Infor LN BFlow Development Guide

Publication Information

Release: 10.5

Publication Date: June 22, 2016

Document number: lnbflowdg (U9874)

Contents

About this Guide

Document Summary

This document describes how developers can use the BFlow test framework to test application business logic.

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal.

If we update this document after the product release, we will post the new version on this website. We recommend that you check this website periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Introduction

1

BFlow (Business Flow) is a Business Process Validation tool for LN. BFlow is intended to verify that end-to-end business logic functions as expected.

A Business Process is a collection of related, structured activities or tasks that produce a specific service or product, to serve a particular goal, for a particular customer or customers.

LN consists of sessions that implement the business logic. An LN user uses these sessions to perform different steps of a business process.

To replicate these steps in BFlow, the user input must be translated into specific actions that are collected in scripts. These scripts can be executed independently or may be combined to create more complex test scenarios, that is, a hierarchy. This way the steps of the user can be reproduced.

BFlow is intended to implement higher level, more functional test scenarios from a user's perspective. BFlow impersonates the UI, that is, it replaces the regular UI.

BFlow scripts generate an extensive log, which can be used for analysis.

You can use BFlow to uncover new software bugs, or regressions, in existing areas of the system after changes such as enhancements, patches, or configuration changes.

The next chapters describe the building blocks of BFlow and how to develop and execute them.

The last chapters explain the underlying concepts and list all the available actions.

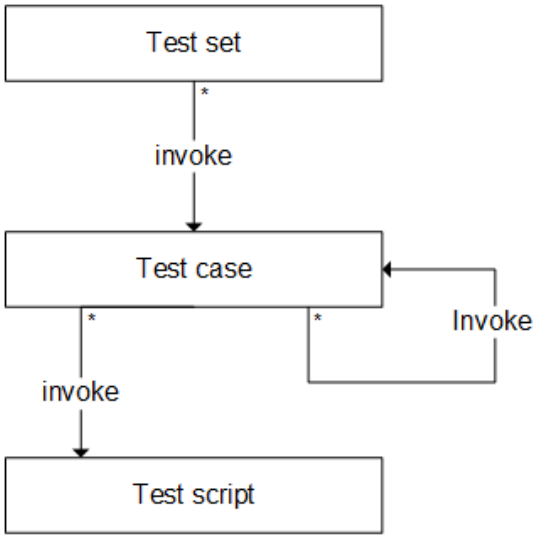
Test script, test case, and test set

2

To manage the different test scenarios a distinction was made between scripts that interact with sessions, scripts that construct business flows, and scheduled execution of collected scripts.

BFlow supports aggregation, which allows the creation of complex test scenarios and re-use of specific test patterns.

This diagram shows the relationship between these components:



The BFlow test scripts and test cases are stored as additional files, which are software components, and are subject to version control (VRC).

- BFlow test scripts must have the `.bflow` extension.
- BFlow test cases must have the `.bfcase` extension.

The test set is a collection of test cases and is stored as table data.

Test script

The test script is the smallest building block.

This script allows all possible interaction with a session. It represents a set of user interface actions, for example:

- Starting a session
- Adding, modifying, deleting data
- Executing form commands
- Using buttons
- Saving the changes

Test scripts are stored as additional files with the `.bflow` extension. You can develop test scripts in LN Studio.

Test case

A test case cannot interact with sessions. The test case is used to define a business process, or a part of it, by calling test scripts or other test cases. The test case can pass arguments to the called test script or test case, and retrieve changed arguments and return value after execution. The test script or test case can be called multiple times with different input arguments. This test case can also be called in test cases. This provides flexibility to create reusable test blocks.

Test cases are stored as additional files with the `.bfcase` extension. You can develop test cases in LN Studio.

Test set

A test set is a collection of dependent or independent test cases which can be scheduled for execution. Within a test set, test cases can be activated or inactivated, and can be run sequentially or in parallel.

This information is stored in tables.

You can maintain test sets in the Test Sets (tlrgt1130m000) session.

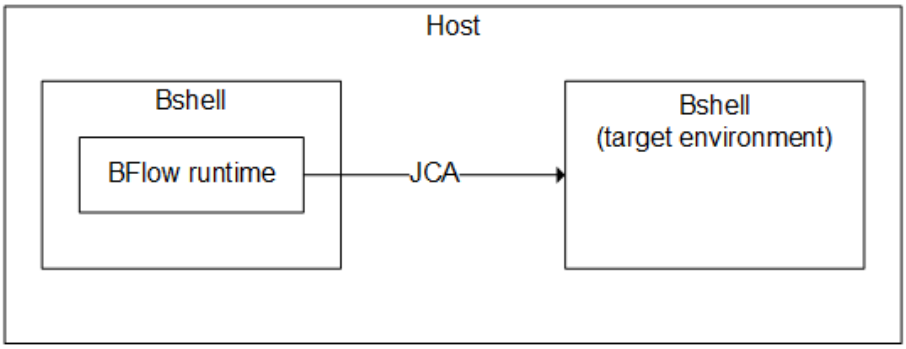
BFlow environment

3

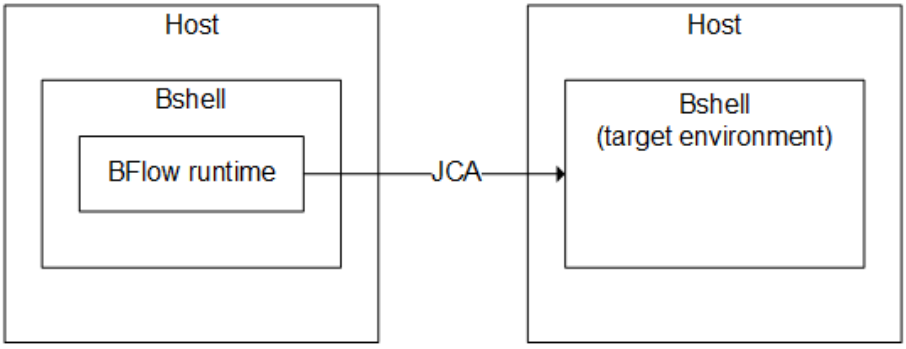
The BFlow environment consists of two major parts: the BFlow runtime and the target environment.

The BFlow runtime runs in a bshell and connects to the target environment using JCA. This connection is similar to logging on to LN using LN UI.

The BFlow runtime and target environment typically run on the same host, in different bshells. This diagram shows this scenario:



The BFlow runtime can also connect to a target environment on another host using the Reconnect action. This diagram shows this scenario:



Between BFlow and the target environment the LN UI protocol is used.

The target environment cannot distinguish between the BFlow runtime and a regular LN user.

Records and occurrences

4

There is an important distinction between records and occurrences.

A record is an entry in the database.

An occurrence is a record displayed in a session. An occurrence can be viewed as a slot that may contain a record.

A session can show several occurrences, in other words show several records. Not all occurrences are necessarily filled, even if more records exist in the database. Which records are displayed is affected by many factors, including sort order, filter, and user actions such as scrolling.

BFlow operates by default on occurrence one (1).

Note: A record that is displayed in occurrence five, can be moved up by the user, through scrolling, and end up in the first occurrence. Its position may also be affected by insert or delete actions on the table.

We recommend that you use the default occurrence number. The Find action returns the requested record in this first occurrence, if it exists.

Good practices and considerations

5

BFlow is a powerful tool and offers a lot of freedom.

To keep the test scripts and test cases maintainable, follow these guidelines:

1. Data management.

   Many sessions strongly depend on the data present in the different companies.

   Think about what master data is required. Other data is probably more volatile, and created and deleted during the execution of the tests.

   Cleanup data after tests have run.

2. Test scripts perform small tasks.

   Keep reusability in mind! For example, create purchase order, add purchase order line.

   The tasks can be validated and a result can be returned to the caller.

   The test script should operate standalone and its actions are potentially driven by data provided by the input arguments.

3. Test cases add complexity and are used to construct the business flows.

   Good practice is to have a single test case for each test scenario. This reduces impact when new test scenarios are implemented, or when existing scenarios must be adapted.

   Test cases cannot interact with sessions. Test cases are used to chain the test script or test cases together and form a business flow. Data exchange between the scripts is achieved by using the arguments of the test scripts and test cases that can have a different scope: In, In/Out, or Out. This allows a test case to feed the result value of one script as input to another script. Also the test case can validate these intermediate values and take appropriate actions if required.

4. Prefer the use of variables over fixed values.

   Do not 'hide' fixed values inside test scripts or test cases. These are notoriously hard to find!

5. Check results against predicted values whenever possible, that is, use ExpectedValue.
6. Check return values of scripts before proceeding with the next step.

   A test scenario that fails at some point, but is allowed to continue, can cause data corruption and may make cleanup unnecessary complex. If something unexpected happens, terminate the script quickly and display a proper error message to help analysis of the problem.

7. Use EasyFilter to locate a record that has time/date in the index field.

Development

6

You can develop test scripts and test cases in LN Studio, the integrated development environment of LN.

For details, see these guides:

- *Infor LN Studio Administration Guide*
- *Infor LN Studio Application Development Guide*

Creating and importing BFlow tests

To create or import a BFlow test script or test case in LN Studio:

1 Select **File > New > Infor LN Component** .
2 In the **Component Type** field, select **Additional File**.
3 Click **Next**.
4 Specify the name and description of the additional file.
   - To create a test script, specify the file name in this format:
     `[package][module][filename].bflow`
   - To create a test case, specify the file name in this format:
     `[package][module][filename].bfcase`
   - To import an existing test, click **Browse** and select the desired file. Alternatively, specify the file name manually.
5 Click **Finish**.

You can now edit the test in the BFlow Editor.

Recording BFlow tests

Alternatively, you can create a BFlow test script by recording user actions in the user interface.

This is a fast and effective way to create an initial version of the test script, which can be developed further to a reusable building block.

To record a test:

1. Start LN UI.
2. Start the BFlow Test Script Recorder (tlrgt1290m000) session.
3. Specify a code and a description for an additional file.
4. Click **Start Recording**.
5. Perform some user actions.
6. In the BFlow Test Script Recorder (tlrgt1290m000) session, click **Stop Recording**. An additional file is created.

You can now open and edit the additional file in LN Studio.

Note: The recorder inserts the values specified by the user in the script. Although useful to learn how a specific session operates, it is hard to create a reusable, more generic, test script from such a recording.
The values must be replaced by variables with scope **In**. Probably some input validation is required.

BFlow Editor

Introduction

Use this editor to edit BFlow test scripts and test cases.

Before you can edit a BFlow test, you must create or import the test in LN Studio. See the *Infor LN BFlow Development Guide* and the LN Studio online help.

General

This tab shows general information and contains a comment section.

General Information

Name
The name of the test.

The name has this format:

`[package][module][filename].bflow` in case of test script

`[package][module][filename].bfcase` in case of test case

Description
The description of the test.

Domain
The domain of the test. Only applicable for test cases.

Owner
The owner of the test. Only applicable for test cases.

Company
The company to start the test in. Only applicable for test cases.

Logistic Company
The logistic company to start the test in. Only applicable for test cases.

Financial Company
The financial company to start the test in. Only applicable for test cases.

Comment

Comment
Shows the comment section of the test.

Variables

Use this tab to add, modify, or delete variables.

Variables

Name
The name of the variable.

Type
The variable type such as Long or String.

Domain
The LN domain name such as tcyesno.

Scope
The scope of the variable. For example, In or Internal.

Initialized
Indicates whether the variable is initialized.

Initial Value
The initial value of the variable.

Description
The description of the variable.

Buttons

Add
Adds a variable in the grid.

Up
Moves the selected variable one row up in the grid.

Down
Moves the selected variable one row down in the grid.

Remove
Removes the selected variable from the grid.

Actions

Use this tab to add, modify, or delete test actions.

The tab consists of these sections:

- **Overview**

  This section shows the test actions. If you select an action in the tree, the corresponding action details are displayed in the **Details** section.

  The tree supports cut/copy/paste and drag/drop functionality.

  You can expand or collapse all nodes of the tree by clicking the icons above the tree.

  Click the show palette link above the tree to open the BFlow Palette view. This view shows all available actions. To add an action to the test, drag and drop an action from the palette, or use the shortcut menu of the tree.

- **Details**

  This section shows the details of the action selected in the **Overview** section.

  For details about the different action types, see the *Infor LN BFlow Development Guide* and the LN Studio online help.

XML

Use this tab to edit the XML source code of the test directly. The XML must be valid before you can save the test or switch to other tabs.

TDE Documentation

Use this tab to view an XML tree with the technical information on the test's Technical Data Entity.

This optional tab is displayed depending on the multipage editor preferences. See the *Infor LN Studio Application Development Guide* and the LN Studio online help.

BFlow Palette view

This view shows the actions you can use in BFlow tests.

Use the buttons in the title bar to change the way the actions are displayed. The actions can be displayed in these ways:

- As a list, in alphabetical order
- Grouped by category

To add an action to a Bflow test, drag the action to the tree in the **Actions** tab in the BFlow Editor. Alternatively, use the shortcut menu of the tree.

Generating Set/Get Fields for a session

In a BFlow test, you can generate Set Field and Get Field actions for a session. You can do this for these actions:

- Select Session
- Start Session
- Select Synchronized Dialog
- Select Satellite

Generating Get Field actions

To generate Get Field commands for an action:

1. Right-click the action, and select **Generate Actions > Generate Get Fields**.
   The Add Get Fields actions dialog box is displayed.
2. Select the fields for which you want to generate Get Field actions.
3. Specify this information:

   Generate Step for selected actions
   If this check box is selected, a step is created under the current action. All generated actions are added to this step.
   Default: Selected

Generate Variables
If this check box is selected, output variables are created for the selected fields. If an input variable with the same name already exists, this variable changes to an input/output variable.

Default: Selected

Initially hidden
If this check box is selected, if the form field was initially hidden, this field is also displayed in the tree to select.

Default: Cleared.

Prefix
If you specify a prefix, the names of any generated variables start with this prefix. A prefix is required if you generate variables for different sessions with the same field names.

4 Click **OK**. For all fields selected, an action is generated. Depending on the information you specified, variables may also be generated for the selected fields.

The generated actions are displayed in the **Actions** tab in the BFlow Editor.

If you selected **Generate Variables**, the generated variables are displayed in the **General** tab in the BFlow Editor.

Generating Set Field actions

To generate Set Field commands for an action:

1 Right-click the action, and select **Generate Actions > Generate Set Fields**.

The Add Set Fields actions dialog box is displayed. This dialog box shows all editable form fields in a tree that represents groups per branch.

2 Select the fields for which you want to generate Set Field actions.

3 Specify this information:

Generate Step for selected actions
If this check box is selected, a step is created under the current action. All generated actions are added to this step.

Default: Selected.

Generate Variables
If this check box is selected, input variables are created for the selected fields. If an output variable with the same name already exists, this variable changes to an input/output variable.

Default: Selected.

Initially hidden
If this check box is selected, if the form field was initially hidden, this field is also displayed in the tree to select.

Default: Cleared.

Generate If Variables
If this check box is selected, an If Variable action is created.

Default: Selected.

Generate Info Fields
If this check box is selected, an Info Field action is created.

Default: Selected.

Single-Occurrence
If this check box is selected, fields are displayed for single-occurrence sessions.

Default: Selected.

Multi -Occurrence
If this check box is selected, fields are displayed for multi-occurrence sessions.

Default: Selected.

Prefix
If you specify a prefix, the names of any generated variables start with this prefix. A prefix is required if you generate variables for different sessions with the same field names.

4 Click **OK**. For all fields selected, an action is generated. Depending on the information you specified, variables may also be generated for the selected fields.

The generated actions are displayed in the **Actions** tab in the BFlow Editor.

If you selected **Generate Variables**, the generated variables are displayed in the **General** tab in the BFlow Editor.

Execution and logging

7

You can run the BFlow scripts, that is, test script and test case, from LN Studio or from an LN session. You can run a test set only from the scheduler, or manually using an LN session.

You can manually run BFlow scripts in LN Studio and in LN sessions. BFlow tests can also be registered in the Test Scheduler for later or periodic execution. The logging and its output location depend on the chosen execution method.

Execution in LN Studio

In LN Studio, you can run and debug test scripts and test cases.

The generated log is displayed in the LN Studio console.

- To run a BFlow test, right-click the test in the Activity Explorer and select **Run As > BFlow Test**.
- To debug a BFlow test:
  a Set the breakpoints.
  b Right-click the test in the Activity Explorer and select **Debug As > BFlow Test**.
  The execution is paused each time a breakpoint is hit. You can then inspect the variables, continue the execution step-by-step, or resume the execution.

  For more information about setting breakpoints and debugging, see the *Infor LN Studio Application Development Guide*.

Execution in LN sessions

To start BFlow scripts outside LN Studio, use the Test Case Logging (tlrgt1170m000) session. The output is stored in tables and can be viewed using the Test Case Logging (tlrgt1170m000) session.

To run a test set manually, use the Test Set (tlrgt1630m000) session.

Creating test sets

To create a test set:

1. Start the Test Sets (tlrgt1130m000) session.
2. Specify this information:
   - A unique name to identify the test set
   - The company in which the test set must run
   - Indication for local or remote execution

   For details, see the session help.

Execution by the Test Scheduler

To create or modify a schedule to run a test set at a particular moment or periodically, use the Test Set (tlrgt1630m000) session. To access the generated log, also use this session.

In addition, to access the log of each run of a specific test set, use the Test Set Log (tlrgt1160m000) session.

Test log

To view the log of runs of a test case, use the Test Case Logging (tlrgt1170m000) session.

To view the log of runs of a test set, use the Test Set Log (tlrgt1160m000) session, or indirectly use the Test Sets (tlrgt1130m000) session.

The execution results are logged on test set, case, and step level.

Concepts

8

This section describes concepts employed by BFlow.

Context

A test runs in a context. This context can be considered an environment that isolates information related to one test from information related to another test.

Among other things, the context contains information on these items:

- Variables
- Current company
- Current role
- Started session(s)
- Logger

When a test case invokes another test case or a test script, a context is created for that new script. On initialization of that context, the variables of the script are initialized with the arguments passed by the calling test case, if any.

If the script must run in another company, this company is also stored in the context. Otherwise the company of the caller is inherited.

In preparation of the execution of the script, a logger is created for the script. All log output is sent to this logger, so afterwards the log can be precisely matched with the script that generated it.

After the script has completed its execution, the arguments that have the **In** and **InOut** scope are updated with the corresponding variables. This way information can be passed from the invoked script to the caller. The execution result, such as failed or succeeded, is passed to the caller. Finally, the context is removed and the caller continues its execution with the next instruction, if any.

Companies

The "current company" concept is employed. In every context a current company is maintained.

If a company is not explicitly specified, the company of the caller is used. For the top level test case or test script, BFlow falls back to the company of the user. All sessions started in a context use the current company of that context.

You can specify the initial current company in the test; this applies only to a test case.

Within the context of a test, you can change the current company through the `ChangeCompany` action.

When a script finalizes, the current company is restored to the company of the parent, if any.

Aggregation

A test case can invoke another test case or a test script. A test script cannot invoke any script.

This table shows how the current company of the invoked test is determined:

Invocation type	Description
Invocation of test script	The current company is inherited from the caller, that is, runs in the same company.
Invocation of test case	If no company is provided for the invoked test, the current company is inherited from the caller, that is, runs in the same company. If a company is provided for the invoked test, the current company is set to that provided company.

Note:

- Within the context of a test, you can change the current company through the `ChangeCompany` action.
- When the invoked test finishes, and control returns to the caller, any changes to the current company are not propagated to the caller. The current company of the caller remains unchanged.

Standalone execution

You can run a BFlow script standalone. In this specialized scenario, the current company equals the company as specified in the user data.

Test result

The following table shows the, pre-defined, results a test can return. These statuses relate directly to a Log status.

Test result	Log status	Description
inactive	inactive	<internal>
skipped	skipped	Condition(s) to execute not true
started	started	Not yet finished
canceled	canceled	External request to stop
succeeded	succeeded	Normal termination
failed	failed	Result unexpected, functionally
aborted	aborted	Critical error occurred

Note: A test may also return a user-defined result. This always results in a "failed" log status!

A test that invokes another test can retrieve the test result of that test and use it to affect its own execution; for example, decide to continue or exit.

Normal execution

When a test runs normally (that is, no errors occur) the script result is by default "succeeded".

It is not necessary to end a script with the `Exit` action.

Exception situation

When an exception occurs, the script result depends on the type of exception.

This table shows the exception types:

Failed	InstructionException	Instruction is invalid.
	ContextException	For example, session not found.
	ScriptResultException	Script result is not assigned to variable.
	WebUiMockException	Interface failure.
	ExitTestException	Caused by ExitTest action.
	ExpressionException	
Aborted	InvocationException	Incorrect script argument.

ConvertException	Type conversion error.
FileIOException	Failed to load script.
AbortTestException	Caused by AbortTest action.
DebugTestException	Debug interface signaled error

Action-related

This table shows actions that can affect the script result:

Action	Description
Exit Test	The script can affect its result by an explicit call to the Exit Test action and pass the desired return value. User-defined values are allowed, but cause the "failed" Log status; see above. **Note:** Control is returned to the caller, if any.
Abort Test	The Abort Test action causes an "aborted" script result. **Note:** The entire test is aborted; any parents are also terminated.
Check Field/Check Variable	These actions have a "stop if check fails" option to terminate the script if the checked condition is not met. This causes the "failed" script result. **Note:** Control is returned to the caller, if any.

Actions

9

This section describes the available actions, and their properties in the BFlow Editor.

You can perform these actions:

- "While" on page 60
- "Zoom Field" on page 61

Abort Test

Aborts the test execution. The test result is "aborted".

Note: The entire test is aborted; any parents are also terminated.

Add to Job

Adds the current session to a job.

Answer

Sets the answer to the occurring question or message.

Fields in the editor

Question
The question code the answer must be applied to (optional).

Message
The message code the answer must be applied to (optional).

Variable / Value
The variable or value that contains the answer to the question or message. The answer must be expressed as the enum constant or its numeric equivalent. For example, **tcyesno.yes** or **1**.

Change Company

Changes the current company.

See "Companies" on page 30.

Fields in the editor

Field
The field that contains the company number.

Occ
The occurrence number of the field (optional).

Variable
The variable that contains the company number.

Value
The company number.

Change Financial Company

Changes the current financial company.

See "Companies" on page 30.

Fields in the editor

Field
The field that contains the company number.

Occ
The occurrence number of the field (optional).

Variable
The variable that contains the company number.

Value
The company number.

Change Logistic Company

Changes the current logistic company.

See "Companies" on page 30.

Fields in the editor

Field
The field that contains the company number.

Occ
The occurrence number of the field (optional).

Variable
The variable that contains the company number.

Value
The company number.

Change Role

Changes the current user's role.

Fields in the editor

Field
The field that contains the user role.

Occ
The occurrence number of the field (optional).

Variable
The variable that contains the user role.

Value
The user role.

If a role with empty string is specified, the current role of the user is used. This can be convenient to revert to the original role after changing the role in a script.

Note: The role is affected only for the duration of the script and is inherited by invoked test cases or test scripts. When control returns to the parent, if any, the role of the parent is restored.

Check Field

Checks a field value against a condition.

Optionally, the script can be terminated when the condition is not met.

Fields in the editor

Name
The field to check.

Occ
The occurrence number of the field (optional).

Operator
The operator used.

Possible values: **=(eq)**, **<>(ne)**, **>(gt)**, **>=(ge)**, **<(lt)** or **<=(le)**

Field (+Occ) / Variable / Value
The field, variable, or value to check the field against.

Error Message
The message that is written to the log if the check evaluates to false.

Success Message
The message that is written to the log if the check evaluates to true.

Stop if check fails
If this check box is selected, the test execution stops if the check fails.

Check Variable

Checks a variable value against a condition.

Optionally, the script can be terminated when the condition is not met.

Fields in the editor

Name
The variable to check.

Operator
The operator used.

Possible values: **=(eq)**, **<>(ne)**, **>(gt)**, **>=(ge)**, **<(lt)** or **<=(le)**

Field (+Occ) / Variable / Value
The field, variable, or value to check the variable against.

Error Message
The message that is written to the log if the check evaluates to false.

Success Message
The message that is written to the log if the check evaluates to true.

Stop if check fails
If this check box is selected, the test execution stops if the check fails.

Choice, Case, Default

Selectively execute action(s) based on the value of an expression.

Syntax:

```
choice expression
  case <value_1>
     action(s)
  end
  case <value_2>
     action(s)
  end
  default
     action(s)
  end
end
```

Fields in the editor

Field (+Occ) / Variable
The field or variable to check.

Close Session

Attempts to close the specified session.

The session can be identified by:

- **ID**, which is set by a previous action
- **Code**
- **Code** + **Parent ID**

If none of the fields is specified, Close Session attempts to close the current session.

It may cause InstructionException if the specified session is unknown.

Fields in the editor

ID
The ID of the session. The ID is set, for example, by a Start Session.

Code
The session code.

Parent ID
The ID of the parent session (optional).

Delete

Deletes the selected records.

Deselect All

Removes the selection.

Duplicate

Creates a new record with the data of the selected record.

Duplicate View

Duplicates the current view.

You can set the view fields by nested `SetField` actions.

Easy Filter

Applies filters on the current dataset using filter row under the table header.

The result is stored in a variable:

- 0: No records found
- 1: A record found (conditions are met).

See "Records and occurrences" on page 15.

Fields in the editor

Column
The column to filter.

Operator
The operator to be used:

- **= (eq)**
- **<> (ne)**
- **> (gt)**
- **>= (ge)**
- **< (lt)**
- **<= (le)**
- **starts with (startsWith)**
- **does not start with (doesNotStartWith)**
- **contains (contains)**
- **does not contain (doesNotContain)**
- **ends with (endsWith)**
- **does not end with (doesNotEndWith)**
- **empty (empty)**
- **not empty (notEmpty)**

Variable/Value
The filter value or the variable that stores the filter value.

Store result in
The variable to store the filter result.

Clear filter
Select this check box to clear the previous filter expression before applying the new one.

Execute Function

Executes a DLL function.

Fields in the editor

Library
The library containing the function.

Function
The function to call.

Arguments

Type
The data type of the function argument.

Scope
The scope of the argument.

Variable/Value
The argument value or variable that stores the argument value.

Return value

Return value
Select this check box if the function has a return value.

Type
The data type of the return value.

Variable
The variable to store the return value.

Execute Test

Executes a BFlow or BUnit test.

Fields in the editor

Name
The name of the additional file in case of BFlow, or the name of the library in case of BUnit.

Type
The type of the test, BFlow or BUnit.

Company
The company to run the test in (optional).

Logistic Company
The logistic company to run the test in (optional).

Financial Company
The financial company to run the test in (optional).

Store result in
The variable to store the execution result.

Arguments

Argument
The name of the argument.

Variable/Value
The variable/value used to set the argument.

Expected Value
The expected value of an argument of scope Out or InOut.

Condition

Left operand
The left operand of the comparison.

Operator
The operator of the comparison.

Right operand
The right operand of the comparison.

Expected Errors
One of these errors is expected when executing the test.

Exit Test

Stops test execution and returns to the parent test with an exit value.

Fields in the editor

Variable/Value
The exit value or the variable that stores the exit value.

Expression

Executes an arithmetic or logical expression.

Takes exactly two operands.

For summarization of more than two operands, see the Sum action.

Syntax:

```
Result = Left_Operand operator Right_Operand
```

This table shows the arithmetic operators:

Arithmetic operator	Operation
+	A + B
-	A - B
*	A * B
/	A / B

This table shows the logical operators:

Logical operator	Operation
and	A and B
or	A or B

This table shows the data types of the operation results for divisions:

A	B	Operation result (A / B)
long	long	long
long	double	double
double	long	double
double	double	double

Note: When the operation result is assigned to the result, a type conversion may occur and cause side effects.

This table shows how the operation result is, possibly, converted to match the expression result type:

Result	Operation	
long	long	unchanged
long	double	truncated
double	long	unchanged
double	double	unchanged

Examples:

```
left value = operation (=right value)
```

	Right value	Left value	
Operation	Operation result	As long	As double
3 / 2	1	1	1.5
3.0 / 2	1.5	1	1.5
6.3 / 2.1	3.0	3	3.0

Fields in the editor

Result
The field/variable to store the expression result.

Left operand
The left operand of the expression.

Operator
The operator of the expression: **+**, **-**, **/**, *****, **and**, **or**.

Right operand
The right operand of the expression.

Validations

- If **Field** is selected, the specified field must exist in the current session.
- If **Variable** is selected, the specified variable must be declared.

Find

Finds the record in the current session that corresponds with the specified field values. The field values are set by nested `SetField` actions.

Possible result values:

- 0: No record found.
- 1: Requested record found.
- 2: A record was found, but it is not the requested one.

Hint:

To check whether a correct record is found, use: `if find.result = (eq) 1`

To check whether a correct record is not found, use: `if find.result <> (ne) 1`

See "Records and occurrences" on page 15.

Fields in the editor

Store result in
The variable to store the result.

Clear fields
Select this check box to clear the fields of the find dialog.

First Record(s)

Shows the first records in a view.

First View

Lists the records of the first view.

The `Find` action verifies the result. If a record was found, the fields that were set by the `SetField` actions are checked against the corresponding fields of the record. If these all match, the find returns one (1) to indicate that the requested record was found.

If at least one of these fields does not match, the action returns two (2) to indicate that the requested record was not found.

If no record was found at all, the action returns zero (0).

Note: If the `Clear fields` option is used, ensure to specify all the required fields. If the requested record was not found, detailed information of the current record is sent to the console.

To use the exact match option of the Find dialog box, use `SetField "__exact.match__"`, with value "1".

For Each

Iterates over all records, for the currently selected session, and executes the actions for each record.

Syntax:

```
foreach
   action(s)
end
```

For each iteration, the next record becomes the current record.

Note: Do not invoke foreach as an action within a foreach action for the same session. This causes an indefinite loop!

Form Command

Runs the specified form command in the current session.

Fields in the editor

Name
The name of the form command (session code or function name).

Get Company

Returns the current company number.

Fields in the editor

Store result in
The variable to store the result.

Get Financial Company

Returns the current financial company number.

Fields in the editor

Store result in
The variable to store the result.

Get Logistic Company

Returns the current logistic company number.

Fields in the editor

Store result in
The variable to store the result.

Get Command State

Reads the enabled and checked state of a standard command or form command and stores it in a variable of type String.

These are possible result values for the enabled state:

- "enabled"
- "disabled"
- "invisible"

These are possible result values for the checked state:

- "true"
- "false"

These are possible command names:

- "CloseSession"
- "SaveAndCloseSession"
- "FirstRecords"
- "PrevRecords"
- "NextRecords"
- "LastRecords"
- "NewView"
- "FirstView"
- "PrevView"
- "NextView"
- "LastView"
- "Delete"
- "Find"
- "Save"
- "Print"
- "New"
- "Duplicate"
- "RevertToSaved"
- "SelectAll"
- "DeselectAll"
- "Open"
- "OpenReadonly"
- "Refresh"
- <form command name>

Fields in the editor

Name
The command to check.

Store enabled state in
The variable to store the enabled state.

Store checked state in
The variable to store the checked state.

Get Field State

Reads the state of a field and stores it in a variable of type string.

These are possible results:

- "enabled"
- "disabled"
- "readonly"
- "invisible"

Fields in the editor

Name
The field to check.

Occ
Occurrence on the screen (optional).

Variable
The variable to store the result in.

Get Message

Reads the last question or message and stores it in a variable.

The variable contains the code of the question or message if available; otherwise the variable contains the message text.

If there is no question or message waiting, the variable is empty.

Fields in the editor

Variable
The variable to store the question or message code if available, or the message text.

Get Role

Returns the current user’s role.

Fields in the editor

Store result in
The variable to store the result.

If

Executes actions if the specified condition is true.

Fields in the editor

Left operand
The left operand of the comparison.

Operator
The operator of the comparison.

Right operand
The right operand of the comparison.

If Field

Checks the field's value and, if the check evaluates to true, executes the sub-actions.

Fields in the editor

Name
The field to check.

Occ
Occurrence on the screen (optional).

Operator
The operator used.

Possible values: **=(eq)**, **<>(ne)**, **>(gt)**, **>=(ge)**, **<(lt)** or **<=(le)**

Field (+Occ) / Variable / Value
The field, variable, or value to check the field against.

If Variable

Checks the variable's value and, if the check evaluates to true, executes the sub-actions.

Fields in the editor

Name
The variable to check.

Operator
The operator used.

Possible values: **=(eq)**, **<>(ne)**, **>(gt)**, **>=(ge)**, **<(lt)** or **<=(le)**

Field (+Occ) / Variable / Value
The field, variable, or value to check the variable against.

Info

Prints log information.

Fields in the editor

Text
The text to print.

Info Field

Prints the field value to the log.

Fields in the editor

Name
The field to print.

Occ
Occurrence on the screen (optional).

Info Variable

Prints the variable value to the log.

Fields in the editor

Name
The variable to print.

Last Record(s)

Shows the last records in a view.

Last View

Lists the records of the last view.

Mark

Marks the given record in the current session.

Fields in the editor

Occ
Occurrence on the screen (optional).

New

Creates a new record.

New View

Enables you to create or find a view of records that all have the same value for one or more specific fields. The field values are set by nested `SetField` actions.

Next Record(s)

Shows the next records in a view.

Next View

Lists the records of the next view.

Open

Opens the details session of the current record.

Fields in the editor

Occ
Occurrence on the screen (optional).

Open (read-only)

Opens the details session of the current record in read-only mode.

Previous Record(s)

Shows the previous records in a view.

Previous View

Lists the records of the previous view.

Print

Starts a print session.

Reconnect

Reconnects to a target environment.

By default the BFlow runtime and target environment run on the same host in different bshells using the same user and package combination.

Through the Reconnect action the BFlow runtime can connect to a target environment on another host, using a different user and a different package combination.

Note:

- After the script that used a reconnect action finalized, the connection with the target environment is closed. In case of a parent test case the connection with the target environment of that test case is created again.
- After using a reconnect action, the original connection is lost. Any running sessions in that bshell are closed, that is, the bshell is shut down.

Fields in the editor

As other user
Use a different user to log on.

OS Username
OS user to be used for logon.

BSE Username
BSE user to be used in the target environment.

Password
Password of the OS User.

Note: In LN Studio the password can be specified normally, after which it is automatically encoded.

To other host
Use a different host to log on.

Hostname
Name of the host to connect to.

Port
Port.

512.

BSE
The BSE to use.

Bshell
The Bshell to use.

Environment
Environment variables to pass to the target environment.

For example, BSE_COMPNR, PACKAGE_COMB.

Refresh

Shows the current session data. For example, you can use this action to show record details that you have changed in a different session.

Revert to Saved

Cancels the changes that you made after you opened or saved the record.

Rotate Currency

Shows amounts in another currency.

Save

Saves the changes.

Possible result values are:

- 0: Save succeeded
- 1: Save failed
- 2: No save required, that is, no records were changed

  Note: A session that uses `enable.save.in.occ.change` can also cause this.

Fields in the editor

Store result in
The variable to store the save result.

Save and Close Session

Saves the changes and closes the session.

Fields in the editor

ID
The ID of the session. The ID is set, for example, by a Start Session.

Code
The session code.

Parent ID
The ID of the parent session.

Select All

Selects all records in the current view.

Select Form Tab

Selects a form tab in case of multiple tabs.

Fields in the editor

Index
The tab index (1-based).

Select Satellite

Makes a satellite of the current session active.

To select a satellite, specify the session code or the index of the satellite.

You can assign a new ID to the selected satellite.

Fields in the editor

Code
The satellite session code.

Index
The tab index of the satellite (1-based).

ID
The ID of the selected session (optional).

Select Session

Makes an open session active.

The session can be identified by:

- ID, which is set by a previous action
- Code
- Code + Parent ID

Fields in the editor

ID
The ID of the session. The ID is set, for example, by a Start Session.

Code
The session code.

Parent ID
The ID of the parent session.

Select Synchronized Dialog

Makes an open synchronized dialog active.

To select the dialog, specify the code or ID of the parent session.

You can assign a new ID to the selected synchronized dialog.

Fields in the editor

Parent code
The code of the parent session.

Parent ID
The ID of the parent session.

ID
The ID of the selected session (optional).

Set Field

Sets the value of a particular field in the current session.

Fields in the editor

Name
The field to set.

Occ
Occurrence on the screen (optional).

Field (+Occ) / Variable / Value
The field, variable, or value that will be used to set the field.

Set Variable

Sets the value of a particular variable.

Fields in the editor

Name
The variable to set.

Field (+Occ) / Variable / Value
The field, variable, or value that will be used to set the variable.

Sort by

Changes the sorting order of the current session.

Fields in the editor

Index
The new sort order.

Start Session

Starts a new session.

You can assign a new ID to the started sessions.

Fields in the editor

Code
The session code.

ID
The ID of the session (optional).

Step

Logical group of test actions.

Fields in the editor

Info
Step info to be printed to the log.

Comment
Additional comments.

Sum

Adds given operands.

Note: Only applies to data type long and double, which may be mixed.

Fields in the editor

Result
The field or variable to store the result.

Operands
The operands to be added to result.

The operands are converted to the data type of the result.

Side effects:

When converting to long, any fractional part is truncated. This table shows some examples:

Operand A	Operand B	Result
Long: 1	Long: 2	Long: 3
		Double: 3
Long: 1	Double: 1,5	Long: 2
		Double: 2,5
Double: 1,5	Double: 1,5	Long: 2
		Double: 3

Suspend

Suspends the test for the specified number of milliseconds.

Fields in the editor

Milliseconds
The number of milliseconds to wait.

While

Repeat action(s) while the specified condition is true.

Syntax

```
while condition
   action(s)
end
```

Fields in the editor

Left operand
The left operand of the comparison.

Operator
The operator of the comparison.

Right operand
The right operand of the comparison.

Zoom Field

Opens the zoom session of the specified field.

Fields in the editor

Name
The field on which the zoom action will be performed.

Occ
Occurrence on the screen (optional).