

GeneXus[™]
by **Globant**

Test Coverage

Benefits of Test Coverage

GeneXus[™]
by Globant



Test coverage helps to monitor the quality of testing, and assists testers to create tests that cover areas that are missing or not validated yet. With a more structured approach, an aim at 100% requirement coverage and effective testing methods, you will not compromise on quality.

The benefits of this feature are:

(Identify code areas uncovered)

Test coverage helps you unearth areas of a program that have not been covered by a set of test cases. It helps make your application more robust and error-free.

(Removes redundant test cases)

Test coverage is especially useful in identifying and removing test cases that don't make much sense in the current project. Your developers can report these cases to remove them and make the overall code lighter.

(Smoother testing cycles)

You can prevent defect leakage using Test coverage analysis. Test coverage also helps in Regression testing, test case prioritization, test suite augmentation and test suite minimization. All this leads to smoother yet efficient testing cycles.

(Removes defects at early stages)

You can identify gaps in requirements, test cases and defects at early stages of your product development life cycle. It saves you from a lot of headaches later.

Test Coverage in GeneXus

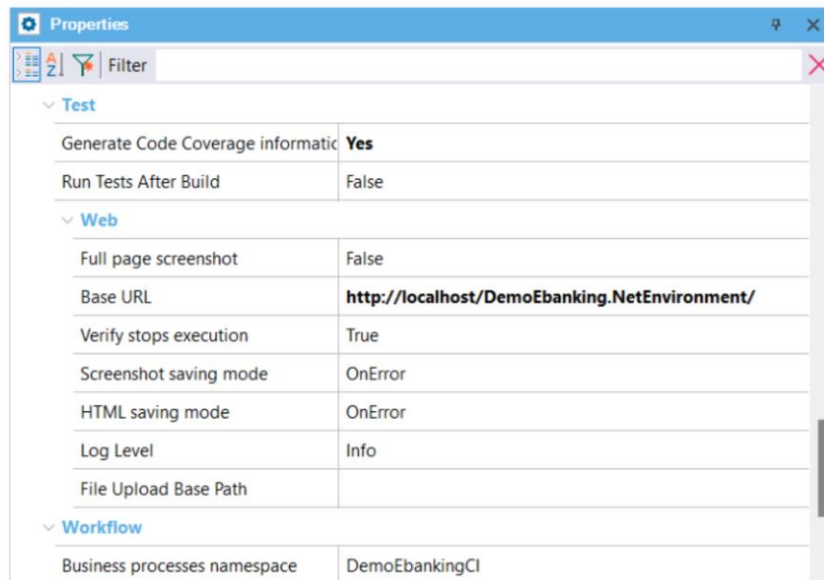
GeneXus[™]
by Globant



Test coverage is defined as a technique which determines whether our test cases are actually covering the application code and how much code is exercised when we run those test cases.

Agile methodologies indicate that 80% of the logic should be verified with unit tests, which are executed early in the development cycle and have a low maintenance cost as can be seen in the pyramid.

By default, in the GeneXus IDE, this feature comes disabled. To use it, the first thing to do is to enable Code Coverage for your environment.



To do this, set the ***Generate Code Coverage information*** property in the application environment to 'Yes' and **rebuild all the KB.**

The screenshot displays the GeneXus IDE interface during a test execution. The main window shows the test results for 'Tests.CheckBalanceForTransferTest' with a coverage of 96%. The test execution details panel on the right provides a summary of the test results, including the start and end times, elapsed time, and a table of expected vs. obtained results. The output window at the bottom shows the execution log, including the path to the coverage data file.

Unit test execution details

Tests: CheckBalanceForTransferTest Coverage: 96%
 Start: Wednesday, September 21, 2022 11:50:34 PM
 Elapsed time: 355 ms

Expected	Obtained	Info
✓ false	false	1. ExpectedIsSuccess:
✓ true	true	2. ExpectedIsSuccess:
✓ false	false	3. ExpectedIsSuccess:

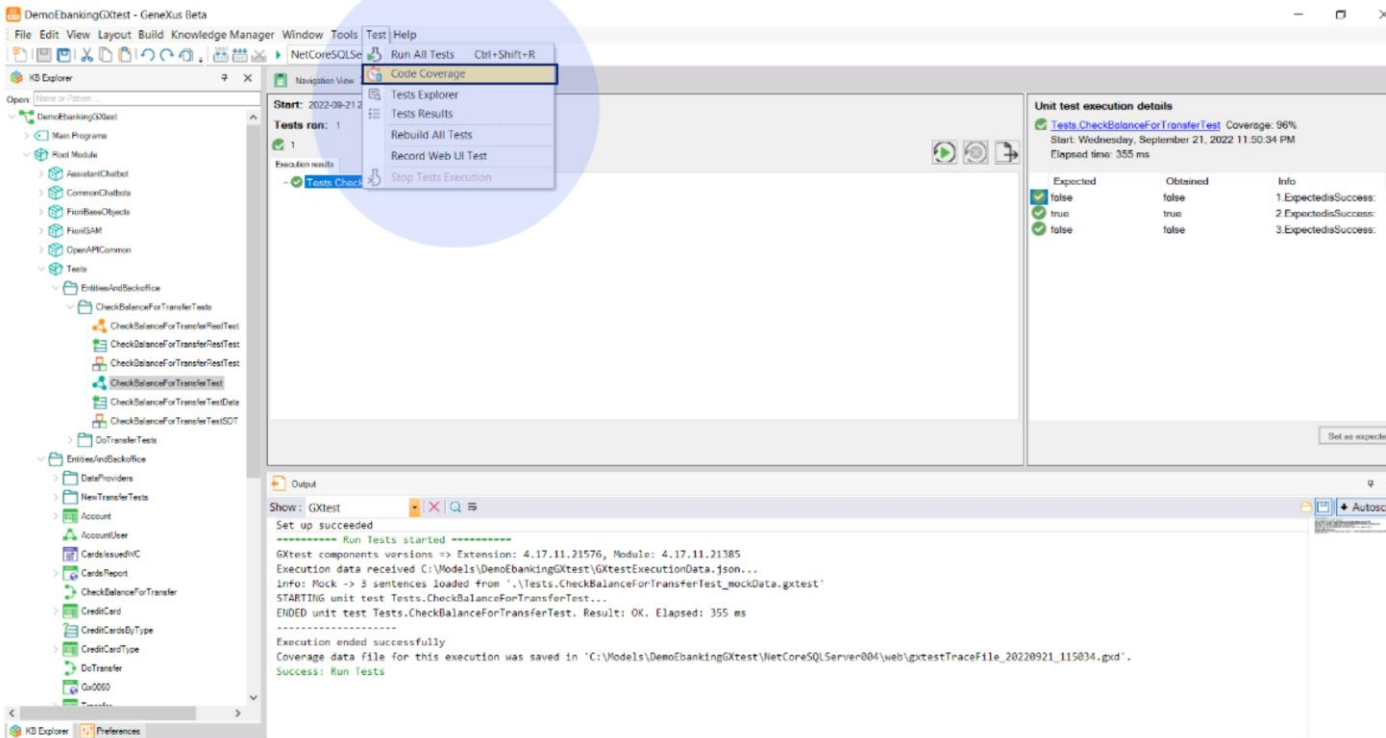
```

Show: GXtest
Set up succeeded
----- Run Tests started -----
GXtest: components versions => Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemoBankingGXtest\GXtestExecutionData.json...
Info: Mock -> 3 sentences loaded from ".\Tests.CheckBalanceForTransferTest_mockData.gxtest"
STARTING unit test Tests.CheckBalanceForTransferTest...
ENDED unit test Tests.CheckBalanceForTransferTest. Result: OK. Elapsed: 355 ms
-----
Execution ended successfully
Coverage data file for this execution was saved in 'C:\Models\DemoBankingGXtest\NetCoreSQLServer004\web\gxtestTraceFile_20220921_115034.gxd'.
Success: Run Tests
  
```

Once the Rebuild All operation finishes, you will be able to visualize the Coverage execution metric after each tests execution.

In the CheckBalanceForTransferUnitTest test execution you can visualize the Coverage percent in the Test Results panel. In the Output GeneXus panel, you will find the path of the Coverage data file for this execution. Coverage information is always related to a particular execution.

The displayed coverage value is the percentage of total lines of all the objects called by the tests aggregated in the execution performed. This means that if you execute 2 tests that each covers half of the lines of an object, the coverage displayed will be 100% as both tests aggregated cover all the lines of the tested object.



The screenshot shows the GeneXus IDE interface. The 'Test' menu is open, and 'Code Coverage' is highlighted. The 'Unit test execution details' panel shows the test 'Tests.CheckBalanceForTransferTest' with a coverage of 96%. The 'Output' window displays the test execution log.

Unit test execution details

Tests.CheckBalanceForTransferTest Coverage: 96%
Start: Wednesday, September 21, 2022 11:50:34 PM
Elapsed time: 355 ms

Expected	Obtained	Info
✓ false	false	1. ExpectedIsSuccess:
✓ true	true	2. ExpectedIsSuccess:
✓ false	false	3. ExpectedIsSuccess:

Output

```
Show: GXtest
Set up succeeded
----- Run Tests started -----
GXtest: components versions => Extension: 4.17.11.21576, Module: 4.17.11.21385
Execution data received C:\Models\DemoBankingGXtest\GXtestExecutionData.json...
Info: Mock -> 3 sentences loaded from ".\Tests.CheckBalanceForTransferTest_mockData.gxtest"
STARTING unit test Tests.CheckBalanceForTransferTest...
ENDED unit test Tests.CheckBalanceForTransferTest. Result: OK. Elapsed: 355 ms
-----
Execution ended successfully
Coverage data file for this execution was saved in "C:\Models\DemoBankingGXtest\NetCoreSQLServer004\web\gxtestTraceFile_20220921_115034.gxd".
Success: Run Tests
```

To open the coverage file and see the details of the Test Coverage go to *Test -> Code Coverage*

The screenshot shows the Code Coverage tool interface. At the top, there is a search bar with the text "Please, type the name of a file to import or select it:" and a file path: "C:\Models\DemoEbanking\Cl\CSharpModel\web\lgtest\TraceFile_20220905_113537.gxd". Below this is a table of objects with the following data:

Object	Hit Count	Time	Time with Children	Time (%)	Coverage (%)
CheckBalanceForTransfer	3	00:00:00.3465390	00:00:00.3465390	72.46	75
CheckBalanceForTransferUnitTest	1	00:00:00.1276944	00:00:00.4782606	26.70	100
CheckBalanceForTransferUnitTestData	1	00:00:00.0012256	00:00:00.0012256	00.26	100
LoadFioriContext	3	00:00:00.0028016	00:00:00.0028016	00.59	100

Below the table is a call graph showing a box for "CheckBalanceForTransfer Unit Test" with a hit count of 3, connected by an arrow to a box for "CheckBalanceForTransfer".

On the right side, there is a code editor showing the following code snippet with coverage data in the left margin:

Time	Hits	%	Code
00:00.0000796	0003	00.02	if &AccountNumber > 0 and &TransferAmount > 0
00:00.3462688	0003	99.92	&Account.Load(&AccountNumber)
00:00.0001906	0003	00.06	&isSuccess = &Account.AccountBalance >= &TransferAmount
			else
			&isSuccess = false
			endif

Load the Coverage data execution file selecting “...” button and then clicking *Load* you will see Coverage detail information.

On the left section, you can see every object involved in the execution and its respective execution information when you select them.

The **Hit Count** or **Hits** is the number of times the object has been executed

The **Time** is the Elapsed Time

The **Time with Children** is also the Elapsed Time, adding the elapsed time of the objects called by it

The **Time (%)** is the Total percentage of the elapsed time related to the rest

The **Coverage (%)** is the Percentage of coverage, that means the lines that were executed over total number of lines.

When an object from this list is selected, below a graph

indicating the call tree is shown. For example, in the current screenshot you can see that `CheckBalanceForTransfer` is called by `CheckBalanceForTransferUnitTest`.

On the right panel, the line codes with their respective trace information are shown.

The screenshot displays the Code Coverage tool interface. At the top, it prompts the user to import a file, with the path `C:\Models\DemoEbanking\CIC\SharpModel\web\gdastTraceFile_20220906_113537.gxd` entered. Below this is a table of coverage data:

Object	Hit Count	Time	Time with Children	Time (%)	Coverage (%)
CheckBalanceForTransfer	3	00:00:00.3465390	00:00:00.3465390	72.46	75
CheckBalanceForTransfer.UnitTest	1	00:00:00.1276944	00:00:00.4782606	26.70	100
CheckBalanceForTransfer.UnitTestData	1	00:00:00.0012256	00:00:00.0012256	0.26	100
LoadFioriContext	3	00:00:00.0028016	00:00:00.0028016	0.59	100

Below the table is a call graph showing a call from `CheckBalanceForTransfer.UnitTest` to `CheckBalanceForTransfer` with a hit count of 3.

The right-hand pane shows a code snippet with execution metrics:

```
00:00.0000796 0003 00.02 if &AccountNumber > 0 and &TransferAmount > 0
00:00.3462688 0003 99.92 &Account.Load(&AccountNumber)
00:00.0001906 0003 00.06 &isSuccess = &Account.AccountBalance >= &TransferAmount
else
    &isSuccess = false
endif
```

Note that it may have lines without information, that means that during the execution these lines weren't executed.

In this example, we can see that the else condition was not covered.

The screenshot displays the Code Coverage tool interface. The top section shows the file path: C:\Models\DemoEbanking\Clc\SharpModel\web\gdxest\TraceFile_20220905_113537.gdx. Below this is a table with the following data:

Object	Hit Count	Time	Time with Children	Time (%)	Coverage (%)
CheckBalanceForTransfer	3	00:00:00.3465390	00:00:00.3465390	72.46	75
CheckBalanceForTransfer.UnitTest	1	00:00:00.1276944	00:00:00.4782606	26.70	100
CheckBalanceForTransfer.UnitTestData	1	00:00:00.0012256	00:00:00.0012256	0.26	100
LoadFioriContext	3	00:00:00.0028016	00:00:00.0028016	0.59	100

Below the table is a call graph showing a call from 'CheckBalanceForTransfer.UnitTest' to 'CheckBalanceForTransfer' with a hit count of 3.

The right side of the interface shows a code editor with the following code snippet:

```

00:00.0000796 0003 00.02 if &AccountNumber > 0 and &TransferAmount > 0
00:00.3462688 0003 99.92 &Account.Load(&AccountNumber)
00:00.0001906 0003 00.06 &isSuccess = &Account.AccountBalance >= &TransferAmount
else
    &isSuccess = false
endif

```

So, let's understand how to use the Test coverage feature.

One of the main variables to monitor is the "Hit count" by which we visualize the number of times the object has been exercised. Also in the right detail, we can see the Hits for each code line of the procedure. With both data, we can gather that the procedure CheckBalanceForTransfer has 3 test cases, and the three cases go into the "if" condition. In consequence, it would be convenient to verify that redundant test cases are not being executed in the "if" condition.

The most important metric is the Coverage percentage column, we can see that 75% of the procedure CheckBalanceForTransfer was exercised. That means that not all code lines were tested, and we should add more test cases to get full test coverage. As we can see in the right detail, the else condition is not exercised by the test.

So, the developer should add a test case for this condition.

```

CheckBalanceForTransferUnitTestData  X
Source | Rules | Variables | Help | Documentation |
1 CheckBalanceForTransferUnitTestSDT
2 {
3     TestCaseId = '1'
4     AccountNumber = 5
5     TransferAmount = 200
6     ExpectedisSuccess = false
7     MsgisSuccess = ''
8 }
9
10 CheckBalanceForTransferUnitTestSDT
11 {
12     TestCaseId = '2'
13     AccountNumber = 6
14     TransferAmount = 200
15     ExpectedisSuccess = true
16     MsgisSuccess = ''
17 }
18
19 CheckBalanceForTransferUnitTestSDT
20 {
21     TestCaseId = '3'
22     AccountNumber = 8
23     TransferAmount = 200
24     ExpectedisSuccess = false
25     MsgisSuccess = ''
26 }
27
28 CheckBalanceForTransferUnitTestSDT
29 {
30     TestCaseId = '4'
31     AccountNumber = 6
32     TransferAmount = -200
33     ExpectedisSuccess = false
34     MsgisSuccess = ''
35 }

```

```

CheckBalanceForTransfer  X
Source | Layout | Rules | Conditions | Variables | Help | Documentation |
Subroutines
1
2 if &AccountNumber > 0 and &TransferAmount > 0
3     &Account.Load(&AccountNumber)
4     &isSuccess = &Account.AccountBalance >= &TransferAmount
5 else
6     &isSuccess = false
7 endif
8

```

So, we go to the test cases defined in the data provider and we will add the test cases for the else condition.

In this example, we will add one more test case with id 4 in which I defined a negative transfer amount to execute the else condition of the proc.

Code Coverage X

Please, type the name of a file to import or select it:

C:\Models\DemoEbanking\CI\CSHarpModel\web\gtest\TraceFile_20220906_105501.gxd

Object	Hit Count	Time	Time with Children	Time (%)	Coverage (%)	Time	Hits	%
CheckBalanceForTransfer	4	00:00:00.1476839	00:00:00.1476839	60.80	100			
CheckBalanceForTransferUnitTest	1	00:00:00.0892400	00:00:00.2429149	36.74	100	00:00.0000830	0004	00.06
CheckBalanceForTransferUnitTestData	1	00:00:00.0014646	00:00:00.0014646	00.60	100	00:00.1469984	0003	99.54
LoadFioriContext	4	00:00:00.0045264	00:00:00.0045264	01.96	100	00:00.0006024	0003	00.41

Call Graph:

```
graph LR; A[CheckBalanceForTransfer UnitTest] -- 4 --> B[CheckBalanceForTransfer]
```

```
if &AccountNumber > 0 and &TransferAmount > 0
  &Account.Load(&AccountNumber)
  &isSuccess = &Account.AccountBalance >= &TransferAmount
else
  &isSuccess = false
endif
```

After running the unit test again, we will see the impact in the Coverage metric of the procedure.

We can see in the Coverage (%) column that the procedure has the 100% because all code lines were exercised? in this unit test execution.

GeneXus[™]
by **Globant**

training.genexus.com
wiki.genexus.com