Genero Ghost Client User Guide
Copyrights and Trademarks................................................................................................. 26
Index........................................................................................................................................27
Genero Ghost Client User Guide

Manual organization at a glance.

<table>
<thead>
<tr>
<th>General</th>
<th>Reference</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What’s New</td>
<td>• GhostGenerator reference on page 18</td>
<td>• What you can test on page 8</td>
</tr>
<tr>
<td>• Overview</td>
<td>• Launcher reference on page 19</td>
<td>• Test Types on page 9</td>
</tr>
<tr>
<td>• Install and Configure</td>
<td>• Display version number on page 21</td>
<td>• The logging mechanism on page 22</td>
</tr>
<tr>
<td>• Framework</td>
<td>• Generate a scenario on page 16</td>
<td>• Write, compile, and run BDL tests on page 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Write, compile, and run Java tests on page 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Demos</th>
<th>Recording log files</th>
<th>Appendices</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Demos on page 23</td>
<td>• Record a scenario log on page 24</td>
<td>• Copyrights and Trademarks on page 26</td>
</tr>
<tr>
<td></td>
<td>• Generate a scenario from a guilog on page 25</td>
<td></td>
</tr>
</tbody>
</table>

GGC 1.10 new features

A summary of new features and changes in functionality introduced with Genero Ghost Client 1.10. The following changes and enhancements are relevant to this publication.

**Table 1: What’s new in GGC 1.10**

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Genero Ghost Generator now ships as part of the FGLGWS package.</td>
<td>See Install and configure for Genero Ghost Client on page 5.</td>
</tr>
<tr>
<td>The Genero Ghost Client has a dedicated User Guide. GGC content is no longer found in the Genero Application Server Administrator Guide.</td>
<td>N/A</td>
</tr>
<tr>
<td>The GhostGenerator class implements a new scenario generator.</td>
<td>See GhostGenerator reference on page 18.</td>
</tr>
<tr>
<td>You now generate test scenarios from guilogs.</td>
<td>See Generate a scenario from a guilog on page 25.</td>
</tr>
<tr>
<td>Test scenarios can no longer be generated from Genero Desktop Client logs, and generation of Java test scenarios are no longer possible.</td>
<td></td>
</tr>
<tr>
<td>You can specify the thread delay (-td) in seconds or milliseconds.</td>
<td>See Launcher reference on page 19.</td>
</tr>
</tbody>
</table>
Ghost Client and Testing Tools

The Genero Ghost Client (GGC) is a Java framework that allows you to test your applications. The GGC allows you to test the business logic and size the infrastructural needs of your applications. It acts as a ghost client because it does not render a graphical user interface. As a result, you can use it to test applications for different front-ends: Genero Browser Client (GBC), Genero Desktop Client (GDC), Genero Mobile for Android™ (GMA), or Genero Mobile for iOS (GMI).

When you want to do load or performance testing, you can specify it to run tests acting as one or more concurrent users from one machine.

For testing, you create different scenarios and user interactions on an application that the GGC can then run automatically against the application. The GGC can be used with a direct connection to the Dynamic Virtual Machine (DVM), or can use an application accessed via the Genero Application Server (GAS).

You can use the Genero Ghost Client to automate the following tests:

- Unit tests
- Load tests
- Performance tests

Test scenarios can be developed in two ways:

- Tests can be written by hand in either Genero Business Development Language (BDL) or Java.
- Test scenarios can be generated from the behavior described in a guilog file. The guilog file can be recorded using a direct connection or a GAS connection using any graphical client. The GhostGenerator then generates Genero BDL files from the log file data, allowing the replay of the scenarios by the Genero Ghost Client.

**Important:** Use Genero BDL to create GGC applications. Genero applications will work for most unit and load tests. Java should only be used to perform critical load testing.

One of the key features of testing with the Genero Ghost Client is that you do not need to modify the original application code to write test scenarios.

The Genero Ghost Client allows you to develop working test case scenarios that can be run as required to test the stability of your applications before release.

**Related concepts**
- The Genero Ghost Client framework on page 7
- The Genero Ghost Client is a Java framework for building application tests.
- Testing with Ghost Client on page 8
- Writing and running tests in Genero BDL and Java.

Install and configure for Genero Ghost Client

The Genero Ghost Client is embedded in the FGLGWS package. No additional installation is required, however configuration may be necessary.

**Note:** The Genero Ghost Client is also integrated with Genero Studio. See the *Genero Studio User Guide* for details. The GGC is embedded in the FGLGWS package in the `$FGLDIR/testing_utilities/ggc` directory.

To use Genero Ghost Client API, you need a Java Development Kit (JDK). Make sure that your JDK version is at least version 1.7 or greater.
Configure for Genero BDL tests

At a command prompt, navigate to FGLDIR and execute the script to set the environment.

- On Linux®/UNIX™/macOS™ execute envcomp.sh
- On Windows®, execute envcomp.bat

Set the PATH environment to include the directory containing the required JVM library on your system.

- On Linux®/UNIX™/macOS™ set the PATH to a directory containing libjvm.so:
  
  ```
  export PATH=/JDK_DIR/server:$PATH
  ```

- On Windows®:
  
  ```
  set PATH=%PATH%;C:\Program Files\Java\jdk1.8.0_20\jre\bin\server
  ```

For more information, see How to set up Java in the Genero Business Development Language User Guide.

Set your CLASSPATH environment variable to the absolute path to the ggc.jar and fgl.jar files. This allows you to compile and run test scenarios from any directory on your disk.

- On Linux®/UNIX™/macOS™:
  
  ```
  export CLASSPATH=/FGLDIR/testing_utilities/ggc/ggc.jar:/FGLDIR/lib/fgl.jar
  ```

- On Windows®:
  
  ```
  set CLASSPATH=C:\FGLDIR\testing_utilities\ggc\ggc.jar;C:\FGLDIR\lib\fgl.jar
  ```

To check your configuration, run the demo testing application. See Demos on page 23.

Configure for Java tests

Set your PATH environment variable to include the Java Development Kit bin directory.

- On Linux®/UNIX™/macOS™:
  
  ```
  export PATH=/JDK_DIR/bin:$PATH
  ```

- On Windows®:
  
  ```
  set PATH=C:\JDK_DIR\bin;%PATH%
  ```

Set your CLASSPATH environment variable to the absolute path to the ggc.jar and fgl.jar files. This allows you to compile and run test scenarios from any directory on your disk.

- On Linux®/UNIX™/macOS™:
  
  ```
  export CLASSPATH=/FGLDIR/testing_utilities/ggc/ggc.jar:/FGLDIR/lib/fgl.jar
  ```

- On Windows®:
  
  ```
  set CLASSPATH=C:\FGLDIR\testing_utilities\ggc\ggc.jar;C:\FGLDIR\lib\fgl.jar
  ```

Once PATH and CLASSPATH are set, compile and run your scenarios as shown in these examples:

- Compile: javac userWorkspace/path/to/generated/.*java

An alternative to setting the CLASSPATH environment variable is to specify the absolute path to the ggc.jar file using the java -classpath (-cp) option at runtime. Below are some examples.

- On Linux®/UNIX™/macOS™:
  
  ```
  Compile: javac -cp absolute/path/to/userWorkspace:/absolute/path/to/ggc.jar
  ```
  
  ```
  userWorkspace/path/to/generated/.*java
  ```
The Genero Ghost Client framework

The Genero Ghost Client framework is a Java framework for building application tests.

If you are developing your scenarios using Genero Business Development Language (BDL), the underlying Java framework is encapsulated and simplified.

If you are developing your tests in Java, knowing a little about its interfaces, classes, and methods outlined here will help you develop tests.

Java infrastructure overview

![Diagram of the Genero Ghost Client infrastructure]

**Figure 1: Ghost Client testing tool**

The Genero Ghost Client infrastructure consists of three components:

- **Launcher**

The **Launcher** is the Genero Ghost Client Java program which starts a testing session for an application.
The SessionManager is the interface that manages the runtime and the test Scenario.

The Scenario is the interface that contains the sequence of tests run during a given session.

The complete details of the packages and classes that make up the Genero Ghost Client can be found in the /doc directory of your GGC package. Please see the help file by launching the /doc/index.html file in your browser.

Developing tests with Ghost Client

When you write tests to be run by the Genero Ghost Client, you need to implement its two main interfaces: the SessionManager and the Scenario. These classes function together to implement the GGC tests. The main classes of the Genero Ghost Client are described in Table 2: Ghost Client interfaces and classes on page 8.

Table 2: Ghost Client interfaces and classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionManager</td>
<td>Instantiates and manages Scenario instances according to incoming VM connections and new runtime launches.</td>
</tr>
<tr>
<td>Scenario</td>
<td>A Scenario describes the action sequence played during the GhostRunner session to simulate user actions on a Genero application.</td>
</tr>
<tr>
<td>GhostRunner</td>
<td>The GhostRunner class is the entry point for test case scenarios. It contains methods for a set of possible actions that an end-user might do on the running application, e.g. set focus on a field, set a value in a field, etc.</td>
</tr>
<tr>
<td>Log</td>
<td>The Log class implements the logging mechanism used to display errors, warnings, and information as output from the tests.</td>
</tr>
</tbody>
</table>

For more information on developing tests with Genero Ghost Client see Testing with Ghost Client on page 8.

Related concepts

Demos on page 23

The Genero Ghost Client installation includes a sample test scenario for you to explore in the samples directory of your GGC package.

Testing with Ghost Client

Writing and running tests in Genero BDL and Java.

What you can test

Any interaction a user would normally do on an application can be automatically tested by the Genero Ghost Client.

For testing purposes it may be helpful to focus on building tests that fall into two categories: introspecting the application AUI tree or executing user action. Functions that perform actions can be tested once to see if they work as expected. You can then run them at any time during development to continually verify that changes to the application do not affect previously tested features.
**Introspecting the application AUI tree**

These are some examples of introspecting the AUI tree:

- List available menu actions
- List available fields
- Get value of a given field
- Get the field that has the focus

**Executing a user action**

These are some examples of executing a user action:

- Send an action
- Input a value in a field
- Scroll through a table
- Open a tree view node

---

**Test Types**

The Genero Ghost Client can assist with unit, load, and performance testing.

**Unit Tests**

*Unit testing* is a method of testing each feature of your application in isolation to make sure it works as expected.

A unit test should provide you with the anticipated responses to a given set of user input, showing that the feature is able to handle both correct and incorrect input. The following good practices are recommended when designing unit tests:

- Each 4gl application should have its own scenario.
- Make a complete list of the application features, from the smallest (for example, displaying the About... screen), to the biggest and / or the most important ones.
- For each feature in this list, write a single test that will test one (and only one) feature.
- If a primary feature implies a secondary feature (for example, to register a new customer, you need to fill out a form), the secondary feature's test should take place before the primary feature's test in the test sequence.

**Load Tests**

*Load testing* describes a testing strategy that simulates a specified number of users using your application at the same time and at normal human speed.

Load testing your application during development allows you to see how it will behave under similar conditions in a production environment. Load testing can help you identify server and network requirements based on the number of anticipated users: how many servers, how much memory is needed, and so on. Observations made under these test conditions can also show an application's weaknesses and allow you to fix them before release.

The GGC allows you to specify several options through the Launcher class for load testing:

- The thread_number option allows you to set the number of users to simulate. One thread represents one user.
- The thread_delay option allows you to set a delay between each user's connection. For example, in a real life situation where a thousand users could potentially launch an application, the likelihood of them all connecting at the exact same time is quite remote. The thread delay option helps you to recreate a more realistic simulation.

You can specify the number of threads (users) and the thread delay using the command line switches when launching tests. As shown in the example below, the GGC will launch the IntegrityTestsSessionManager with 3 users, allowing a 3 second delay between each launch.
Testing with Ghost Client


The Scenario for load testing needs to reflect how a real user would use an application. For example, the setDelay() method can be set to make the GhostRunner instance wait for a specified number of seconds between each action. This allows you to try to reproduce the human speed of interaction with an application, which is usually much slower than computer speed. Any interaction a user would normally do on an application can be played.

Performance Tests

Performance testing is similar to load testing as it also consists of simulating a significant number of users, however the aim of the test differs. Performance testing can determine when the system's performance will start to degrade. To performance test, you specify increased numbers of simulated users so as to observe at what number of users the request/response delay is affected.

Write, compile, and run BDL tests

The Genero Ghost Client has different methods for creating tests depending on whether you are using Genero BDL or Java. In this section the topics are about writing, compiling, and running BDL tests.

- Write a Genero BDL test on page 10
- How to compile BDL tests on page 12
- How to run BDL tests on page 13

Write a Genero BDL test

To write a test in Genero BDL, one file may contain the function that manages the session and the function or functions that run tests.

Note: In Genero Studio you can write tests using Genero BDL and compile them to Java using the Genero Ghost Client infrastructure through the Java Bridge. The Java Bridge allows you to import GGC classes. See the The Java interface section of the Genero Business Development Language User Guide.

1. Begin your program code by importing the Genero Ghost Client Java classes.

   Your test requires these classes:
   
   - The Session class is the entry point for Genero BDL tests. A session needs to be created to give the URL to start the main application.
   - The Application class is got from the Session. It uses methods that reference the interface form objects such as fields, actions, etc. Values can be retrieved and set.
   - The Log class implements the logging mechanism used to display errors, warnings, and information as output from the tests.

   IMPORT JAVA com.fourjs.ggc.fgl.Application
   IMPORT JAVA com.fourjs.ggc.fgl.Session
   IMPORT JAVA com.fourjs.ggc.util.Log
   IMPORT JAVA java.lang.String

2. Create a function that manages the test session.

   In our sample code you can see:
   
   - The Session.create(url) method takes the argument of the URL of the application to test.
   - The s.getNextApplication(10) method gets the session's next pending application to test. Timeout is set to ten seconds.
• The `launchDemo` function represents the test scenario. In our case we choose to pass arguments of the `app` class that represents the application we want to test. The test is targeting the "IA demo" application in the application tree of the `gwc-demo`.
• The `childApp` variable gets another instance of the application, which is run while the main application is still running, as shown in the example.
• The `app.enqueueAction("cancel")` method takes as argument an action name as defined in your 4GL dialog statement (`on action ...`). The `EnqueueAction` method uses the underlying `GhostRunner` to interact with the application but it uses an intermediate queue to free up the runner for the program using the application object to do something else while the runner is executing the requested operation. The `app.synchronize(10)` method waits ten seconds for this application interface queue to empty.
• The `getApplicationListener().getLogger()` method gets the logger for monitoring the application. If the application fails to run or an application is missing, etc., the `messageFromBdl` method is invoked to output the information to the standard output.

```/*
* Sample function to manage a test of the "IA Demo" application from the FGLGWS demo */
FUNCTION testSession(url)
DEFINE s Session
DEFINE app Application
DEFINE childApp Application
DEFINE childApp2 Application
...
DEFINE sLog Log
DEFINE aLog Log

LET msg = SFMT("Running DemoTests on %1 ...", url)

-- Run main application and make some basic introspection
-----------------
LET s = Session.create(url)
LET sLog = s.getSessionListener().getLogger()
CALL sLog.messageFromBdl(msg)

LET app = s.getNextApplication(10)
IF app IS NULL THEN
  CALL sLog.errorFromBdl("Main Runner not spawned")
  EXIT PROGRAM 1
END IF

  LET aLog = app.getApplicationListener().getLogger()
LET v = app.getApplicationName()
CALL sLog.messageFromBdl("Main application name:", v)
IF v != "demo" THEN
  CALL sLog.errorFromBdl("Incorrect application name. Expecting [demo]")
  EXIT PROGRAM 1
END IF
...

-- Running the IADemo application
-----------------------------------------

CALL launchDemo(app, "Compatibility/IADemo", "The IA demo")

LET childApp = s.getNextApplication(10)
IF childApp IS NULL THEN
  CALL sLog.errorFromBdl("Unable to get the expected child application [ia]")
  EXIT PROGRAM 1```
END IF
-- RUN WAITING a new IADemo instance from IADemo
--------------------------
LET v = childApp.getApplicationName()
CALL sLog.messageFromBdl("Child application name:", v)
IF v != "ia" THEN
    CALL sLog.errorFromBdl("Incorrect application name. Expecting [ia]")
    EXIT PROGRAM 1
END IF

CALL sLog.messageFromBdl("We successively made a run without waiting!")

CALL app.enqueueAction("cancel")
CALL app.synchronize(10)
...

IF app.isRunning() THEN
    CALL sLog.messageFromBdl("app is still running!")
ELSE
    CALL sLog.messageFromBdl("app successfully ended!")
END IF

IF s.getSessionListener().computeResult("DemoTests") THEN
    LET msg = SFMT("DemoTests successful on %1", url)
    CALL sLog.messageFromBdl(msg)
    LET success = TRUE
ELSE
    LET msg = SFMT("DemoTests not successful on %1", url)
    CALL sLog.messageFromBdl(msg)
    LET success = FALSE
END IF
END FUNCTION

Note:
The complete source code for the example can be found in the DemoTests.4gl file of the Genero Studio GGC_Sample.4pw project included in the samples/BDL directory.

Related concepts
Write, compile, and run BDL tests on page 10
The Genero Ghost Client has different methods for creating tests depending on whether you are using Genero BDL or Java. In this section the topics are about writing, compiling, and running BDL tests.

The logging mechanism on page 22
The Genero Ghost Client implements a logging mechanism to display errors, warnings, and information as output from the tests. Learn how to control the output, and use the logging mechanism efficiently to detect and solve application problems.

Demos on page 23
The Genero Ghost Client installation includes a sample test scenario for you to explore in the samples directory of your GGC package.

How to compile BDL tests
To compile your Genero BDL tests, you run the fglcomp tool command as shown from the command line.

Before you begin:
• It is assumed that you have created a Genero BDL test file, for example myapp_tests.4gl.
• It is assumed that you have placed the test file in the directory path/to/myTests.
• If you are using the GAS:
  • Make sure your GAS version is at least version 2.50.34 or greater.
• Make sure that the standalone dispatcher httpdispatch is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser. See the Genero Application Server Administrator Guide for more information.

To compile your test, at the command line type the following:

cd path/to/myTests ; fglcomp myapp_tests.4gl

An application (42m) file is created.

How to run BDL tests

The Genero BDL tests that you have compiled, are run from the command line as shown.

When running tests, it is recommend that you pass the name of the application and how to test it as an argument. This can be either:

• a command (for direct mode)
• an URL (when running tests over GAS)

Examples of running tests for both methods are given. For an example of how to handle the argument values in your code, see the sample $FGLDIR/testing_utilities/ggc/samples/SimpleTest.

Before you begin:

• It is assumed that you have compiled the Genero BDL test file, for example myapp_tests.4gl.
• It is assumed that you have placed the compiled file in the directory path/to/myTests.

1. To run the tests through the GAS using the application URL, type the commands shown below:

```
cd path/to/myTests ; fglrun myapp_tests http://localhost:6394/ua/r/myapp
```

Or alternatively you can run the test in direct connection mode without using the GAS.

2. To run the tests using a stand-alone DVM, type the command:

```
cd path/to/myTests ; fglrun myapp_tests "fglrun myapp"
```

Important: It is assume that the BDL test file has been placed in the same directory as the tested application, for example path/to/myTests.

Write, compile, and run Java tests

The Genero Ghost Client has different methods for creating tests depending on whether you are using Genero BDL or Java. In this section the topics are about writing, compiling, and running Java tests.

• Write a Java test on page 13
• How to compile Java tests on page 15
• How to run Java tests on page 15

Write a Java test

Use the information and the examples shown here to begin writing your own test scenarios. A sample program is provided to assist you in writing your own Java tests.

Note:

The complete details of the packages and classes that make up the Genero Ghost Client can be found in the /doc directory of your GGC package. Please see the help file by launching the /doc/index.html file in your browser.

1. Begin your Java code by importing the Genero Ghost Client Java classes.

Your session manager function manages the whole test session life cycle. It requires these classes:
• The SessionManager class.
• The Scenario class. You need to instantiate a scenario by including a getScenario() function. You can have a sequence of scenarios, if required, to run your tests.

In this example, the scenario instance created is given to the main application. This is the first application that will be launched when the test is run.

If child applications are run by the main application, the SessionManager will also get each of them a scenario and instantiate them, as shown in the example.

```java
import com.fourjs.ggc.Scenario;
import com.fourjs.ggc.SessionManager;

public class mySessionManager
    implements SessionManager
{
    boolean mStarted = false;
    public Scenario getScenario()
    {
        if (!mStarted) {
            mStarted = true;
            return new FirstScenario(); // will be given to the first 4gl application, the mother
        } else {
            return new SecondScenario(); // will be given to the second 4gl application, the child launched via run or run without waiting by the mother
        }
    }
}
```

2. Write a function to test your application.

Your test function or scenario requires these methods:

• A play(GhostRunner runner) method, which describes the user interaction to play; that is the sequence of tests it plays.
  
  You decide what tests are to be carried out by providing within the play(GhostRunner runner) function the required user interactions.

• An InvokeFrontcall(String module, String name, String[] args) method, which describes how to handle the call to the front-end client when and if it is required by the application.

• A getScenarioListener() method. This method is called by the GhostRunner framework to perform runtime monitoring.

In this code sample the CustOrders application from the FGLGWS demo is being tested for integrity.

The mRunner.getActions() method passes details about the form's menu actions into an array and checks these against an expected list of actions. If an action menu is missing, the runner.log().error method outputs the information to the standard output.

```java
/**
 * The Scenario to test the "CustOrders" application from the FGLGWS demo *
 */
public void play(GhostRunner runner)
{
    mRunner = runner;
    try {
        // at least, we expect these actions
```
String[] expected = {"zoom_city", "cust_query", "cust_next", "cust_last", "dialogtouched", "cust_append", "cust_delete", "ord_append", "ord_modify", "ord_delete", "close"};
ArrayList<ActionInfo> actions = mRunner.getActions();
for (String act : expected) {
    boolean present = false;
    for (ActionInfo action : actions) {
        if (act.equals(action.getName())) {
            present = true;
            break;
        }
    }
    if (!present) {
        runner.log().error("CustomerOrderScenario: action [", act, "] is missing");
    }
}

How to compile Java tests
To compile your Java tests, you run the javac command as shown from the command line..

Before you begin:

- It is assumed that you have built two new test Java files:
  - One SessionManager implementation named, for example, mySessionManager.java that will manage the testing session
  - One Scenario implementation named, for example, myScenario.java that will instantiate real scenarios based on the application launched.
- It is assumed that your new test Java files are in the package path.to.myTests and therefore you have placed them in the directory path/to/myTests.
- It is assumed your environment is configured to run Java and the GGC, see Install and configure for Genero Ghost Client on page 5.
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
  - Make sure that the standalone dispatcher httpdispatch is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser. See the Genero Application Server Administrator Guide for more information.

To compile your newly created tests, type the following:

javac path/to/myTests/*.java

How to run Java tests
The Java tests that you have compiled, are run from the command line using the Launcher of the Genero Ghost Client Java program which starts a testing session for an application.

Before you begin:

- It is assumed that your new test Java files are compiled in the directory path/to/myTests.
- It is assumed your environment is configured to run Java and the GGC, see Install and configure for Genero Ghost Client on page 5.
- If you are using the GAS:
  - Make sure your GAS version is at least version 2.50.34 or greater.
• Make sure that the standalone dispatcher httpdispatch is started and that you can access the GAS demos welcome page, http://localhost:6394/demos.html, from your browser. See the Genero Application Server Administrator Guide for more information.

1. To run the tests through the GAS using the application URL, type the command:

   ```
   ```

Or alternatively you can run the test in direct connection mode without using the GAS.

2. To run the tests using a stand-alone DVM, type the command:

   ```
   cd path/to/myTests ; java com.fourjs.ggc.Launcher -s path.to.myTests.mySessionManager --command "fglrun myApplication"
   ```

   **Note:**
   
   • We first change to the directory where the application is located.
   • The DVM command to launch the application needs to be passed in double quotes (for example, "fglrun demo").

### Generate a scenario

You generate a scenario from a guilog file.

**Note:** The GGC command line is a string without line breaks. Line breaks may have been added to the command line examples in this section to support the printed version of our documentation.

This procedure creates a test scenario for the Orders.42r application. It assumes that your command line environment is properly set, and that all files can be found. You would replace the name "Orders" with the name of your application.

**Tip:** Open a Genero Workplace Window to provide a well-configured environment.

1. Navigate to your test directory.
   
   This example uses a directory named /test_dir.

2. Type fglrun --start-guilog="Orders.guilog" Orders.42r.
   
   The application launches in recording mode. The --start-guilog option specifies the file name for the guilog.
   
   The Orders.guilog file is created.

3. Use the application to complete a series of steps. When you have completed the steps for your test, exit the application.

4. Type java com.fourjs.ggc.generator.GhostGenerator Orders.guilog

   com.fourjs.ggc.generator.BDLProducer Orders_test.4gl.

   This command uses the GhostGenerator class to generate a scenario from filename.guilog.

5. Type fglcomp -I FGLDIR/testing_utilities/ggc/lib Orders_test.4gl.

   **Note:** The generated test relies on the macros and functions defined in the BDL GGC lib provided in FGLDIR/testing_utilities/ggc/lib. The path must be included (-I) for the compilation.

   The generated test scenario compiles and creates the application file Orders_test.42m.

6. Type one of the following commands to run the test.

   To directly launch the test application: fglrun Orders_test --launcher "fglrun Orders" --workingDir "\test_dir" -useLogTiming

   To run the test application using the Genero Application Server: fglrun Orders_test --launcher "http://localhost:6394/ua/r/Orders" -useLogTiming

   The test scenario application runs against the referenced application in the referenced location.
These are options available for use by the generated test, to provide details needed to run the test successfully. Each option can be omitted if not needed.

---

**--launcher cmd-or-url**

Specify the command or URL to use to run the tested application.

If a URL (a string starting with "http:" or "https:" is detected, the application will run using the GAS and the UA protocol. Otherwise, the GGC will consider the string as a command and connect to it using the direct protocol.

For example: `fglrn Orders_test --launcher "fglrn Orders"`

If the `--launcher` option is not specified, the tested application will run using the command `fglrn appName` where `appName` is extracted from the guilog used to generate the application.

---

**--workingDir path**

Specify the application working directory when running is direct mode. The working directory of the application process will be set to this path.

For example: `fglrn Orders_test --launcher "fglrn Orders" --workingDir "\test_dir"

If the `--workingDir` option is not specified, the default is the current working directory.

This parameter should not be specified when running against the UA protocol.

---

**--useLogTiming**

When specified, the scenario will play at the pace specified in the guilog; the test executes in the same time as the recorded session.

For example: `fglrn Orders_test --launcher "fglrn Orders" --useLogTiming fglrn Orders_test "fglrn Orders" "\test_dir"

If this parameter is omitted, the test plays as quickly as possible.

The test runs and the results display in the output.

---

**Handling frontcalls in a generated test**

As a developer, you must handle how frontcalls are handled in your unit test.

There are several frontcall implementations that are handled by default by the GGC:

- Frontcall `ui.Interface.frontCall("standard","feInfo",["feName"],[info]) returning "GGC"
- Frontcall `ui.Interface.frontCall("standard","feInfo",["osType"],[info]) returning "Windows" or "Unix"

For all others, you must decide how the test will handle the frontcall. You may:
• Ignore any frontcall and manage the behavior in the source application by nesting your frontcalls in an IF statement:

```plaintext
CALL ui.Interface.frontCall("standard", "feInfo", ["feName"], [info])
IF info != "GGC" THEN
  CALL ui.Interface.frontCall("standard", "feinfo", "userPreferredLang", result)
... END IF
```

• Decide to mimic the recorded log exactly. You register the frontcall values using the `com.fourjs.ggc.fgl.Application.registerFrontcallResult()` API. Refer to the GGC javadoc for more information.

• Decide to use generic values. You register the generic values using the `com.fourjs.ggc.fgl.Application.registerFrontcallResult()` API. Refer to the GGC javadoc for more information.

Depending on the call, the correct option for each frontcall in your test may differ.

**Finding frontcalls in the generated test**

If your application includes a frontcall, the frontcall is highlighted in the generated test within comments.

For example, if your application contains this frontcall:

```plaintext
CALL ui.Interface.frontCall("standard", "feinfo", "userPreferredLang", result)
DISPLAY result
```

The generated unit test will include comments:

```plaintext
-- Frontend Call:
-- FunctionCall 100 (moduleName = "standard", name = "feinfo", isSystem = "0",
  paramCount = "1", returnCount = "1") {
  -- FunctionCallParameter 101 (dataType = "STRING", value = "userPreferredLang",
  isNull = "0") { } }
-- Frontend result:
-- <FunctionCallEvent id="0" result="0" ><FunctionCallReturn id="0"
  dataType="STRING" isNull="0" value="en_US" ></FunctionCallReturn></FunctionCallEvent>
```

It is important to understand that no actual 4GL code is created in the generated test. It is up to you to decide what you want to return for the frontcall and modify the generated test accordingly.

---

**Genero Ghost Client reference**

Reference topics for the Genero Ghost Client.

**GhostGenerator reference**

The `GhostGenerator` class is the entry point to create a scenario.

The complete details of the packages and classes that make up the Genero Ghost Client can be found in the `/doc` directory of your GGC package. Please see the help file by launching the `/doc/index.html` file in your browser.
Syntax

```
java com.fourjs.ggc.generator.GhostGenerator path-to-guilog
   producer-classname path-to-scenario
```

Table 3: GhostGenerator options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>path-to-guilog</code></td>
<td>Specify the path to the guiLog generated using the --start-guilog option of <code>fglrun</code>. See Record a scenario log on page 24.</td>
</tr>
<tr>
<td><code>producer-classname</code></td>
<td>Specify the name of the <code>ScenarioProducer</code> implementation. You can create your own implementation, or you can use the provided <code>BDLProducer</code> implementation to create a scenario in Genero Business Development Language (BDL).</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> To create your own producer or template, start by reviewing the documentation for the Genero Ghost Client API.</td>
</tr>
<tr>
<td></td>
<td>The complete details of the packages and classes that make up the Genero Ghost Client can be found in the <code>/doc</code> directory of your GGC package.</td>
</tr>
<tr>
<td></td>
<td>Please see the help file by launching the <code>/doc/index.html</code> file in your browser.</td>
</tr>
<tr>
<td><code>path-to-scenario</code></td>
<td>Specify the path and filename of the source code file to be created for the scenario.</td>
</tr>
<tr>
<td><code>--help</code></td>
<td>Print the usage and exit.</td>
</tr>
</tbody>
</table>

Example

This example creates a Genero BDL application from the GuiLog named `myapp.guilog`. The application name will be `myapp.4gl`.

```
java com.fourjs.ggc.generator.GhostGenerator myapp.guilog
   com.fourjs.ggc.generator.BDLProducer myapp.4gl
```

Launcher reference

The `Launcher` class is the entry point to run a scenario written by hand in Java.

The complete details of the packages and classes that make up the Genero Ghost Client can be found in the `/doc` directory of your GGC package. Please see the help file by launching the `/doc/index.html` file in your browser.

Syntax

```
java com.fourjs.ggc.Launcher [options]
```

Table 4: Launcher options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--url url</code></td>
<td>Specify the URL to use to test your application, see Launch tests to different front-ends on page 21.</td>
</tr>
<tr>
<td><code>-u url</code></td>
<td></td>
</tr>
</tbody>
</table>
### Option | Description
---|---
--manager session-manager-classname  
-s session-manager-classname | Specify the fully package-qualified java SessionManager to use.  
**Important**: This option is mandatory.

--command command  
-cd command | Specify a test from a stand-alone DVM, using the direct connection mode.  
**Note**: Not used with the URL -u option, see Launch tests with direct connection on page 20

--command-wd workingDir  
-cwd workingDir | Specify the directory from where to launch the application being tested.

--thread_number number  
-t number | Specify the number of thread instances to launch to simulate the number of users using an application.

--thread_delay delay  
-td delay | Specify the delay (in seconds or milliseconds) between the launch of Ghost Client sessions when running multiple thread instances. The default unit is seconds; append "ms" for milliseconds.

--no_x509_check  
-x | Prevent the server X.509 certificates checking.

--help  
-h | Print the usage and exit.

--version  
-v | Print the version and exit.

See also Display version number on page 21.

---

**Launch tests with direct connection**

Run the integrity tests in direct connection mode.

In the syntax of the following command line example it is assumed that the GGC package was unzipped in the same directory where you are running the Launcher, and that your environment is configured to run Java and the GGC, see Install and configure for Genero Ghost Client on page 5.

For example, to run the integrity tests for the `demo` application from a stand-alone DVM, using the direct connection mode, you run this command:

```bash
cd FGLDIR\demo; java com.fourjs.ggc.Launcher -s com.fourjs.ggc.testcases.IntegrityTestsSessionManager --command "fglrun demo"
```

In this example a URL is not used to launch the application. Therefore instead of the -u option, the Launcher is run with just the -s option and the path to the session manager.

The `--command` option is added to invoke the DVM to launch the application for testing.

**Note:**
- We first change to the directory where the application is located.
- The DVM command to launch the application needs to be passed in double quotes (for example, "fglrun demo").
**Launch tests to different front-ends**

Run tests behaving as different front-ends.

In the syntax of the following command line example it is assumed that the GGC package was unzipped in the same directory where you are running the Launcher, and that your environment is configured to run Java and the GGC, see [Install and configure for Genero Ghost Client](#) on page 5.

Depending on how you want to test, run the command option in example 1 to run your application tests against different front-end clients, or run the command option in example 2 to test all clients at once:

1. `java com.fourjs.ggc.Launcher -u http://localhost:6394/ua/r/myapp -s path.to.mySessionManager.mySessionManager`

   The **Launcher** is the entry point of the Genero Ghost Client Java program which starts a testing session for an application. This command is run by launching Java to run the **Launcher** class followed by the -u option and the URL of your application. The -s option is used followed by the path to the session manager.

   **Tip:** Depending on the user agent protocol you provide in the URL, you can specify what front-end the test is to be run as:
   - `/ua/r/`; for version 3 clients: Genero Web Client for JavaScript (GWC-JS), GDC 3.00, GMA, or GMI - UAProxy.
   - `/ja/r/`; for Genero Desktop Client (GDC 2.50) - GDCProxy.
   - `/wa/r/`; for Genero Web Client for HTML5 (GWC-HTML5) - HTML5Proxy.

2. `java com.fourjs.ggc.Launcher -u http://<host>:<port>/gas/ua/r/<group>/<myapp> -s path.to.mySessionManager.mySessionManager --all_mode`

   This example is basically the same instruction but with the addition of the --all_mode option at the end to specify that the test is to be run for all clients.

   **Note:** The front-end you specify in the URL (`ua/`) will be the first one to be run.

---

**Macro reference**

Tests created by the Genero Ghost Client (GGC) use macros.

The generated test file includes an **&include** statement which specifies the file containing macros:

```
&include "GGCTestUtilsMacro.4gl"
```

The standard is that macros are written in uppercase. In this snippet from the generated test file, two macros can be found: **WAIT_FORAPPLICATION** and **ASSERT_EQUALS**:

```
... 
WAIT_FOR_APPLICATION("app_1484", 1025) 
ASSERT_EQUALS(getWindowName(), "w_orderform") 
...
```

Refer to the **GGCTestUtilsMacro.4gl** file to view the definitions of each of the macros.

---

**Display version number**

To display the version of the Genero Ghost Client, execute `fpi -l` from the command line.

Refer to the [Genero Business Development Language User Guide](#) for details regarding the `fpi` command.
The logging mechanism

The Genero Ghost Client implements a logging mechanism to display errors, warnings, and information as output from the tests. Learn how to control the output, and use the logging mechanism efficiently to detect and solve application problems.

- GHOSTLOG on page 22
- Log test scenario output on page 22

GHOSTLOG

The GHOSTLOG environment variable defines the logging level for the Genero Ghost Client. It allows you to control the output messages.

The GHOSTLOG setting defines what information is outputted as tests are run, based on the logging detail included in your test play method (see Log test scenario output on page 22).

You can adjust the log level with a numeric or literal value.

Valid literal values include:

- ERROR - Only errors are displayed.
- WARNING - Errors and warnings are displayed.
- INFO - Errors, warnings and info are displayed.
- DEBUG - Errors, warnings, info, and debug information are displayed. In other words, basically everything is displayed.

The Launcher automatically sets GHOSTLOG to INFO. To select a different level, set it at the command line before launching a test.

- On Linux®/UNIX™/macOS™
  export GHOSTLOG=VALUE
  For example:
  export GHOSTLOG=ERROR
- On Windows®:
  set GHOSTLOG = VALUE
  For example:
  set GHOSTLOG = ERROR

Log test scenario output

Use Log class methods to output errors, warnings, and information from test scenarios as they play.

In the play method of your Java test scenarios, for example, to have information about errors, warnings, or debug information displayed as the test is running, add calls to the log class with the required log method, for example:

```java
public void play(GhostRunner runner)
{
    try {
        runner.sendAction("INPUT");
        ArrayList<FieldInfo> fields = runner.getFields();
        for (FieldInfo field : fields) {
            runner.log().info(field.getId(), " / ", field.getName(), " / ", field.getType());
        }
    }
```
In this code sample from the ButtonEditScenario class sample, the ButtonEdit application from the FGLGWS demo is being tested. The `runner.sendAction("INPUT");` method selects the INPUT button of the application. This opens a form for user input.

The `runner.getFields()` method gets details about the form fields; their id, name, and type. Using the `runner.log().info` method outputs this information.

In the catch block, any exception raised by the `GhostException` method can also be logged as an error by the `runner.log().error` method.

Figure 2: ButtonEdit scenario log output

You can adjust the log level with the `GHOSTLOG` environment variable. This will effect what is outputted to the display. However, it depends on how detailed the logging is in your test play method.

**Related concepts**

`GHOSTLOG` on page 22

The GHOSTLOG environment variable defines the logging level for the Genero Ghost Client. It allows you to control the output messages.

**Demos**

The Genero Ghost Client installation includes a sample test scenario for you to explore in the `samples` directory of your GGC package.

Inside the `samples` directory, you find the `SimpleTest.4gl` Genero Ghost Client test scenario, written by hand in Genero. You also find a `java` directory containing a Java example of the SimpleTest application.

**Important:** Use Genero BDL to create GGC applications. Genero applications will work for most unit and load tests. Java should only be used to perform critical load testing.
The SimpleTest demo

With this demo, you can check that everything is correctly set up for compiling and running a test scenario. It also provides a simple template to use as a reference when you start creating your own test scenarios.

To run the SimpleTest test scenario:

1. Compile the test scenario:
   
   fglcomp SimpleTest

2. Execute the test scenario (direct mode):
   
   • Unix:
     
     fglrun SimpleTest "fglrun Edit" $FGLDIR/demo/Widgets
   
   • Windows®:
     
     fglrun SimpleTest "fglrun Edit" %FGLDIR%/demo/Widgets

3. Execute the test scenario (using the Genero Application Server):
   
   fglrun SimpleTest http://localhost:6394/ua/r/simple-demo

4. To execute the test scenario using the Genero Application Server for Java, create an empty war file pointing to the test scenario. Refer to the *Genero Application Server for Java Administrator's Guide* for details on creating the war file.

Recording log files

A recorded log file can be used by the Genero Ghost Client to store the scenario for testing. You can record logs for the purpose of generating scenarios for testing.

The topic in this section details how to record a log file from the Dynamic Virtual Machine (DVM).

- Record a scenario log on page 24
- Generate a scenario from a guilog on page 25

Record a scenario log

Describes the procedure used to record a log file of the interaction with an application launched to a Genero web client.

Record a guilog log using a direct connection

1. Launch the Genero Desktop Client.
2. Start the application using fglrun and the the --start-guilog option.
   
   For example:
   
   D:\fjs\gst\3.10.01\fgl\demo\Widgets>fglrun --start-guilog=D:/tmp/log/edit.log Edit

   In this example:
   
   • The DVM is run with the --start-guilog option to record a log file.
   • The interaction is saved to a file named edit.log.

   For more information on using the --start-guilog DVM option, see the *GUI front-end connection* chapter of the *Genero Business Development Language User Guide*.
3. Use the application.
4. To finish recording, close the application.
When you have completed recording the log, your next task is to generate test sets from the recorded log file. See **Generate a scenario from a guilog** on page 25.

**Record a scenario log via the GAS**

1. To record a log from the DVM end, add an `APPLICATION_EXECUTION_DVM` element to the configuration file of your application.

   ```xml
   <?xml version="1.0" encoding="UTF-8" ?>
   <APPLICATION Parent="defaultgwc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://www.4js.com/ns/gas/3.00/cfextwa.xsd">
   <EXECUTION>
   <PATH><path_to_your_application></PATH>
   <DVM>fglrun --start-guilog=/tmp/myapp.log</DVM>
   <MODULE>myapp.42r</MODULE>
   </EXECUTION>
   <UA_OUTPUT>
   ... 
   </UA_OUTPUT>
   </APPLICATION>
   ```

   In our example:
   - The DVM is run with the `--start-guilog` option to record a log file.
   - The interaction is saved to a file named `myapp.log`.

   For more information on using the `--start-guilog` DVM option, see the GUI front-end connection chapter of the Genero Business Development Language User Guide.

2. Save the configuration file in the `$(res.appdata.path)/app` directory.
3. Run the application in your browser by starting the standalone dispatcher from the command line using `httpdispatch`.
   - Open the application by entering the URL. In this example, the URL is looking for a configuration file named `myapp.xcf`:
     ```
     http://localhost:6394/ua/r/myapp.
     ```
   - As you interact with the application, all this activity is recorded in the file specified.

4. To finish recording, close the application.

   **Warning:** Next time you run the application with this DVM configuration, the log file is overwritten.

**Generate a scenario from a guilog**

Generate test sets from recorded log files.

In the syntax of the following command line examples it is assumed that the GGC package was unzipped in the same directory where you are running the `GhostGenerator`, and that your environment is configured to run Java and the Genero Ghost Client. See Install and configure for Genero Ghost Client on page 5.

This example creates a Genero BDL application from the GuiLog named `myapp.guilog`. The application name will be `myapp.4gl`.

```
java com.fourjs.ggc.generator.GhostGenerator myapp.guilog
```
When you have completed generating the test sets from the log, your next task is to compile and run them. See Write, compile, and run BDL tests on page 10.

Copyrights and Trademarks

The Copyright and trademark page is shared amongst all Four Js documentation. Appearance in this page does not guarantee a mention within the current document.

Copyright notice
© Copyright Four Js Development Tools, Inc. Use, duplication or disclosure restricted.

Trademarks
Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated.

IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corp., in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information.

Linux is a registered trademark of Linus Torvalds.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Other company, product, or service names may be trademarks or service marks of others.
Index

D
Demos
  SimpleTest 23

F
frontcalls
  manage in test 17

G
Generated test
  Macros 21
Genero Ghost Client
  how it works 7
  infrastructure 7
Ghost Client
  compile BDL test 12
  compile Java test 15
  generate test sets from logs 25
  Genero BDL test 10
  Genero Java test 13
  get version 21
  GHOSTLOG 22
  log output 22
  logging 22
  options 18, 18, 19
  recording log files 24
  run BDL test 13
  run Java test 15, 16
  scenario 8
  testing 8
Ghost Client Testing Tool 5

I
Installation 5

T
testing
  load tests 9
  performance test 9
  unit tests 9

W
what's new
  Version 1.10 4