

Testing Xtext Languages

Lorenzo Bettini

Dipartimento di Informatica, Università di Torino, Italy
itemis GmbH, Zurich, Switzerland

EclipseCon Europe 2015

Wrong Workflow

- Modify the language
- Start an Eclipse runtime instance
- Manually check that “everything” works in the editor

Wrong Workflow

- Modify the language
- Start an Eclipse runtime instance
- Manually check that “everything” works in the editor

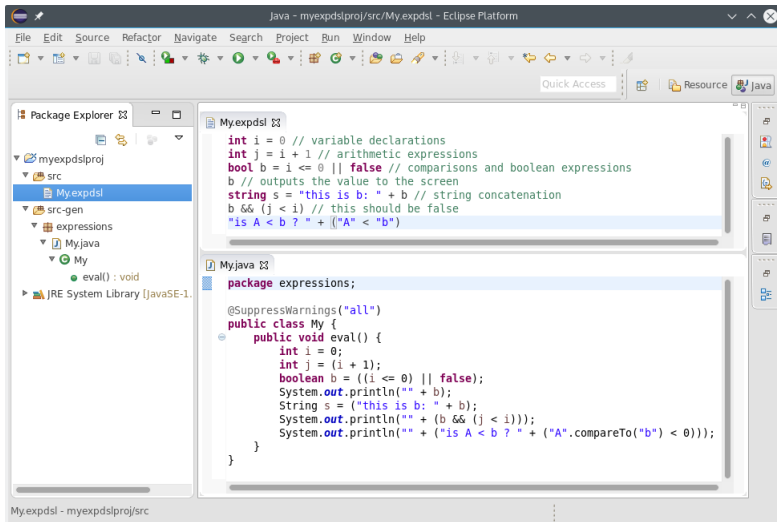
Write tests instead!

The example for this talk

<http://github.com/LorenzoBettini/ecefrance2015-xtext-example>

Example DSL: simple expressions

Write simple variables and expressions and generate Java code



The screenshot shows the Eclipse IDE with a project named 'myexpdslproj'. The Package Explorer on the left shows the project structure: 'myexpdslproj' contains 'src', which contains 'My.expdsl'. The 'My.expdsl' file is selected, and its content is displayed in the editor. The DSL code defines variables and expressions. Below the DSL editor, the 'Myjava' editor shows the generated Java code, which includes a package declaration, a class 'My', and a method 'eval()' that implements the DSL logic using Java constructs like 'int', 'boolean', 'String', and 'System.out.println'.

```
int i = 0 // variable declarations
int j = i + 1 // arithmetic expressions
bool b = i <= 0 || false // comparisons and boolean expressions
b // outputs the value to the screen
string s = "this is b: " + b // string concatenation
b && (j < i) // this should be false
"is A < b ? " + ("A" < "b")
```

```
package expressions;

@SuppressWarnings("all")
public class My {
    public void eval() {
        int i = 0;
        int j = (i + 1);
        boolean b = ((i <= 0) || false);
        System.out.println(" " + b);
        String s = ("this is b: " + b);
        System.out.println(" " + (b && (j < i)));
        System.out.println(" " + ("is A < b ? " + ("A".compareTo("b") < 0)));
    }
}
```

Initial Grammar

grammar org.eclipsecon.expdsl.Expressions **with** org.eclipse.xtext.common.Terminals

generate expressions "<http://www.eclipsecon.org/expdsl/Expressions>"

ExpressionsModel:

elements += AbstractElement*;

AbstractElement:

Variable | Expression ;

Variable:

declaredType=Type name=ID '=' expression=Expression;

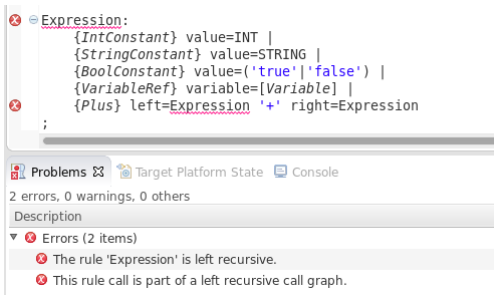
Type:

{IntType} 'int' |
{BoolType} 'bool';

Expression:

{IntConstant} value=INT |
{StringConstant} value=STRING |
{BoolConstant} value=('true'|'false') |
{VariableRef} variable=[Variable];

Initial Problems: “left recursion”

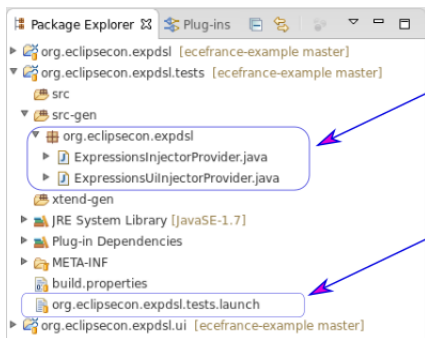


Solution

- Apply Left Factoring technique
- But let's start testing right away!
- To check that we get associativity/priority right

Projects Structure

- Xtext already generates a testing project
- With injector providers (headless and UI tests)

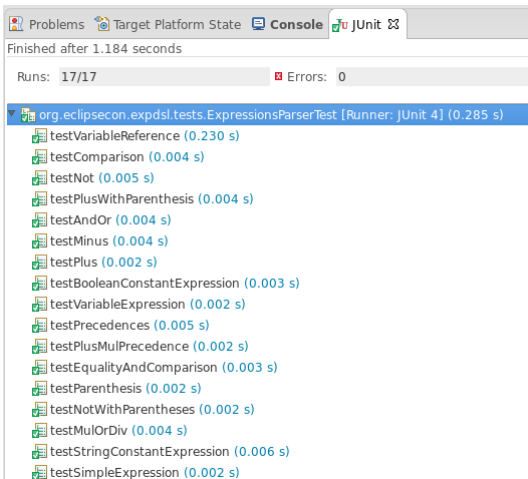


injector providers

launch configuration

Junit tests

- Most parts can be tested as Junit tests (no UI required)
- This is much faster!



Typical TestCase (Xtend)

- Run with `XtextRunner`
- `InjectWith` the generated injector provider
- `Inject` Xtext testing utility classes

```
package org.eclipsecon.expdsl.tests

import static extension org.junit.Assert.*

@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsInjectorProvider))
class ExpressionsParserTest {

    @Inject extension ParseHelper<ExpressionsModel>
    @Inject extension ValidationTestHelper

    @Test
    def void testSimpleExpression() {
        "10".parse.assertNoErrors
    }
}
```

Testing the Parser

- Tests that the program is parsed without syntax errors
- Test that the AST is created as expected:
 - expected associativity
 - expected operator precedence
- `ParseHelper`: parses a string and returns the AST

Testing the Parser

- Tests that the program is parsed without syntax errors
- Test that the AST is created as expected:
 - expected associativity
 - expected operator precedence
- `ParseHelper`: parses a string and returns the AST

Additional Benefits

- After parsing, the “real” language developer’s job starts:
- Validate the AST, generate code, etc.
- So it’s better that the AST is created the way you expect it

Testing the Parser

- Given the AST we create a string representation that highlights precedence/associativity
- Check that it is as expected

Testing the Parser

- Given the AST we create a string representation that highlights precedence/associativity
- Check that it is as expected

```
def String stringRepr(Expression e) {  
  switch (e) {  
    Plus: "(" + e.left.stringRepr + " + " + e.right.stringRepr + ")"  
    // other cases (omissis)  
    Not: "!((" + e.expression.stringRepr + ")"  
    IntConstant: "" + e.value  
  }  
}
```

Testing the Parser

```
@Test def void testPlusMulPrecedence() {
    "10 + 5 * 2 - 5 / 1".assertRepr("((10 + (5 * 2)) - (5 / 1))")
}
@Test def void testComparison() {
    "10 <= 5 < 2 > 5".assertReprNoValidation("(((10 <= 5) < 2) > 5)")
}
@Test def void testEqualityAndComparison() {
    "true == 5 <= 2".assertRepr("(true == (5 <= 2))")
}
@Test def void testAndOr() {
    "true || false && 1 < 0".assertRepr("(true || (false && (1 < 0)))")
}

def assertRepr(CharSequence input, CharSequence expected) {
    input.parse => [
        expected.assertEquals(
            (elements.last as Expression).stringRepr
        )
    ]
}
```

Validation

- We want to avoid forward variable references

```
int i = j // This should be an error!  
int j = 0
```

Validation

- We want to avoid forward variable references

```
int i = j // This should be an error!  
int j = 0
```

- Let's write a utility class, `ExpressionsModelUtils`, that
 - Given any element
 - returns the list of variables declared before that element
 - (we have to walk in the AST model)
- Let's test such class separately
- i.e., before writing the validator itself

Testing Forward References

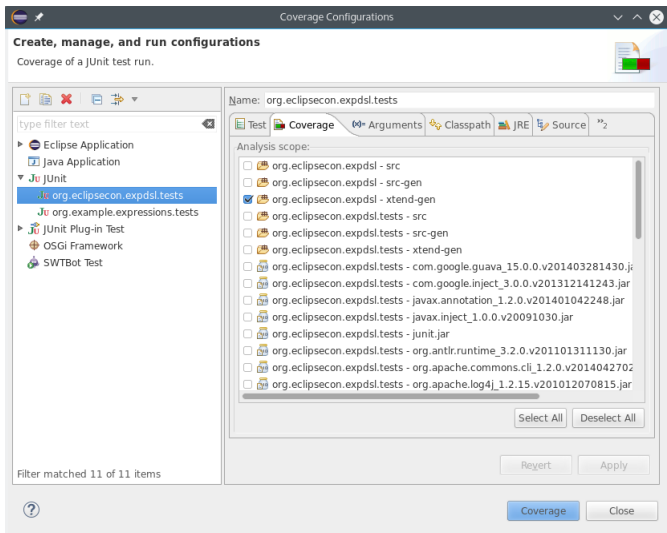
```
@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsInjectorProvider))
class ExpressionsModelUtilTest {

    @Inject extension ParseHelper<ExpressionsModel>
    @Inject extension ExpressionsModelUtil

    @Test def void variablesBeforeVariable() {
        ...
        true           // (0)
        int i = 0       // (1)
        i + 10         // (2)
        int j = i       // (3)
        i + j          // (4)
        ....parse => [
            assertVariablesDefinedBefore(0, "")
            assertVariablesDefinedBefore(1, "")
            assertVariablesDefinedBefore(2, "i")
            assertVariablesDefinedBefore(3, "i")
            assertVariablesDefinedBefore(4, "i,j")
        ]
    }

    def void assertVariablesDefinedBefore(ExpressionsModel model,
        int elemIndex, CharSequence expectedVars) {
        expectedVars.assertEquals(
            model.elements.get(elemIndex).variablesDefinedBefore.map[name].join(","))
    }
}
```

Let's “code coverage” right away (EclEmma/Jacoco)



Let's “code coverage” right away (Eclemma/Jacoco)

- Even if it's generated Java code (by Xtend)

The screenshot shows an IDE with three tabs: `ExpressionsModelUtil.java`, `ExpressionsModelUtil.xtend`, and `ExpressionsModelUtilTest.xtend`. The `ExpressionsModelUtil.java` file is open, displaying the following code:

```
@SuppressWarnings("all")
public class ExpressionsModelUtil {
    public List<Variable> variablesDefinedBefore(final AbstractElement e) {
        List<Variable> xblockexpression = null;
        {
            ExpressionsModel _containerOfType = EcoreUtil2.<ExpressionsModel>getContainingElement(e);
            final EList<AbstractElement> allElements = _containerOfType.getElements();
            final Function1<AbstractElement, Boolean> _function = new Function1<AbstractElement, Boolean>() {
                @Override
                public Boolean apply(final AbstractElement it) {
                    return Boolean.valueOf(EcoreUtil.isAncestor(it, e));
                }
            };
            final AbstractElement containingElement = IterableExtensions.<AbstractElement>indexOf(allElements, containingElement);
            int indexOf = allElements.indexOf(containingElement);
            List<AbstractElement> sublist = allElements.subList(0, indexOf);
            xblockexpression = EcoreUtil2.<Variable>typeSelect(_sublist, Variable.class);
        }
        return xblockexpression;
    }
}
```

Below the code editor, the `Coverage` tab is active, showing a table of coverage data for the test run `org.eclipse.secon.expdsl.tests (Jun 18, 2015 3:40:06 PM)`.

Element	Coverage	Covered Instructions	Missed Instructions
org.eclipse.secon.expdsl	100.0 %	60	0
xtend-gen	100.0 %	60	0
org.eclipse.secon.expdsl.scoping	100.0 %	3	0
org.eclipse.secon.expdsl.util	100.0 %	54	0
ExpressionsModelUtil.java	100.0 %	54	0
ExpressionsModelUtil	100.0 %	39	0
variablesDefinedBefore(AbstractElement)	100.0 %	36	0
org.eclipse.secon.expdsl.validation	100.0 %	3	0

Now we can write the Validator

```
class ExpressionsValidator extends AbstractExpressionsValidator {  
  
    public static val FORWARD_REFERENCE = "org.eclipsecon.expdsl.ForwardReference";  
  
    @Inject extension ExpressionsModelUtil  
  
    @Check  
    def void checkForwardReference(VariableRef varRef) {  
        val variable = varRef.getVariable()  
        if (!varRef.variablesDefinedBefore.contains(variable)) {  
            error("variable forward reference not allowed: "  
                + variable.name + "\"",  
                VARIABLE_REF_VARIABLE, // where to put the error marker  
                FORWARD_REFERENCE  
            )  
        }  
    }  
}
```

Testing the Validator

- And now we test that such error is correctly generated
 - On the expected element of the program
- We use the utility class `ValidationTestHelper`
- We can do that headlessly:
 - There's no need to run the workbench

Testing the Validator

- We use the utility class `ValidationTestHelper`

```
@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsInjectorProvider))
class ExpressionsValidatorTest {

    @Inject extension ParseHelper<ExpressionsModel>
    @Inject extension ValidationTestHelper

    @Test
    def void testForwardReference() {
        val input = '''
            int i = j
            int j = 10
        ...
        input.parse.assertError(
            // type of the element with error
            ExpressionsPackage.Literals.VARIABLE_REF,
            // error code
            ExpressionsValidator.FORWARD_REFERENCE,
            input.indexOf("j"), // offset of expected error
            1,                  // length of the region with error
            // expected error message
            "variable forward reference not allowed: 'j'"
        )
    }
}
```

Type System & Type Checking

- Check that expressions are **correct with respect to types**
 - subexpressions of a multiplication are integer
 - in a variable declaration, initialization expression can be assigned to the variable declared type
 - ...
- Implement the **type system** and test it
- Implement **type checking** in the validator, using the type system, and test it
- (not shown here: please have a look at the code)

Code Generation

- For each `expdsl` file we generate:
 - A Java class with the same name of the `expdsl` file in the package `expressions`
 - With an `eval()` method
 - For each variable declaration we generate a Java variable declaration
 - For each standalone expression we generate a `System.out.println`

Testing the Compiler

- Add the dependency `org.eclipse.xtext.xbase.junit`
- Add the dependency `org.eclipse.jdt.core`
- Use `CompilationTestHelper`
- Small pitfall: use a custom injector provider for missing Google Guice bindings expected by Xbase classes (see the code)

Testing the generated Java code is as expected

```
@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsInjectorProviderCustom))
class ExpressionsCompilerTest {

    @Rule @Inject public TemporaryFolder temporaryFolder
    @Inject extension CompilationTestHelper
    @Inject extension ReflectExtensions

    @Test def void testGeneratedJavaCode() {
        ...
        int i = 0
        int j = i + 1
        i > 0 || (j < i)
        ....assertCompilesTo(
        ...

package expressions;

@SuppressWarnings("all")
public class MyFile {
    public void eval() {
        int i = 0;
        int j = (i + 1);
        System.out.println("" + ((i > 0) || (j < i)));
    }
}

    ...
)
}
```

Testing the generated Java code is correct Java code

```
@Test def void testCorrectJavaCode() {  
    ...  
    int i = 0  
    int j = i + 1  
    bool b = i <= 0  
    b  
    string s = "this is b: " + b  
    b || (j < i)  
    s < "a"  
    ...compile[  
        // this will compile the generated Java code  
        compiledClass  
    ]  
}
```

- The test will fail if the generated Java code contains Java errors.

Testing the generated Java executes correctly

```
@Test def void testExecuteJavaCode() {  
    ...  
    int i = 0  
    int j = i + 1  
    bool b = i <= 0  
    b // this should be true  
    string s = "this is b: " + b  
    b && (j < i) // this should be false  
    "is A < b ? " + ("A" < "b")  
    ...  
    .compile[  
        val out = new ByteArrayOutputStream()  
        val backup = System.out  
        System.setOut(new PrintStream(out))  
        try {  
            // instantiate the compiled Java class  
            val obj = it.compiledClass.newInstance  
            obj.invoke('eval')  
        } finally {  
            System.setOut(backup)  
        }  
        ...  
        true  
        false  
        is A < b ? true  
        ...  
        .toString.assertEquals(out.toString)  
    }  
}
```

Testing UI

- The content assist
- The integration with the workbench
- The outline

Testing the Content Assist

- Add the dependency `org.eclipse.xtext.common.types.ui`
- Use as base class
`o.e.x.xbase.junit.ui.AbstractContentAssistTest`
- Inject your test case with `ExpressionsUiInjectorProvider`

Testing the Content Assist

- Add the dependency `org.eclipse.xtext.common.types.ui`
- Use as base class
`o.e.x.xbase.junit.ui.AbstractContentAssistTest`
- Inject your test case with `ExpressionsUiInjectorProvider`
- This must be run as a “Junit Plug-in Test”
- Set memory arguments in the launch configuration:
- `-Xms256m -Xmx1024m -XX:MaxPermSize=256m`

Testing the Content Assist

- Use as base class

`o.e.x.xbase.junit.ui.AbstractContentAssistTest`

Testing the Content Assist

- Use as base class

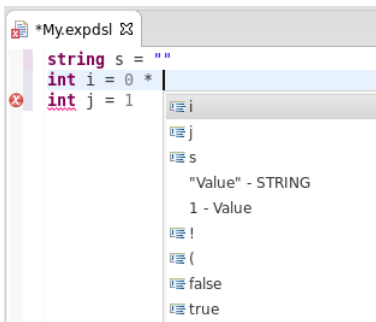
`o.e.x.xbase.junit.ui.AbstractContentAssistTest`

```
@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsUiInjectorProvider))
class ExpressionsContentAssistTest extends AbstractContentAssistTest {

    @Test
    def void testVariableReference() {
        newBuilder.append("int i = 10 1+").assertProposal('i')
    }

    @Test
    def void testProposals() {
        newBuilder.append("int i = 10 1+").
            assertText('!', "Value", '(', '+', '1', 'false', 'i', 'true')
    }
}
```

Tweaking the Content Assist



- Don't propose forward references (e.g., **i** and **j**)
- Don't propose references of the wrong type (e.g., **s**)

Testing the modified Content Assist

```
@Test
def void testForwardVariableReference() {
    // specify cursor position with <|>
    newBuilder.append("<|> int i = 10 ").assertNoProposalAtCursor("i")
    // i must not be present in proposals, before its definition
}
```

```
@Test
def void testForwardVariableReference2() {
    // specify the cursor character explicitly
    newBuilder.append("int k=0 int j=1 1+ int i = 10 ").
        //
        assertTextAtCursorPosition("+", 1,
            '!', "Value", '(', '+', '1', 'false', 'j', 'k', 'true')
    // i must not be present in proposals, before its definition
    // but j and k must be there
}
```

```
@Test
def void testProposeOnlyIntegerVariablesInMultiplication() {
    newBuilder.append("string s='a' int k=0 int j=1 1* ").
        //
        assertTextAtCursorPosition("*", 1,
            '!', "Value", '(', '*', '1', 'false', 'j', 'k', 'true')
    // s must not be present in proposals: it has the wrong type
}
```

Testing the Workbench Integration

- How is my DSL integrated with the workbench?
- Does a correct file lead to the generation of a Java file into `src-gen` folder?
- Does everything compile fine?
- We use
 - `o.e.x.junit4.ui.AbstractWorkbenchTest` as the base class
 - utility methods from `JavaProjectSetupUtil` and `IResourcesSetupUtil`
 - methods for accessing programmatically the workbench (from `o.e.core.resources`)

Testing the Workbench Integration

```
import static org.eclipse.xtext.junit4.ui.util.JavaProjectSetupUtil.*
import static extension org.eclipse.xtext.junit4.ui.util.IResourcesSetupUtil.*

class ExpressionsWorkbenchIntegrationTest extends AbstractWorkbenchTest {

    val TEST_PROJECT = "mytestproject"

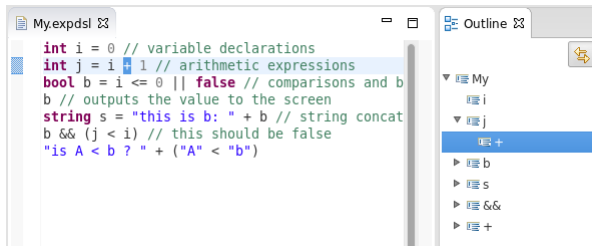
    @Before
    override void setUp() {
        super.setUp
        createJavaProject(TEST_PROJECT) => [
            project.addNature(XtextProjectHelper.NATURE_ID)
        ]
    }

    def void checkProgram(String contents, int expectedErrors) {
        createFile(TEST_PROJECT + "/src/test.expdsl", contents)
        waitForBuild();
        val markers = root.findMarkers(IMarker.PROBLEM, true, IResource.DEPTH_INFINITE)
        assertEquals(expectedErrors, markers.size)
    }

    @Test
    def void testValidProgram() {
        checkProgram("int i = 0", 0)
    }

    @Test
    def void testNotValidProgram() {
        checkProgram("foo", 1) // expect one error: unresolved variable reference
    }
}
```

Outline



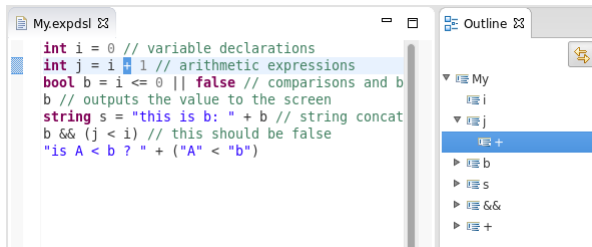
The screenshot shows the Eclipse IDE interface. On the left, the 'My.expdsl' editor displays the following code:

```
int i = 0 // variable declarations
int j = i + 1 // arithmetic expressions
bool b = i <= 0 || false // comparisons and b
b // outputs the value to the screen
string s = "this is b: " + b // string concat
b && (j < i) // this should be false
"is A < b ? " + ("A" < "b")
```

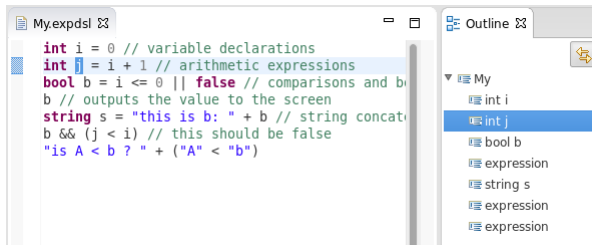
On the right, the 'Outline' view shows a hierarchical tree of the code elements:

- My
 - i
 - j
 - +
 - b
 - s
 - &&
 - +

Outline



Let's customize it like that



Testing the Outline

See the code for [AbstractOutlineWorkbenchTest](#)

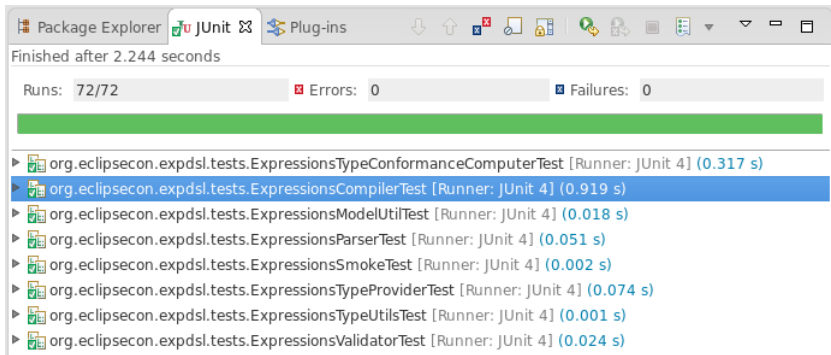
```
@RunWith(typeof(XtextRunner))
@InjectWith(typeof(ExpressionsUiInjectorProvider))
class ExpressionsOutlineTest extends AbstractOutlineWorkbenchTest {

    override protected getEditorId() {
        ExpressionsActivator.ORG_ECLIPSECON_EXPDSL_EXPRESSIONS
    }

    @Test
    def void testOutlineOfExpDslFile() {
        ...

        int i = 0
        string s = "a"
        s + i
        bool b = false
        ....assertAllLabels(
        ...
        test
        int i
        string s
        expression
        bool b
        ...
        )
    }
}
```

Required Time for Junit tests

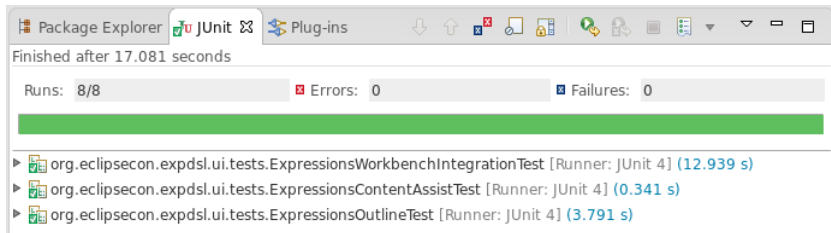


The screenshot shows the Eclipse IDE's Package Explorer and JUnit test results. The Package Explorer is on the left, showing the 'JUnit' package. The JUnit test results are displayed in the main area, showing a list of tests and their execution times. The tests are:

- org.eclipsecon.expdsl.tests.ExpressionsTypeConformanceComputerTest [Runner: JUnit 4] (0.317 s)
- org.eclipsecon.expdsl.tests.ExpressionsCompilerTest [Runner: JUnit 4] (0.919 s)
- org.eclipsecon.expdsl.tests.ExpressionsModelUtilTest [Runner: JUnit 4] (0.018 s)
- org.eclipsecon.expdsl.tests.ExpressionsParserTest [Runner: JUnit 4] (0.051 s)
- org.eclipsecon.expdsl.tests.ExpressionsSmokeTest [Runner: JUnit 4] (0.002 s)
- org.eclipsecon.expdsl.tests.ExpressionsTypeProviderTest [Runner: JUnit 4] (0.074 s)
- org.eclipsecon.expdsl.tests.ExpressionsTypeUtilsTest [Runner: JUnit 4] (0.001 s)
- org.eclipsecon.expdsl.tests.ExpressionsValidatorTest [Runner: JUnit 4] (0.024 s)

The test results are displayed in a table with columns for the test name, the runner, and the execution time. The test 'ExpressionsCompilerTest' is highlighted in blue. The overall status is 'Finished after 2.244 seconds'.

Required Time for JUnit Plug-in tests



The screenshot shows the Eclipse IDE's JUnit test runner window. The title bar includes 'Package Explorer', 'JUnit', and 'Plug-ins'. The main area displays 'Finished after 17.081 seconds'. Below this, a summary bar shows 'Runs: 8/8', 'Errors: 0', and 'Failures: 0'. A green progress bar is visible. The test results list shows three tests, all passing:

- ▶ org.eclipsecon.expdsl.ui.tests.ExpressionsWorkbenchIntegrationTest [Runner: JUnit 4] (12.939 s)
- ▶ org.eclipsecon.expdsl.ui.tests.ExpressionsContentAssistTest [Runner: JUnit 4] (0.341 s)
- ▶ org.eclipsecon.expdsl.ui.tests.ExpressionsOutlineTest [Runner: JUnit 4] (3.791 s)

Headless Build - No UI

```
<plugin>
  <groupId>org.eclipse.tycho</groupId>
  <artifactId>tycho-surefire-plugin</artifactId>
  <version>${tycho-version}</version>
  <executions>
    <execution>
      <!-- no UI -->
      <id>default-test</id>
      <phase>integration-test</phase>
      <configuration>
        <useUIHarness>false</useUIHarness>
        <useUIThread>false</useUIThread>
        <!-- tycho.testArgLine repeated to re-use the configuration for argLine
              for jacoco agent -->
        <argLine>${tycho.testArgLine} ${memoryArgs}</argLine>
        <includes>
          <include>**/expdsl/tests/*Test.java</include>
        </includes>
      </configuration>
      <goals>
        <goal>test</goal>
      </goals>
    </execution>
```

Headless Build - No UI

```
<plugin>
  <groupId>org.eclipse.tycho</groupId>
  <artifactId>tycho-surefire-plugin</artifactId>
  <version>${tycho-version}</version>
  <executions>
    <execution>
      <!-- no UI -->
      <id>default-test</id>
      <phase>integration-test</phase>
      <configuration>
        <useUIHarness>false</useUIHarness>
        <useUIThread>false</useUIThread>
        <!-- tycho.testArgLine repeated to re-use the configuration for argLine
              for jacoco agent -->
        <argLine>${tycho.testArgLine} ${memoryArgs}</argLine>
        <includes>
          <include>**/expdsl/tests/*Test.java</include>
        </includes>
      </configuration>
      <goals>
        <goal>test</goal>
      </goals>
    </execution>
```

Or use [maven-surefire-plugin](#) if you experience problems with the classloader.

Headless Build - UI

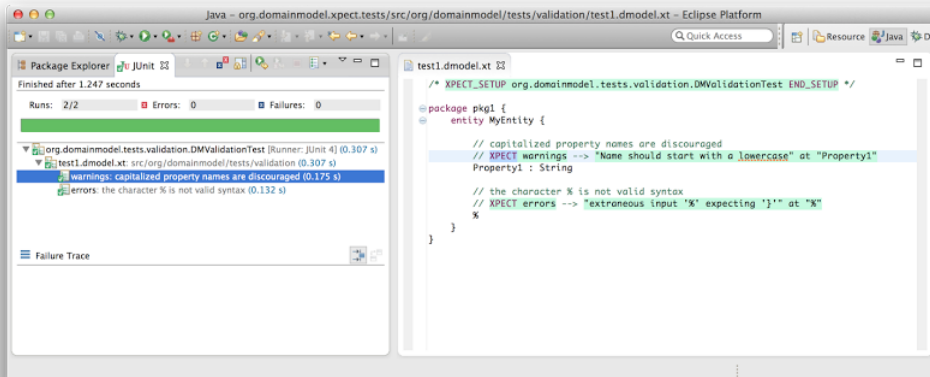
```
<execution>
  <id>TychoSurefirePluginUITest</id>
  <phase>integration-test</phase>
  <configuration>
    <useUIHarness>true</useUIHarness>
    <useUIThread>true</useUIThread>
    <argLine>${tycho.testArgLine} ${memoryArgs}</argLine>
    <includes>
      <include>**/expdsl/ui/tests/*Test.java</include>
    </includes>
  </configuration>
  <goals>
    <goal>test</goal>
  </goals>
</execution>
```

Testing other aspects

- If some aspects are not covered by Xtext testing framework
- you can have a look at Xtext sources
- and get some inspiration...
 - This is what I did for the outline tests

Other Testing Frameworks

- Xpect: <http://www.xpect-tests.org>
- SWTBot, RCP Testing Tool, Jubula for functional tests



Use CI, e.g., Jenkins

search

Lorenzo Bettini | log out

Jenkins > Xtext > xtext-ecefrance-example-gerrit-tycho >

ENABLE AUTO REFRESH

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Maven project

Configure

Modules

Coverage Trend

Build History

#17

Jun 20, 2015 4:18 PM

2916.1

#16

Jun 20, 2015 2:25 PM

2915.1

#15

Jun 20, 2015 2:17 PM

2914.1

#14

Jun 20, 2015 2:09 PM

2913.1

#13

Jun 19, 2015 7:31 PM

2912.1

#12

Jun 19, 2015 7:26 PM

2911.2

#11

Jun 19, 2015 7:22 PM

2911.1

#10

Jun 19, 2015 6:59 PM

2910.1

#9

Jun 19, 2015 6:34 PM

2909.2

#8

Jun 19, 2015 6:28 PM

2909.1

Maven project xtext-ecefrance-example-gerrit-tycho

add description

Disable Project

Workspace

Recent Changes

Latest Test Result (no failures)

Latest Test Result (no failures)

Latest Aggregated Test Result (no tests)

Permalinks

- Last build (#17), 1 day 18 hr ago
- Last stable build (#17), 1 day 18 hr ago
- Last successful build (#17), 1 day 18 hr ago
- Last unstable build (#8), 2 days 15 hr ago
- Last unsuccessful build (#8), 2 days 15 hr ago

Test Result Trend

count

#1 #2 #3 #4 #5 #6 #7 #8 #9 #10 #11 #12 #13 #14 #15 #16 #17

[\(just show failures\)](#) [enlarge](#)

Code Coverage Trend

lineCovered

lineMissed

#1 #3 #5 #7 #9 #11 #13 #15 #17

[enlarge](#)

Lorenzo Bettini

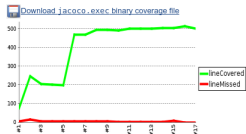
Testing Xtext Languages

EclipseCon Europe 2015

40/43

Use CI, e.g., Jenkins

JaCoCo Coverage Report



Overall Coverage Summary

name	instruction	branch	complexity	line	method
all classes	100% M: 0 C: 1841	100% M: 0 C: 78	100% M: 0 C: 103	100% M: 0 C: 498	100% M: 0 C: 64

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method
org.eclipse.secdex.expdsl.expressions	M: 0 C: 160 100%	M: 0 C: 0 0%	M: 0 C: 3 100%	M: 0 C: 54 100%	M: 0 C: 3 100%
org.eclipse.secdex.expdsl.generator	M: 0 C: 898 100%	M: 0 C: 30 100%	M: 0 C: 36 100%	M: 0 C: 253 100%	M: 0 C: 21 100%
org.eclipse.secdex.expdsl.scoping	M: 0 C: 3 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%
org.eclipse.secdex.expdsl.typing	M: 0 C: 517 100%	M: 0 C: 36 100%	M: 0 C: 39 100%	M: 0 C: 126 100%	M: 0 C: 21 100%
org.eclipse.secdex.expdsl.ui.contentassist	M: 0 C: 92 100%	M: 0 C: 4 100%	M: 0 C: 6 100%	M: 0 C: 19 100%	M: 0 C: 4 100%
org.eclipse.secdex.expdsl.ui.labeling	M: 0 C: 38 100%	M: 0 C: 0 0%	M: 0 C: 4 100%	M: 0 C: 9 100%	M: 0 C: 4 100%
org.eclipse.secdex.expdsl.ui.outline	M: 0 C: 7 100%	M: 0 C: 0 0%	M: 0 C: 3 100%	M: 0 C: 3 100%	M: 0 C: 3 100%
org.eclipse.secdex.expdsl.ui.quickfix	M: 0 C: 3 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%
org.eclipse.secdex.expdsl.util	M: 0 C: 54 100%	M: 0 C: 0 0%	M: 0 C: 4 100%	M: 0 C: 12 100%	M: 0 C: 4 100%
org.eclipse.secdex.expdsl.validation	M: 0 C: 69 100%	M: 0 C: 8 100%	M: 0 C: 6 100%	M: 0 C: 20 100%	M: 0 C: 2 100%

...or Travis

```
.travis.yml
sudo: false
language: java
jdk: oraclejdk7
cache:
  directories:
    - $HOME/.m2
env: DISPLAY=:99.0
install: true
before_script:
  - sh -e /etc/init.d/xvfb start
script:
  - export
  - mvn -f org.eclipsecon.expsdsl.parent/pom.xml clean verify
```

Small Advertising O:-)

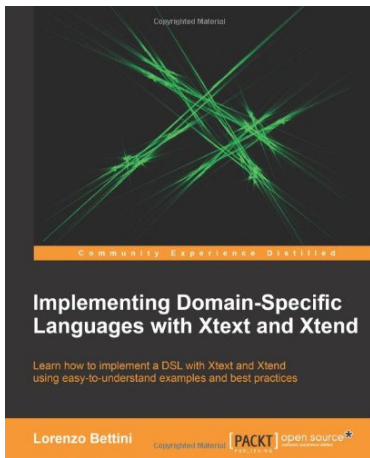
<http://bit.ly/ids50>



- An entire chapter on testing

Small Advertising O:-)

<http://bit.ly/ids50>



- An entire chapter on testing

Discount codes:

- IDS50 - 50% ebook
- IDS20 - 20% printed book

Small Advertising O:-)

<http://bit.ly/ids50>



- An entire chapter on testing

Discount codes:

- IDS50 - 50% ebook
- IDS20 - 20% printed book

Please evaluate this session



Small Advertising O:-)

<http://bit.ly/ids50>



- An entire chapter on testing

Discount codes:

- IDS50 - 50% ebook
- IDS20 - 20% printed book

Please evaluate this session



Thanks! and Questions?