

Build and Source Control - Feature #1888

Feature # 1886 (Closed): implement a unit testing framework

integrate the framework into the build

10/31/2012 04:47 PM - Greg Shah

Status:	Closed	Start date:	
Priority:	Normal	Due date:	
Assignee:	Vladimir Tsichevski	% Done:	100%
Category:		Estimated time:	16.00 hours
Target version:		version:	
billable:	No		
vendor_id:	GCD		
Description			

History

#1 - 11/16/2016 01:16 PM - Greg Shah

- Target version deleted (24)

#2 - 03/04/2021 09:50 AM - Vladimir Tsichevski

- Status changed from New to WIP

- Assignee set to Vladimir Tsichevski

It is quite easy to create a Gradle-based project with unit-test for FWD.

The only thing necessary to enable test is to add a dependency on the unit-testing framework, for example:

```
dependencies {
    testImplementation 'junit:junit:4.13.1'
}
```

After this done, the tests can be run with:

```
gradle test
```

The complete build.gradle for a project may look like this:

```
apply plugin: 'java'
apply plugin: 'maven'

group = 'com.goldencode'
version = '0.0.1-SNAPSHOT'

description = ""FWD unit-testing""

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
    mavenCentral()
    mavenLocal()
}

dependencies {
    compile (group: 'com.goldencode', name: 'p2j', version:'4.0.fwd-SNAPSHOT') {
```

```
        exclude group: 'com.bea.xml', module: 'jsr173-ri'
    }
    testImplementation 'junit:junit:4.13.1'
}
```

Note: in this example FWD need to be build and installed in local Maven repository.

Note1: in the example project, I used the standard Maven-style source directory structure, so Gradle knows where to find tests already.

#3 - 03/04/2021 09:59 AM - Greg Shah

Note: in this example FWD need to be build and installed in local Maven repository.

We deliberately picked Gradle and avoided Maven. I don't want to add a dependency on a local Maven repo for testing.

Note1: in the example project, I used the standard Maven-style source directory structure, so Gradle knows where to find tests already.

This was another thing we deliberately avoided by choosing Gradle. I don't want to shift to a standard Maven source directory structure.

Is there an advantage to having a separate Gradle build for testing? I prefer to just add testing support to our existing Gradle build.

#4 - 03/04/2021 10:48 AM - Vladimir Tsichevski

The reason I did this in a project other than FWD so far is I failed to add tests into the FWD build process. The 'test' task is supported by the 'java' plugin, and FWD gradle build is based on ant, which seems to be mutual exclusive with 'java'.

#5 - 03/04/2021 10:50 AM - Vladimir Tsichevski

BTW is there a reason such a mix of ant and gradle was created instead of using gradle only?

#6 - 03/04/2021 10:59 AM - Hynek Cihlar

Vladimir Tsichevski wrote:

The reason I did this in a project other than FWD so far is I failed to add tests into the FWD build process. The 'test' task is supported by the

'java' plugin, and FWD gradle build is based on ant, which seems to be mutual exclusive with 'java'.

Create a simple Gradle java target and run ConsoleLauncher class from JUnit 5.

#7 - 03/04/2021 11:00 AM - Greg Shah

We started the project in 2004 with ant. We only shifted to Gradle in 2017. At that time we decided to add dependency processing in Gradle but to not rewrite all of the ant build since there was little advantage and lots of work needed.

We would like to have a full Gradle build, remove ant, but it just takes time.

#8 - 03/04/2021 11:01 AM - Hynek Cihlar

Vladimir Tsichevski wrote:

BTW is there a reason such a mix of ant and gradle was created instead of using gradle only?

This is a temporal state, we should eventually move the Ant stuff to Gradle, too.

#9 - 03/04/2021 11:10 AM - Vladimir Tsichevski

Hynek Cihlar wrote:

Create a simple Gradle java target and run ConsoleLauncher class from JUnit 5.

I am not fluent with Gradle, could you provide an example, please. Note also: the 'java' plugin is not available in FWD gradle build.

#10 - 03/04/2021 11:48 AM - Hynek Cihlar

Vladimir Tsichevski wrote:

Hynek Cihlar wrote:

Create a simple Gradle java target and run ConsoleLauncher class from JUnit 5.

I am not fluent with Gradle, could you provide an example, please. Note also: the 'java' plugin is not available in FWD gradle build.

Try something like the following,

```
task test(type: JavaExec) {  
    classpath = configurations.fwdAllRuntime  
    main = 'org.junit.platform.console.ConsoleLauncher'  
    args '<see the docs for ConsoleLauncher>'  
}
```

You will need to add dependency on JUnit 5 including junit-platform-console-standalone.

Also new Gradle configuration (extending fwdAllRuntime) should be added so that the JUnit (and its transitive dependencies) don't get deployed in production. Those JUnit dependencies should be added to this new configuration and the configuration used in the test task.

#11 - 03/12/2021 04:41 PM - Vladimir Tsichevski

Hynek Cihlar wrote:

You will need to add dependency on JUnit 5 including junit-platform-console-standalone.

Also new Gradle configuration (extending fwdAllRuntime) should be added so that the JUnit (and its transitive dependencies) don't get deployed in production. Those JUnit dependencies should be added to this new configuration and the configuration used in the test task.

Thanks, Hynek, it works for me now.

Now we shall decide on the test tree source location and structure. Any ideas?

#12 - 03/13/2021 11:22 AM - Greg Shah

Please put the test classes into the same package as the classes they test, but with a test base directory instead of src. This means that unit tests for src/com/goldencode/p2j/util/date.java would be in test/com/goldencode/p2j/util/date.java.

I know and accept that this different from the standard Maven layout, but we already deviate. I prefer to be a deviant. :)

#13 - 03/13/2021 12:08 PM - Vladimir Tsichevski

Greg Shah wrote:

I know and accept that this different from the standard Maven layout, but we already deviate. I prefer to be a deviant. :)

I prefer to use bicycles rather than invent them :)

#14 - 03/16/2021 10:35 AM - Vladimir Tsichevski

Greg Shah wrote:

Please put the test classes into the same package as the classes they test, but with a test base directory instead of src. This means that unit tests for `src/com/goldencode/p2j/util/date.java` would be in `test/com/goldencode/p2j/util/date.java`.

We will need additional classes with common support for testing. We can use something like `com.goldencode.p2j.tests` as the base package name for that classes, OK?

#15 - 03/16/2021 10:39 AM - Greg Shah

We can use something like `com.goldencode.p2j.tests` as the base package name for that classes, OK?

For any test helpers that have dependencies upon FWD, put them in `com.goldencode.p2j.testing`. For anything generic, please put the helpers in `com.goldencode.testing`.

I don't want to use the package name `tests` since this suggests that the classes are in fact tests themselves.

#16 - 03/16/2021 10:42 AM - Vladimir Tsichevski

Greg Shah wrote:

... For anything generic, please put the helpers in `com.goldencode.testing`.

By anything generic do you mean OE-related? For example, where the OE error numbers and messages shall go?

#17 - 03/16/2021 11:26 AM - Greg Shah

4GL (OE) related stuff is FWD-specific so that is NOT generic.

#18 - 03/16/2021 11:26 AM - Greg Shah

By generic, I mean that the code would be used in other projects outside of FWD.

#19 - 03/17/2021 09:56 AM - Vladimir Tsichevski

- % Done changed from 0 to 80

- Status changed from WIP to Review

JUnit-based testing support was added in 3821c rev. 12130.

Also 139 tests in 13 containers for parsing/applying various formats were added.

#20 - 03/17/2021 11:13 AM - Greg Shah

Code Review Task Branch 3821c Revision 12130

It is good. I really only see coding standards issues.

1. Please write abstract public class AbstractFWDTest as public abstract class AbstractFWDTest.
2. Imports should use *. This affects almost all of the files.
3. Please split extends <parent> onto its own line. This affects almost all of the files.
4. Data members should use the single line form of javadoc where possible.

```
/**
 * Default format string for DATETIME-TZ types.
 */
public static final String FULL_DATETIMETZ_FORMAT = "99/99/9999 HH:MM:SS.SSS+HH:MM";
```

should be this:

```
/** Default format string for DATETIME-TZ types. */
public static final String FULL_DATETIMETZ_FORMAT = "99/99/9999 HH:MM:SS.SSS+HH:MM";
```

5. Multi-line lambdas should follow the same curly braces rules as "regular code":

```
protected static final Function<String, ?> makeFormat = a -> {
    ...
};
```

would be:

```
protected static final Function<String, ?> makeFormat = a ->
{
    ...
};
```

Single line lambdas can be protected static final Function<String, ?> makeFormat = a -> { ... };

6. Comments inside of methods should use // instead of javadoc style.

```
public void testError()
{
```

```

/**
 * FIXME: This must throw an exception, but it currently does not.
 * FWD formats any value with empty format to an empty string.
 */
assertMustThrowError164(formatCharacter, "");
}

```

would be:

```

public void testError()
{
    // FIXME: This must throw an exception, but it currently does not.
    // FWD formats any value with empty format to an empty string.
    assertMustThrowError164(formatCharacter, "");
}

```

7. There are quite a few methods and some data members that are missing javadoc.

8. Some javadoc is formatted with extra blank lines or doubled comments.

```

/**
 * TODO: the DateFormat constructor must throw reasonable variants
 * of ErrorConditionException instead of DisplayFormatParsingException, which
 *
 * Test year is no more than 32768
 */
/**
 * TODO: fix the date.toString: it declares
 * IllegalArgumentException, but never throws it (and must not)!
 */

@BeforeClass
public static void beforeClass()

```

should be:

```

/**
 * TODO: the DateFormat constructor must throw reasonable variants
 * of ErrorConditionException instead of DisplayFormatParsingException, which
 * TODO: fix the date.toString: it declares
 * IllegalArgumentException, but never throws it (and must not)!
 *
 * Test year is no more than 32768
 */
@BeforeClass
public static void beforeClass()

```

#21 - 03/17/2021 02:30 PM - Vladimir Tsichevski

- % Done changed from 80 to 0

Greg Shah wrote:

Code Review Task Branch 3821c Revision 12130

It is good. I really only see coding standards issues.

...

Fixed in rev. 12135.

#22 - 03/17/2021 02:57 PM - Greg Shah

It looks good.

Please add javadoc for `AbstractDateTest.createDate()`, `TestDateFormat.beforeClass()`, `TestDateFormatParse.beforeClass()`, `TestDatetimeFormat.beforeClass()`.

Is this task done? Please set the % Done accordingly.

#23 - 03/17/2021 03:14 PM - Vladimir Tsichevski

- % Done changed from 0 to 100

Greg Shah wrote:

It looks good.

Please add javadoc for `AbstractDateTest.createDate()`, `TestDateFormat.beforeClass()`, `TestDateFormatParse.beforeClass()`, `TestDatetimeFormat.beforeClass()`.

Is this task done? Please set the % Done accordingly.

Done in rev. 12138. I do not know if there is anything left to do in this very task.

#24 - 03/17/2021 03:18 PM - Greg Shah

- Status changed from Review to Closed