costs-currency  = 'EUR'.
  lo_result->set_value( iv_name = 'PRICE-SHIPPING_COSTS' ia_value = ls_shipping_costs ).
```

```
  ls_taxes-number    = '5'.
  ls_taxes-currency  = 'EUR'.
  lo_result->set_value( iv_name = 'PRICE-TAXES' ia_value = ls_taxes ).
```

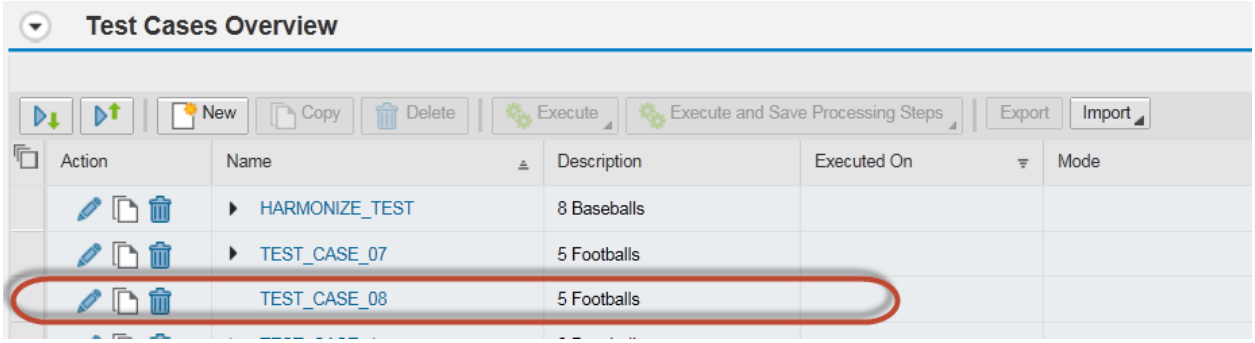
16 Fill expected result

Finally the test case can be created as follows.

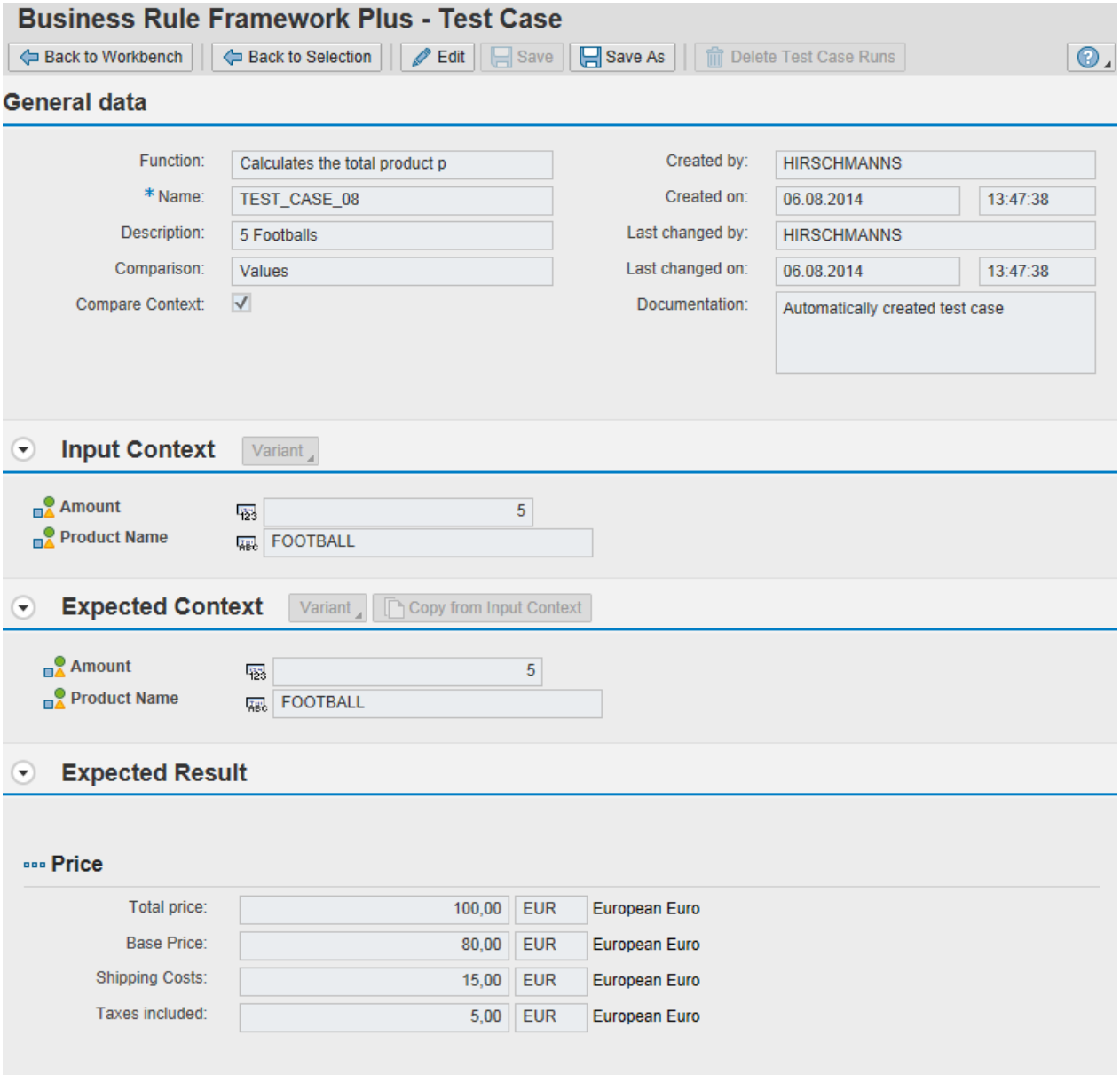
```
* Get test tool
  lo_test_tool = cl_fdt_factory=>get_instance( )->get_test_tool( ).

* create test case
  lo_test_tool->create_test_case(
    EXPORTING iv_function_id      = lv_function_id
              iv_name             = lv_test_case_name
              iv_description      = lv_description
              io_input_context    = lo_context
              io_expected_context = lo_context
              io_expected_result  = lo_result
*              is_result_options   =
              iv_compare_context  = abap_true
              iv_documentation    = lv_documentation
    IMPORTING et_error            = lt_error ).
  IF lt_error IS NOT INITIAL.
   WRITE 'Test Case could not be created, because the following error occurred'.
   LOOP AT lt_error INTO ls_error.
     WRITE: / ls_error-text.
   ENDLOOP.
  ENDIF.
```

17 Create test case

After the test case has been created, it is visible in the test case tool.



18 Created test case in test case tool overview



19 Details of created test case in test case tool

Now the automatically created test case can be executed, either using ABAP code or using the test case tool.

www.sap.com