Build and Source Control - Feature #6183

design and implement (as needed) the approach for writing automated 4GL compatibility tests

03/16/2022 11:53 AM - Greg Shah

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Build and Source Control - Support #6858: rework existing testcase...	**Closed**

History

#1 - 04/13/2022 09:19 AM - Greg Shah

In this task I want to find some answers so that we can move ahead with implementing the full approach described in Testing Process for automated non-interactive 4GL compatibility testing.

In Writing 4GL Testcases, we have recorded some best practices for writing tests. I want to extend this as needed to cover the full range of testcases AND we need to decide on the infrastructure that will be used for automation.

I want to start with a discussion of our requirements. Based on that, we will make decisions or further investigations about the automation toolset and any framework code we may need as helpers. Some initial requirements that come to mind:

- we will assume that all 4GL code that has correct syntax and can compile in OE
- must run with both OE and FWD as a back end so that the results are comparable
  - the OE runs would be used during:
    - the development (and maintenance) of the tests themselves
    - capturing results for different versions of OE
    - we will not be using CI to run against OE, but they still must be able to be executed on an automated basis
  - the FWD runs:
    - will be automated using CI/CD so that we can execute them nightly
    - will be able to be run ad-hoc
- subsets of the tests must be able to be executed on their own
- the full set of compatibility testing should be able to be executed as a single batch (if possible)
- specific failures should be able to be easily tracked back to the failing code, very granular reporting with specifics about the test is important
- the tests must record details from the 4GL environment which allow a deep comparison, for example:
  - for error handling, we should be able to easily test error number(s), error message(s), whether it is an error/warning
  - for triggers we should be able to easily check that the event has fired on the correct target widget and perhaps we want to check some (or even a lot of) additional widget state at that moment
- we must have standardized techniques to easily test:
  - boundary conditions
  - side effects
  - error/warning handling
  - structured error handling
  - transactions/subtransactions/undo
  - control flow, retry, infinite loop protection

Before we start the discussion, please review the library_calls/ tests in Testcases which I wrote for #1634. This is an example of an early approach at automating a complex suite of tests.

Issues with library_calls/:

- Uses a one-off approach for automation with a custom test runner.
- The 4GL helpers are probably too simplistic and could be extended to be more robust.
- Relies too heavily on logging and logfile comparison to debug issues.

Marian: Do you have a set of tests in Testcases we can look at as a good example of some alternative approaches to library_calls/?

#2 - 04/14/2022 02:50 AM - Marian Edu

Greg, all of our tests are using the same approach that is basically generate output in log file and then compare the results. Each test procedure starts by recording a start tag line and ends with a corresponding end tag line, with a number of tests executed. The reason for using the start/end tag lines was to make sure the test procedure executed and completed without (unhandled) errors - if the end tag line is missing then there was some unhandled exception (like a NPE).

As a general rule we always use NO-ERROR and check ERROR-STATUS afterwards, we have some assert helper routines to check for specific error number(s) and to compare the result against expected value - it's basically just assert-true so we had to add extra details in the message ourselves. We only record exceptions in the log file (apart from the start/end tag lines), if we must compare output generated by routines we use separate output files in a 'data output' folder and then compare those with reference output files generated by 4GL that we store in a separate folder. For temp-table and dataset I recall we had some helper assert routines to check the content match - I think we've used that for XML read/write.

It's not very different from your approach, we do not clear the log file on each test just because we wanted to run multiple tests in one run (like a full package test in OO) and check the outcome - if we clear the log file on each test we only get the result from the last test run.

That being said I do not think this can be seen as good practice for automated testing, normally those tests do not 'fail' - you have to check the log file to see the results. Each procedure is a single test although there are a sometimes quite large number of cases that are being tested but in the end we are stil able to see all cases where FWD implementation was different from the 4GL one and ultimately we just need to change the assert routines to log those exceptions in a more structured way (xml/json), sort of what ABLUnit is doing.

But, since we're in Java I would say a better approach would be to hook-up into JUnit somehow to be able to run those in CI/CD environment - well, partially at least since UI testing is not always possible in a non-interactive mode. I saw another discussion about annotations support so using ABLUnit annotations during conversion to generate JUnit tests is probably the natural approach, however this will mean the existing tests in testcases needs to be refactored and this is not going to be easy :(

#3 - 05/30/2022 03:48 PM - Greg Shah

I agree that the best approach is to use some form of JUnit-like approach.

We have customers using both OEUnit (#6237) and ABLUnit (#3827) so we will be implementing a replacement for both. Hopefully we can implement a single approach that handles both. My plan was to implement a JUnit "TestEngine" to handle the integration between JUnit and the FWD server where code must run.

For OEUnit, it is an open source project so we could actually convert it and just run it directly inside the FWD server. On the other hand, I suspect it may need modifications to work properly. It comes with its own GUI for test execution where I would prefer to use JUnit directly. So it is probably best to convert as JUnit applications and implement the TestEngine like with ABLUnit.

From my quick look, ABLUnit is not a 100% replacement for OEUnit so there are pros and cons of each. Do you have an opinion on an approach? I assume we would choose one or the other so that the tests could run in OpenEdge and then we would be able to convert and run in FWD as well.

> however this will mean the existing tests in testcases needs to be refactored and this is not going to be easy :(

Given that we convert to ABLUnit or OEUnit, what is the work effort to rework the tests into this approach?

#4 - 10/13/2022 03:02 PM - Greg Shah

OK, we will write our non-interactive 4GL compatibility tests using unit testing.

Marian: Do you have a recommendation between OEUnit and ABLUnit? I want to standardize our work on just one of these.

One thing to consider: we will need a mechanism to disable tests at runtime (e.g. those tests which we already know cannot work in FWD because of missing implementation).

#5 - 10/14/2022 02:05 AM - Marian Edu

Greg Shah wrote:

> OK, we will write our non-interactive 4GL compatibility tests using unit testing.
>
> Marian: Do you have a recommendation between OEUnit and ABLUnit? I want to standardize our work on just one of these.

I would go with ABLUnit due to the 'conservative' nature of most PSC customers that always wait for PSC to provide something, although the type of customers interested in FWD as an alternative might be exactly the opposite and might have looked around before ABLUnit... I just find the 'extensions' in OEUnit not very useful if not complete nonsense like the DataProvider, I know Cameron and he's a great guy but sometimes just because you can doesn't mean you should :)

> One thing to consider: we will need a mechanism to disable tests at runtime (e.g. those tests which we already know cannot work in FWD because of missing implementation).

Both have support for @Ignore.

#6 - 10/14/2022 05:33 AM - Greg Shah

In our experience, ABLUnit is used more widely, which means it is probably the safer choice. More developers will know it, it will be better tested.

OK, we will use ABLUnit.

#7 - 11/03/2022 03:11 PM - Greg Shah

> Both have support for @Ignore.

Using @Ignore will work for something we want to exclude from testing in both environments OR if we use a helper include file, then we can use the FWD-VERSION preprocessor definition to conditionally disable something in FWD only. This will work for code that can successfully convert.

I think there is another case to consider: the scenario where a test won't convert in FWD. We should have a standard approach for such cases. Perhaps we use conditional preprocessing with the FWD-VERSION preprocessor definition for this as well (allow the code to be present in OE and drop it out if in FWD).

#8 - 11/03/2022 03:43 PM - Greg Shah

We should consider how we measure code coverage during testing. The most obvious choice is JaCoCo. It is open source, has good features and there are good integrations with IDEs, build environments and CI/CD like Jenkins. Considerations:

- At one point (some years ago), a customer was running a FWD server using JaCoCo. When JaCoCo was used, it caused at least one bug that could not be otherwise reproduced. Since JaCoCo (and most of the code coverage tools) use some form of static or dynamic bytecode instrumentation to measure coverage, it can (and did) introduce problems in complex projects like FWD. It is perfectly likely that this particular bug is gone now but we really don't know.
- We need to confirm that the results of many different runs (even on different FWD servers) can be aggregated into a single set of reports. We don't expect any single unit testing run to execute all tests. Without aggregation, code coverage reports would not be very useful. My understanding is that JaCoCo does have this capability, but it will need to be proven.
- The technique itself would not be something we always want to have activated. Early on, I think we would use it periodically to get the reports. Keeping it disabled by default is probably a better approach to ensure that bugs are not introduced and performance is not impacted.

#9 - 11/04/2022 11:55 AM - Greg Shah

Email discussion with Marian:

> Since ABLUnit works both with procedures and classes what would be the preference,

I have no hard preference. My instinct is to use OO as the default approach to structure code but to use procedures where it makes sense. To the degree that some features need to be implemented in procedures, those should be used as needed.

I'm open to your feedback.

> not sure if both are supported by Vladimir's engine but there is no point in using both 'styles'?

The intention is for us to support the full range of behavior, so we should pick whatever makes sense. It is important that we do have enough tests to ensure that Vladimir's ABLUnit engine work can itself be tested. Other than that we have no hard requirements.

#10 - 11/14/2022 08:14 AM - Vladimir Tsichevski

1. #6183-4.
   One thing to consider: we will need a mechanism to disable tests at runtime (e.g. those tests which we already know cannot work in FWD because of missing implementation).
   AFAIK, there is no suitable mechanizm in ABLUnit or OEUnit, but we can use usual approach with conditional compilation.
2. #6183-5.
   Both have support for @Ignore.
   IMO @Ignore is not what we need here: we still need to be able to run test in OE.

1. #6183-7.
   Perhaps we use conditional preprocessing with the FWD-VERSION preprocessor definition for this as well
   I would choose this approach too.
2. #6183-6.
   In our experience, ABLUnit is used more widely, which means it is probably the safer choice. More developers will know it, it will be better tested.
   Agreed. We probably need to support OEUnit if some customers still use it, but we have no reason to use both similar engines in our own projects.

1. #6183-9
   not sure if both are supported by Vladimir's engine but there is no point in using both 'styles'?
   Yes, both are supported.

#11 - 11/14/2022 09:54 AM - Marian Edu

Vladimir Tsichevski wrote:

> 1. #6183-5.
>   Both have support for @Ignore.
>   IMO @Ignore is not what we need here: we still need to be able to run test in OE.
>
> 1. #6183-7.
>   Perhaps we use conditional preprocessing with the FWD-VERSION preprocessor definition for this as well
>   I would choose this approach too.

Those two are going to be used together, most probably we'll use a `FWD-IGNORE` preprocessor which will 'inject' the @Ignore annotation on FWD side.

#12 - 11/14/2022 09:59 AM - Vladimir Tsichevski

Marian Edu wrote:

> Vladimir Tsichevski wrote:
>
> 1. #6183-5.
>   Both have support for @Ignore.
>   IMO @Ignore is not what we need here: we still need to be able to run test in OE.
>
> 1. #6183-7.
>   Perhaps we use conditional preprocessing with the FWD-VERSION preprocessor definition for this as well
>   I would choose this approach too.
>
> Those two are going to be used together, most probably we'll use a `FWD-IGNORE` preprocessor which will 'inject' the @Ignore annotation on FWD side.

This will not work if the **conversion** of the test code is currently unsupported.

#13 - 11/14/2022 10:42 AM - Marian Edu

Vladimir Tsichevski wrote:

> This will not work if the **conversion** of the test code is currently unsupported.

You mean there is no conversion support for the class/method/properties under test (aka the converted Java code does not compile)?

Anyway, for this to work the preprocessor directives should have been evaluated before the ABLUnit annotations else the `injected` @Ignore won't be visible down-stream. It will probably be easier to just add a @FwdIgnore annotation as alias for @Ignore, that one will only have meaning in FWD then.

#14 - 11/14/2022 11:09 AM - Vladimir Tsichevski

Marian Edu wrote:

> Vladimir Tsichevski wrote:
>
> > This will not work if the **conversion** of the test code is currently unsupported.
>
> You mean there is no conversion support for the class/method/properties under test (aka the converted Java code does not compile)?

Yes

> Anyway, for this to work the preprocessor directives should have been evaluated before the ABLUnit annotations else the `injected` @Ignore won't be visible down-stream.

AFAIK the preprocessor directive will cut out the code before the 4gl parser sees it, so the cut out code can contain even unparsable stuff.

> It will probably be easier to just add a @FwdIgnore annotation as alias for @Ignore, that one will only have meaning in FWD then.

This would contradict occam's razor: we should avoid inventing extra stuff if we can go without this. Besides, this will not protect us from the stuff we cannot process with FWD yet.

#15 - 11/17/2022 07:27 AM - Vladimir Tsichevski

- Status changed from New to WIP

- File TestErrorSending.cls added

I've created an ABLUnit test for creating and sending all built-in errors implemented in FWD, which can be created from 4gl.
The test is attached as TestErrorSending.cls. It compiles and runs as expected in FWD.

I wonder which repo this and similar tests should belong to?

#16 - 11/17/2022 09:07 AM - Greg Shah

In the "new" Testcases project, there is the abl_unit/ directory with the existing testcases. I think you've been using them in #3827. Your testcase seems to fit there.

#17 - 11/18/2022 02:44 PM - Vladimir Tsichevski

A problem with skeleton/oo4gl/OpenEdge/DataAdmin/Internal/Util/CdcTablePolicyInstanceEnum.cls:

```
enum OpenEdge.DataAdmin.Internal.Util.CdcTablePolicyInstanceEnum:

   def enum.

end enum.
```

This class does not parse:

```
     [java] Builtin class com.goldencode.p2j.oo.dataadmin.internal.util.CdcTablePolicyInstanceEnum is not impl
emented in FWD!
     [java] ./skeleton/oo4gl/OpenEdge/DataAdmin/Internal/Util/CdcTablePolicyInstanceEnum.cls:3:4: unexpected t
oken: def
     [java]     at com.goldencode.p2j.uast.ProgressParser.enum_stmt_body(ProgressParser.java:11080)
     [java]     at com.goldencode.p2j.uast.ProgressParser.enum_def(ProgressParser.java:8050)
     [java]     at com.goldencode.p2j.uast.ProgressParser.single_block(ProgressParser.java:7575)
     [java]     at com.goldencode.p2j.uast.ProgressParser.block(ProgressParser.java:7299)
...
```

#18 - 11/21/2022 09:24 AM - Greg Shah

- Related to Support #6858: rework existing testcases to use ABLUnit added

#19 - 05/25/2023 10:23 AM - Galya B

Moving the #3827#note-469 - #3827#note-472 discussion here. Waiting for a decision on a testing approach for running tests with different client configs. Poll?

#20 - 05/25/2023 10:33 AM - Greg Shah

I have no objection to having different sets of tests in tests/log_manager/<test_set_name>/ directories which each would have their own named client.xml.

I strongly prefer to have tests/<4gl_feature_name>/ as our top level approach.

#21 - 05/25/2023 10:36 AM - Galya B

This could lead to substantial slowing down of a potential future CI plan. Not sure what resources / time starting the engine requires.

#22 - 05/25/2023 10:42 AM - Greg Shah

> This could

What is "This"?

#23 - 05/25/2023 10:47 AM - Galya B

Greg Shah wrote:

> What is "This"?

It's multiplication. Let's say we need 2 types of .xml files and 10 features using each, this makes 20 runs. Otherwise if the dirs are structured around the .xml files, it will be 2 runs.

But there is more to it. What will be the rule of launching a new test run? If it's just the number of folders in tests, they already get too many.

#24 - 05/25/2023 10:53 AM - Galya B

The matter is complicated anyways, because to have a good coverage we should probably come up with hundreds of tests for the 6 startup params of LOG-MANAGER + all the invalid corner cases I can think of. And in certain scenarios the tests themselves may not have to change, while the configs may do.

#25 - 05/25/2023 10:57 AM - Greg Shah

Galya B wrote:

> Greg Shah wrote:
>
> > What is "This"?
>
> It's multiplication. Let's say we need 2 types of .xml files and 10 features using each, this makes 20 runs. Otherwise if the dirs are structured around the .xml files, it will be 2 runs.

The xml file issue is primarily for log-manager and some i18n features. For 99% of our testcases, this is not an issue. I don't want to avoid a good structure just so that a small portion of the overall testcases can be organized by xml file.

In this case, 4GL feature is log-manager and the subdirs could in fact each have their own xml. Why is that an issue?

> But there is more to it. What will be the rule of launching a new test run? If it's just the number of folders in tests, they already get too many.

I would expect that we will normally be running testing at the 4GL feature level, but it will depend on the feature since some like log-manager may need a more granular approach.

#26 - 05/25/2023 10:58 AM - Greg Shah

Galya B wrote:

> The matter is complicated anyways, because to have a good coverage we should probably come up with hundreds of tests for the 6 startup params of LOG-MANAGER + all the invalid corner cases I can think of. And in certain scenarios the tests themselves may not have to change, while the configs may do.

Yes, I would expect so. That is why I suggested in #3827-470 that we should consider if we can write common code and then test it different ways using different client.xml files.

#27 - 05/25/2023 10:59 AM - Galya B

Greg Shah wrote:

> I would expect that we will normally be running testing at the 4GL feature level, but it will depend on the feature since some like log-manager may need a more granular approach.

Startup args are handled differently from programmatically set attributes. It's something to consider for all 4GL features that can be affected by startup args.

#28 - 05/25/2023 11:01 AM - Galya B

Greg Shah wrote:

> In this case, 4GL feature is log-manager and the subdirs could in fact each have their own xml. Why is that an issue?

At the moment one run covers all tests. With another structure, there will be a minimum number of runs equal to the number of covered features.

#29 - 05/25/2023 11:02 AM - Greg Shah

> I would expect that we will normally be running testing at the 4GL feature level, but it will depend on the feature since some like log-manager may need a more granular approach.

> Startup args are handled differently from programmatically set attributes. It's something to consider for all 4GL features that can be affected by startup args.

Are you talking about something specific to log-manager here? I would have expected that setting the entry-types via command line versus via program setter would have the same result from a 4GL code perspective. Obviously we implement the actual setting differently in FWD, but are you saying that the 4GL code results are different?

Most 4GL features are not affected by startup args. log-manager and i18n are exceptions.

#30 - 05/25/2023 11:04 AM - Greg Shah

> In this case, 4GL feature is log-manager and the subdirs could in fact each have their own xml. Why is that an issue?

> At the moment one run covers all tests. With another structure, there will be a minimum number of runs equal to the number of covered features.

We already know that reliance upon startup parameters causes multiple runs to be required. There is no easy way around that without a major redesign to our ABLUnit support. I am not planning on a redesign. If log-manager and i18n need multiple runs, then that is just how it will work.

#31 - 05/25/2023 11:16 AM - Galya B

Here is the full list: Startup-Parameter-Descriptions . We haven't implemented many indeed: The list from ClientParameters is windowSystem, param, startupProc, debugAlert, errorStack, batchMode and the LOG-MANAGER related clientLog, logEntryTypes, loggingLevel, logThreshold, numLogFiles,

inp. Exactly that's why I'm asking if only for log manager it's a good option to have separate runs for each feature.

> Obviously we implement the actual setting differently in FWD, but are you saying that the 4GL code results are different?

I think we're testing FWD and if FWD is compatible with 4GL and that's why we should test it from each angle. Yes, it's different in FWD how the attribute gets set from cmd line or via the handle's method.

#32 - 05/25/2023 11:19 AM - Galya B

Greg Shah wrote:

> I am not planning on a redesign.

I'm asking if you will accommodate testing on LOG-MANAGER. You decide if you want to plan a redesign.

> If log-manager and i18n need multiple runs, then that is just how it will work.

Exactly. What is the automated rule making a script able to determine which should be in one classpath or another?

#33 - 05/25/2023 11:25 AM - Greg Shah

I'm not expecting an automated rule for this. Our testing environments will already be split over multiple systems and will have quite a bit of complexity (database dependencies, appserver, rest, external API dependencies, batch mode/gui/chui...). It will not be common to add a top level feature or a sub-feature and when we do we can check in the right scripting to go along with it.

#34 - 05/25/2023 11:26 AM - Greg Shah

> I'm asking if you will accommodate testing on LOG-MANAGER.

If the above discussion does not already accommodate log-manager, then help me understand what else is needed.

#35 - 05/25/2023 11:54 AM - Vladimir Tsichevski

Is it possible to provide the ability to configure/start/stop the log manager without stopping Java machine? We can add this feature for testing purposes solely.

My experience shows that if we want to unit-test software, we usually should redesign it or refactor to provide additional test points.

#36 - 05/25/2023 12:06 PM - Marian Edu

Galya B wrote:

> The matter is complicated anyways, because to have a good coverage we should probably come up with hundreds of tests for the 6 startup params of LOG-MANAGER + all the invalid corner cases I can think of. And in certain scenarios the tests themselves may not have to change, while the configs may do.

@Galya, I do not think we need to actually start the tests with all kind of start-up parameters, after all those start-up parameters just provides the initial values for some system-wide settings that (most of the time) can be set/changed at runtime. As far as the log-manager is concerned the tests that we wrote update those settings at runtime and tests the behaviour based on those setting, we're not assuming any start-up parameter to be set.

There could probably be tests for validating the fact that the start-up parameters (or xml client/server configuration) gets applied correctly and those tests could potentially instantiate the client more than once although I do not think those needs to be `4GL` unit-tests, pure Java unit-tests could work just as well and imho some things should probably have unit-tests right in FWD engine (aka junit) instead of just 4GL style unit-tests (which in this context are more integration tests than pure unit-tests anyway).

#37 - 05/25/2023 12:19 PM - Marian Edu

Greg Shah wrote:

> I have no objection to having different sets of tests in tests/log_manager/<test_set_name>/ directories which each would have their own named client.xml.
>
> I strongly prefer to have tests/<4gl_feature_name>/ as our top level approach.

Greg, now that I think about it (and after spending lot of time with testcases) I would say we should have had tests separated based on the 'execution environment' as top level - server side business logic, character client, gui, application server, rest, web services, web handler... this way one can run only the tests pertaining for a specific customer environment and the 'test engine' could also be setup accordingly (chui/gui).

Like for security feature instead of dealing with multiple databases each with different security setup in a single 'test suite' that potentially test everything, we can have smaller test suites that are setup to run with different configurations to test specific a specific feature/use-case.

#38 - 05/25/2023 02:13 PM - Vladimir Tsichevski

Some sources in testcases project do not convert correctly. Namely, the tests/ui_tests/fill_in/TestAttrSpace.cls converts to produces the follow Java code:

```
   @LegacySignature(type = Type.METHOD, name = "TestSetValid")
   @Test
   public void testSetValid()
   {
      internalProcedure(TestAttrSpace.class, this, "TestSetValid", new Block((Body) () ->
      {
         silent(() ->
         {
            //  static
            hFillS;
            new logical(true);
         });
         Assert.isTrue(hFillS);
```

Which contains 3 Java errors in it.

What should we do with such tests? Create issues in Base Language or Conversion Tools? Document such source as uncompilable, so other can exclude them from conversion lists? Create a project-wide unconvertable file source list in repository for everyone's use?

#39 - 05/25/2023 02:33 PM - Vladimir Tsichevski

I have a problem having multiple versions of Java on my computer, so I have to force the source and target compatibility in a whole project to one version. To make this more easy, I modified the build.xml, so the source and target version can be set.

Below is the diff, it sets the java version to 1.8, this can be changed in build.properties, if needed.

If nobody objects, I will commit the change to the testcases project.

```
=== modified file 'build.xml'
--- build.xml    2023-04-26 08:15:31 +0000
+++ build.xml    2023-05-25 18:26:10 +0000
@@ -98,12 +98,16 @@
      compile.debug        Should compilation include the debug option?
      compile.deprecation  Should compilation include the deprecation option?
      compile.optimize     Should compilation include the optimize option?
+     compile.source       Java source compatibility, default is 1.8.
+     compile.target       Java target class compatibility, default is 1.8.

    -->

    <property name="compile.debug"       value="true"/>
    <property name="compile.deprecation" value="false"/>
    <property name="compile.optimize"    value="true"/>
+   <property name="compile.source"      value="1.8"/>
```

```
+    <property name="compile.target"      value="1.8"/>

     <!-- path used when compiling Java classes -->
     <path id="compile.classpath">
@@ -455,6 +459,8 @@
        <!-- Compile DMO and UI classes only -->
        <javac srcdir="${src.home}"
               destdir="${build.home}/classes"
+              source="${compile.source}"
+              target="${compile.target}"
               includes="${pkgroot}/dmo/**
                         ${pkgroot}/ui/**"
               debug="${compile.debug}"
@@ -469,6 +475,8 @@
        <!-- Compile all other classes -->
        <javac srcdir="${src.home}"
               destdir="${build.home}/classes"
+              source="${compile.source}"
+              target="${compile.target}"
               debug="${compile.debug}"
               deprecation="${compile.deprecation}"
               memoryMaximumSize="4G"
```

#40 - 05/26/2023 02:45 AM - Galya B

Marian Edu wrote:

> after all those start-up parameters just provides the initial values for some system-wide settings that (most of the time) can be set/changed at runtime.

LOG-THRESHOLD:

> LOG-THRESHOLD attribute
> The file size threshold of log files. When the current log file becomes equal to or greater than the specified size, the AVM renames and saves the log file and creates a new log file.
>
> Data type: INTEGER
>
> **Access: Read-only**
>
> Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

NUM-LOG-FILES

> NUM-LOG-FILES attribute
> The number of rolled over log files to keep on disk at any one time, for ABL session, including the current log file.

> Data type: INTEGER
>
> **Access: Read-only**
>
> Applies to: DSLOG-MANAGER system handle, LOG-MANAGER system handle

#41 - 05/26/2023 02:55 AM - Marian Edu

Galya B wrote:

> Marian Edu wrote:
>
> > after all those start-up parameters just provides the initial values for some system-wide settings that (most of the time) can be set/changed at runtime.
>
> LOG-THRESHOLD:
>
> > LOG-THRESHOLD attribute
> > The file size threshold of log files. When the current log file becomes equal to or greater than the specified size, the AVM renames and saves the log file and creates a new log file.
> >
> > Data type: INTEGER
> >
> > **Access: Read-only**

Yes, that's why I was saying 'most of the time' :)

Like for i18n there are some parameters that can only be set at start-up, for those cases imho there should be separate test suites for start-up parameters. I don't see how 4GL unit tests could be written for those cases anyway, at the point the test is executing we've already passed that point and the AVM is up and running :)

In this particular case I would just write a junit test that start-up the FWD engine with a smaller numbers for num files/threshold and log messages to a point the log rotation mechanism is validated, this is completely transparent for a 4GL developer.

#42 - 05/26/2023 02:58 AM - Galya B

Marian Edu wrote:

> Greg Shah wrote:
>
> > I have no objection to having different sets of tests in tests/log_manager/<test_set_name>/ directories which each would have their own named client.xml.
> >
> > I strongly prefer to have tests/<4gl_feature_name>/ as our top level approach.
>
> Greg, now that I think about it (and after spending lot of time with testcases) I would say we should have had tests separated based on the 'execution environment' as top level - server side business logic, character client, gui, application server, rest, web services, web handler... this way one can run only the tests pertaining for a specific customer environment and the 'test engine' could also be setup accordingly (chui/gui).
>
> Like for security feature instead of dealing with multiple databases each with different security setup in a single 'test suite' that potentially test everything, we can have smaller test suites that are setup to run with different configurations to test specific a specific feature/use-case.

Actually there is a way to achieve both requirements, but it's about automation.

The structure of each feature in a separate dir can be saved, while the actual "test suite" for a match of configs (server, client and db) can be compiled by a script that uses a list of tests to form the classpath. Then there will be a second root folder for the automation with a dedicated sub-folder for each variant with the combo of configs and classpath list. It's a bit more manual work, but it will be very flexible. In the future a Dockerfile can be added to each config folder to prepare an image for a quick execution of tests in the CI.

#43 - 05/26/2023 03:08 AM - Marian Edu

Vladimir Tsichevski wrote:

> Some sources in testcases project do not convert correctly. Namely, the tests/ui_tests/fill_in/TestAttrSpace.cls converts to produces the follow Java code:

That was one of the reasons behind the test skeleton generator, at least we know that those attributes/methods are supported by the conversion.

> What should we do with such tests? Create issues in Base Language or Conversion Tools? Document such source as uncompilable, so other can exclude them from conversion lists? Create a project-wide unconvertable file source list in repository for everyone's use?

Nothing better than test cases that fail :)
I think there is a separate task for testing the conversion (#6860, #6861) so, maybe create sub-tasks there for each issue found?

#44 - 05/26/2023 07:12 AM - Vladimir Tsichevski

Marian Edu wrote:

> What should we do with such tests? Create issues in Base Language or Conversion Tools? Document such source as uncompilable, so other can exclude them from conversion lists? Create a project-wide unconvertable file source list in repository for everyone's use?

> Nothing better than test cases that fail :)

I did not get it: the result of conversion cannot be compiled. What is dead may never die :-)

> I think there is a separate task for testing the conversion (#6860, #6861) so, maybe create sub-tasks there for each issue found?

Subtasks are banned at GCD Redmine AFAIK.

#45 - 05/26/2023 07:59 AM - Greg Shah

> Below is the diff, it sets the java version to 1.8, this can be changed in build.properties, if needed.

We support both Java 8 and Java 11 right now. I hesitate to hard code things to Java 8.

> Subtasks are banned at GCD Redmine AFAIK.

Not banned, but only to be used sparingly. When you use them, the parent task cannot be closed until all child tasks are complete. That is often not what we want.

#46 - 05/26/2023 08:03 AM - Greg Shah

> What should we do with such tests? Create issues in Base Language or Conversion Tools?

Yes, we would want bugs created in the project that corresponds to their feature (UI, Database, Base Language, Conversion Tools, Runtime Infra). If the problem is UI code, even if it is a conversion issue, put it in the UI project.

> Document such source as uncompilable, so other can exclude them from conversion lists? Create a project-wide unconvertable file source list in repository for everyone's use?

I think we are moving to have a conversion list (or better a profile) that is specific to a given set of tests. We can exclude from there as needed.

#47 - 05/26/2023 08:10 AM - Greg Shah

> Greg, now that I think about it (and after spending lot of time with testcases) I would say we should have had tests separated based on the 'execution environment' as top level - server side business logic, character client, gui, application server, rest, web services, web handler... this way one can run only the tests pertaining for a specific customer environment and the 'test engine' could also be setup accordingly (chui/gui).

I want to ensure that we can run conversion for a given FWD server and include the code needed. However, I think it is confusing to have to search for all the test code for a given 4GL feature and have it split into many different possible top-level locations. You'll never know if you've found it all.

I also want to maximize the common code here. It seems to me that it is often reasonable to write the same test code that can be run in GUI, ChUI, batch... whatever and it can have conditional logic or preprocessing or whatever to detect the mode and do whatever differential processing is needed. The idea is that most of the code can be the same and only a small amount needs to be specialized.

Using the 4GL feature as the top level directory facilities both of these objectives.

> Like for security feature instead of dealing with multiple databases each with different security setup in a single 'test suite' that potentially test everything, we can have smaller test suites that are setup to run with different configurations to test specific a specific feature/use-case.

I agree that we need to determine the number of test servers/database and other runtime dependencies for each test set. Then we should build conversion profiles and runtime configurations to implement the necessary environments. The same 4GL feature will almost always be tested across multiple environments.

Files

TestErrorSending.cls	5.95 KB	11/17/2022	Vladimir Tsichevski