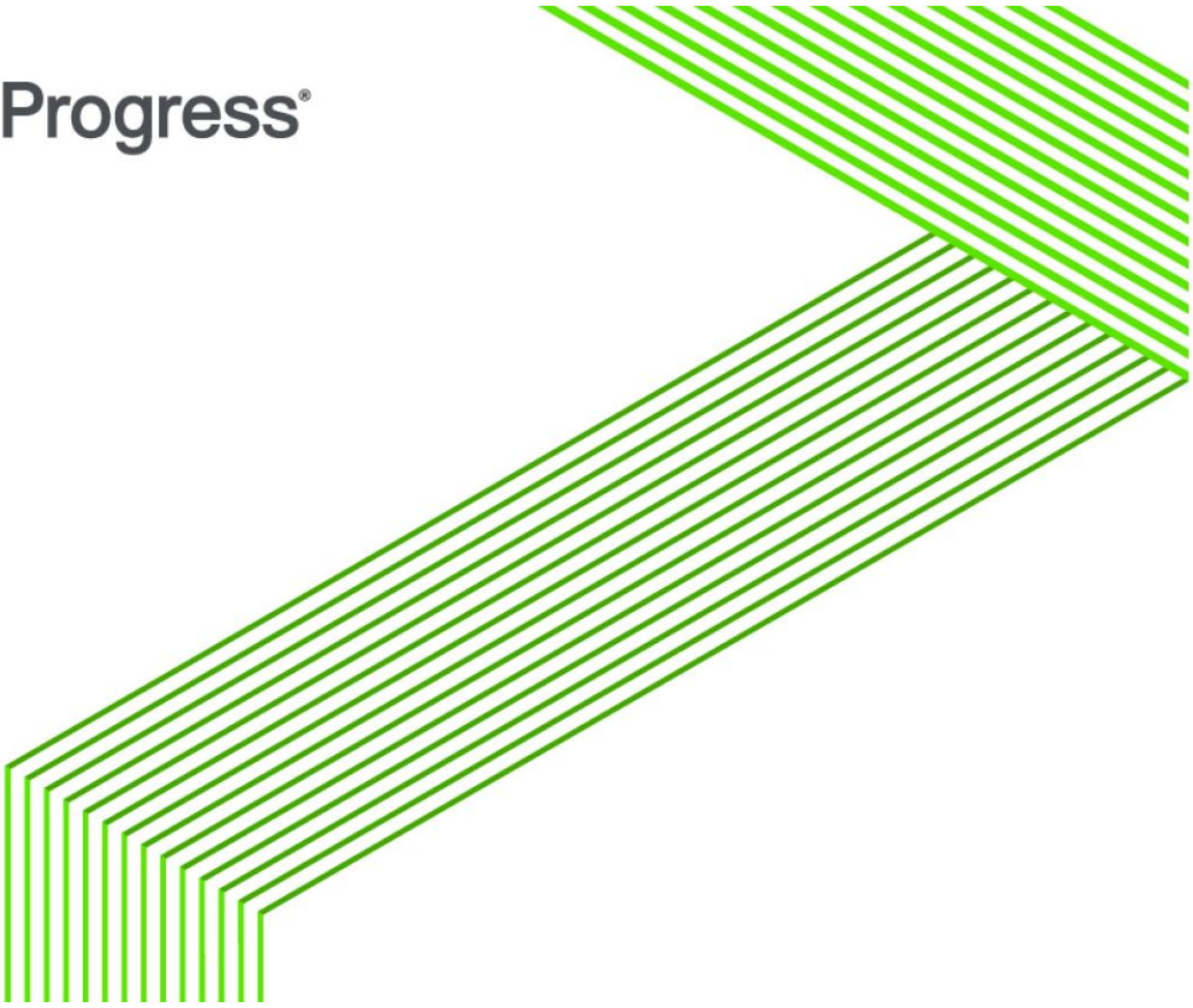

Migrating Tests from OEUnit to ABLUnit

White Paper

Progress OpenEdge

Notices

October 2016

Contents

1 Introduction

ABLUnit and OEUnit are frameworks that you can use for writing repeatable tests for ABL sources. Progress Developer Studio for OpenEdge (PDS for OpenEdge) 11.4 introduced ABLUnit, a unit testing framework based on the xUnit framework for writing repeatable tests for ABL sources. This feature was then enhanced in OpenEdge 11.5 and 11.6.
Before the introduction of ABLUnit, OEUnit was a very popular framework for writing repeatable tests in the OpenEdge community. ABLUnit provides a rich set of APIs along with superior tooling support for writing tests, running those tests, and analyzing the results. On the other hand, OEUnit, the predecessor capability, also has a good feature set including parameterized tests, as well as a set of assert APIs. Both frameworks work with OEMock, the mocking framework.

This document compares and contrasts the features between OEUnit and ABLUnit, and discusses in detail how to migrate from OEUnit to ABLUnit. It also discusses the features present in OEUnit and the equivalent features present in ABLUnit if they exist, or equivalent capability that be derived from other constructs. This document can be used by OEUnit users to start adopting the ABLUnit framework. You can use the equivalent codes in ABLUnit to migrate from OEUnit to the ABLUnit framework.

See the table below for a high-level comparison:

Table 1: Comparison of ABLUnit and OEUnit

Description	**ABLUnit**	**OEUnit**
Supported test entities	Classes and Procedures	Classes
Tooling	PDS FOR OpenEdge Tooling (Project, Wizards, Templates, Results View and so on)	ABL Window that can be integrated with PDS for OpenEdge
Ant Task	Yes	From PCT
Command Line Tool	Yes	No
No of assertions, including overloaded	~ 156	~ 180
Support for mocking framework	Can work with OEMock	Can work with OEMock

OEUnit and ABLUnit test cases look similar in most of the cases as seen in Figures 1 and 2. Figure 1 shows a test class, which has a single test case written in OEUnit. Similarly, Figure 2 displays an equivalent of that in ABLUnit.

```
USING OEUnit.Assertion.Assert.

BLOCK-LEVEL ON ERROR UNDO, THROW.

CLASS OEUnitTest:

    @Test.
    METHOD PUBLIC VOID TestPass():
        Assert:IsTrue(TRUE).
    END METHOD.

END CLASS.
```

Figure 1: OEUnit Test Class

```
USING OpenEdge.Core.Assert.

BLOCK-LEVEL ON ERROR UNDO, THROW.

CLASS ABLUnitTest:

    @Test.
    METHOD PUBLIC VOID TestPass():
        Assert:IsTrue(TRUE).
    END METHOD.

END CLASS.
```

Figure 2: ABLUnit Test Class

2 Migrating the Source Code

When moving from OEUnit to ABLUnit, there are either direct feature equivalents or the ability to construct feature equivalents in ABLUnit. These are described in this section.

2.1 Migrating the Tests

This section covers how to migrate different test entity sources including test cases, test classes and test suites.

Test Case

A class method is identified as a test case if it utilizes the @Test annotation. This is the same for the OEUnit and ABLUnit frameworks. The sample code in Snippet 1 shows a test case:

Snippet 1: Test Case

```
@Test.
METHOD PUBLIC VOID TestPass():
    Assert:IsTrue(TRUE).
END METHOD.
```

Sometimes, when you call an API method with a particular input, it might throw an exception at runtime. Both the OEUnit and ABLUnit frameworks support testing the expected exception at runtime, and the expected exception is displayed the same way in both the frameworks. The snippet below is an example of a test in the ABLUnit framework that expects an exception, OpenEdge.Core.AssertionFailedError, to be thrown at runtime:

Snippet 2: Test Case with an Expected Error

```
@Test(expected="OpenEdge.Core.AssertionFailedError").
METHOD PUBLIC VOID TestPass():
    Assert:IsTrue(FALSE).
END METHOD.
```

Test Suite

A test suite is a collection of test cases and/or other test suites. A test class is a test suite which contains one or more test cases. The snippet below is an example of a test suite created in OEUnit from two test classes **TestClassPass** and **TestClassFail**:

Snippet 3: Test Suite Class in OEUnit

```
CLASS SampleSuite INHERITS TestSuite:

  CONSTRUCTOR PUBLIC SampleSuite (  ):

        AddTest(NEW TestClassPass()).
        AddTest(NEW TestClassFail()).

  END CONSTRUCTOR.

END CLASS.
```

In ABLUnit, a test suite class is equivalent to the above example but simpler in representation as shown in the following code snippet:

Snippet 4: Test Suite Class in ABLUnit

```
@TestSuite(classes="TestClassPass,TestClassFail").
CLASS SampleSuite:

END CLASS.
```

Lifecycle Methods

Before you execute a test, it might have to meet some prerequisites. For example, to setup configuration and data objects that will be used by your test, you might need to clean up the resources that are created in the setup. Both frameworks support configuring specific methods for such purposes. The table below describes the annotations that can be configured for specific methods in both the frameworks:

Table 2: Lifecycle Annotations

Purpose	OEUnit	ABLUnit
Execute a method before executing each test.	@Before	@Setup
Execute a method after executing each test.	@After	@TearDown

Purpose	OEUnit	ABLUnit
Execute a method as a first method before the start of any test in a class.	@BeforeClass	@Before
Execute a method as a last method after finishing the execution of all the tests in a class.	@AfterClass	@After

@Ignore

In OEUnit, if you want to temporarily disable a particular test case or a test suite, you annotate the methods with *@Ignore*- these tests will not be executed during testing. The ABLUnit framework currently does not provide a similar annotation to ignore a test- the equivalent is to comment out the @Test annotations that you wish to ignore.

DataProvider

You may want to write a series of tests which differ only in inputs and expected values. Parametrized tests simplify this by allowing you to write a single test and run the same test against different sets of values. In OEUnit, DataProvider is designed for this purpose. The code snippet below declares a method as data provider:

Snippet 5: Declaration of a method as a DataProvider

```
@DataProvider.
  METHOD PUBLIC DataProvider StatusProvider(  ):

        DEFINE VARIABLE dataProvider AS DataProvider NO-UNDO.
        dataProvider = NEW DataProvider().
        dataProvider:FromJSON("~{~"status~": ~"Success~"~}").
        RETURN dataProvider.

  END METHOD.
```

The single test case displayed below is executed repeatedly based on the DataProvider configured data. For each of the items in the JSON data provider, the test 'ProviderTest' is executed by passing the parameters from the JSON data:

Snippet 6: Test case consumes the configured DataProvider

```
@Test(dataProvider="StatusProvider").
  METHOD PUBLIC VOID ProviderTest( INPUT vStatus AS CHARACTER ):

        Assert:AreEqual("Success", vStatus).

  END METHOD.
```

There is no equivalent for this in ABLUnit currently: you must write individual tests with different inputs that are present in the data provider.

2.2 AssertionFailedError

Most unit testing frameworks are designed to throw an exception during the invocation of an assertion API when the expected value does not match the actual value and is considered to be a failure. In OEUnit, the assertion failed exception class is *OEUnit.Assertion.AssertionFailedError.* The equivalent class in ABLUnit is *OpenEdge.Core.AssertionFailedError*. When a test fails, an *AssertionFailedError* is raised with a detailed exception stack trace.

2.3 Assert API Migration

This section covers how to migrate the assertion APIs. An assertion is a method that verifies the behavior of a unit under test. It is a logical condition that is true for the results that are expected at runtime. The failure of an assertion typically results in an exception being thrown and the execution of the current test aborts.

Assertion APIs are a set of methods to test the expected values at runtime for different data types including built-in and user-defined types.

Assertions in OEUnit are part of the Assert and AssertString classes in the OpenEdge.Assertion package. In ABLUnit, the assertion APIs are part of the Assert class in the OpenEdge.Core package and AssertError, AssertFile, AssertJson, and AssertObject classes in the OpenEdge.Core.Assertion package.

> Note: For the rest of this section, package names in the respective frameworks are omitted for conciseness.

The overloaded versions of APIs, where the additional parameter is a CHARACTER, is different in both frameworks. In OEUnit, it is the complete error message and in ABLUnit it is the name of the parameter from which the error message is constructed. For example, both frameworks allow you to check that an object is not null. In OEUnit, if the second parameter is a string ("The given objects are not equal"), the below API call raises the assertion failed error with the given message:

Snippet 7: Assert API from OEUnit with addition CHARACTER parameter

```
Assert:IsNotNull(anObject, "The given object is null").
```

In ABLUnit, the overloaded version of the APIs assume that the additional parameter is the name of the data type that will be used to form an error message. The API example given below raises an error with the constructed message "Input object cannot be null".

Snippet 8: Assert API from ABLUnit with addition CHARACTER parameter

```
Assert:IsNotNull(anObject, "Input object").
```

The table below discusses different APIs based on their purpose in both frameworks. If there is no direct equivalent, equivalent that can be easily constructed from other APIs are provided. The table below does not discuss the overloaded versions of the APIs:

Table 3: Assert APIs in OEUnit and ABLUnit

Purpose	OEUnit	ABLUnit
Assert that two entities are equal	Assert: AreEqual(Progress.Lang.Object, Progress.Lang.Object)	AssertObject: Equals(Progress.Lang.Object, Progress.Lang.Object)
	Similarly, there are overloaded versions of APIs for the CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INT64, INTEGER, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, ROWID ABL data types.	Assert: Equals(<DataType>, <DataType>) Assert class contains the API shown above for the CHARACTER, DECIMAL, HANDLE, INT64, INTEGER, LONGCHAR, RECID, ROWID ABL data types. There are no direct APIs for the ABL data types below but you can construct them. See **Snippet 9** for a DATE type example. COM-HANDLE, DATE, DATETIME, DATETIME-TZ, LOGICAL, MEMPTR, RAW
Assert that two entities are not equal	Assert: AreNotEqual(Progress.Lang.Object, Progress.Lang.Object)	AssertObject:NotEqual (Progress.Lang.Object, Progress.Lang.Object)
	Similarly, overloaded versions of APIs for the following ABL data types. CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INT64, INTEGER, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, ROWID	Assert: NotEqual(<DataType>, <DataType>) Assert class contains the API shown above for the CHARACTER, DECIMAL, HANDLE, INT64, INTEGER, LONGCHAR, RECID, ROWID ABL data types There are no direct APIs for the ABL data types below but

Purpose	OEUnit	ABLUnit
		you can construct them. See **Snippet 10** for a DATETIME-T type example. COM-HANDLE, DATE, DATETIME, DATETIME-TZ, LOGICAL, MEMPTR, RAW
Assert that two objects refer to the same instance	Assert: AreSame(Progress.Lang.Object, Progress.Lang.Object)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 11**.
Assert that two objects do not refer to the same instance	Assert: AreNotSame(Progress.Lang.Object, Progress.Lang.Object)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 12**.
Assert that the given object reference is null	Assert: IsNull(Progress.Lang.Object)	AssertObject: IsNull(Progress.Lang.Object)
Assert that the given object reference is not null	Assert: IsNotNull(Progress.Lang.Object)	AssertObject: NotNull(Progress.Lang.Object)
Assert that the condition is true	Assert:IsTrue(LOGICAL)	Assert:IsTrue(LOGICAL)
Assert that the condition is false	Assert:IsFalse(LOGICAL)	Assert:IsFalse(LOGICAL)
Fails the test/Raise an error	Assert:Fail() Assert:Fail (CHARACTER)	Assert: RaiseError(CHARACTER)
Assert that the given entity is null or empty	AssertString: IsNullOrEmpty (CHARACTER)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 13**.
	AssertString: IsNullOrEmpty (LONGCHAR)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 14**.

Purpose	OEUnit	ABLUnit
Assert that the given entity is not null or empty	AssertString: IsNotNullOrEmpty (CHARACTER)	Assert: NotNullOrEmpty (CHARACTER)
	AssertString: IsNotNullOrEmpty (LONGCHAR)	Assert: NotNullOrEmpty (LONGCHAR)
Assert that the given entity contains a character string	AssertString: Contains (CHARACTER, CHARACTER)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 15**.
	AssertString: Contains (LONGCHAR, CHARACTER)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 16**.
Assert that the given entity does not contain a character string	AssertString: DoesNotContain(CHARACTER, CHARACTER)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 17**.
	AssertString: DoesNotContain (LONGCHAR, CHARACTER)	There are no direct APIs for the ABL data types but you can construct them. See **Snippet 18**.

There are many alternatives to construct the equivalent to the missing OEABL APIs in ABLUnit. The snippets below list one such possible snippet code for some of the missing APIs:

Snippet 9

If date1 and date2 are the two DATE objects, the code snippet below demonstrates comparing them for equality:

Snippet 9: Code Snippet for DATEs equality constructed in ABLUnit

```
Assert:NotNull(date1).
Assert:NotNull(date2).
Assert:IsTrue(date1 EQ date2).
```

Snippet 10

If dateTime1 and dateTime2 are the two DATETIME-TZ objects, the code snippet below demonstrates comparing them for equality:

Snippet 10: Code Snippet for DATETIME-TZ s equality constructed in ABLUnit

```
Assert:NotNull(dateTime1).
Assert:NotNull(dateTime2).
Assert:IsTrue(dateTime1 NE dateTime2).
```

Snippet 11

If object1 and object2 are Progress.Lang.Objects, the code snippet below checks if they reference the same object in memory:

Snippet 11: Code Snippet for Progress.Lang.Objects reference equal constructed in ABLUnit

```
Assert:NotNull(object1).
Assert:NotNull(object2).
Assert:IsTrue(object1 EQ object2).
```

Snippet 12

If object1 and object2 are Progress.Lang.Objects, the code snippet below checks if they do not reference the same object in memory:

Snippet 12: Code Snippet for Progress.Lang.Objects reference equal constructed in ABLUnit

```
Assert:NotNull(object1).
Assert:NotNull(object2).
Assert:IsTrue(object1 NE object2).
```

Snippet 13

For a given CHARACTER data, ‘charData’, the code snippet below checks if it is either a null or an empty string:

Snippet 13: Code Snippet for Checking a CHARACTER for null or empty

```
Assert:IsTrue(charData = ? OR charData = "").
```

Snippet 14

For a given LONGCHAR data, ‘lcharData’, the code snippet below checks if it is either a null or an empty string.

Snippet 14: Code Snippet for Checking a LONGCHAR for null or empty

```
Assert:IsTrue(lcharData = ? OR lcharData = "").
```

Snippet 15

For a given CHARACTERs, 'string' and 'charSeq', the code snippet below checks if 'string' contains 'charSeq'.

Snippet 15: Code Snippet for Checking a character sequence in another character sequence

```
Assert:NotNull(string).
Assert:NotNull(charSeq).
Assert:IsTrue(INDEX(string, charSeq) > 0).
```

Snippet 16

For a given LONGCHAR, 'lstring', the code snippet below checks if 'lstring' contains 'charSeq'.

Snippet 16: Code Snippet for Checking a character sequence in another long character sequence

```
Assert:NotNull(lstring).
Assert:NotNull(charSeq).
Assert:IsTrue(INDEX(lstring, charSeq) > 0).
```

Snippet 17

For a given CHARACTERs, 'string' and 'charSeq', the code snippet below checks if 'string' does not contain 'charSeq'.

Snippet 17: Code Snippet for Checking a character sequence in another character sequence

```
Assert:NotNull(string).
Assert:NotNull(charSeq).
Assert:IsTrue(INDEX(string, charSeq) EQ 0).
```

Snippet 18

For a given LONGCHAR, 'lstring', the code snippet below checks if 'lstring' does not contain 'charSeq'.

Snippet 18: Code Snippet for Checking a character sequence in another long character sequence

```
Assert:NotNull(lstring).
Assert:NotNull(charSeq).
Assert:IsTrue(INDEX(lstring, charSeq) EQ 0).
```

3 Migrating a Build

Support for a feature as part of the build process is an added advantage in any framework. It can also be used in the nightly build that takes place automatically. Both OEUnit and ABLUnit frameworks provide an ANT task for running tests as part of the build process.

This document does not discuss the configuration of OEUnit and ABLUnit tasks. An OEUnit Ant task is part of Progress Compilation Tools (PCT, located at https://github.com/jakejustus/pct). For more details on ANT tasks, see https://github.com/jakejustus/pct/wiki/OEUnit.

The snippet below is a sample Ant task in the OEUnit framework:

Snippet 19: Build task for OEUnit

```
<target name="main" depends="...">
    <OEUnit destDir="C:/ABLUnit/test-reports" format="junit">
      <fileset dir="src" includes="**/*.cls" />
      <propath>
            <pathelement path="C:/ABLUnit/src/project1" />
            <pathelement path="C:/ABLUnit/src/project2" />
      </propath>
    </OEUnit>
</target>
```

The build task for ABLUnit is similar to OEUnit. The snippet below is a sample Ant task in ABLUnit.

Snippet 20: Build task for ABLUnit

```
<target name="main" depends="...">
   <ABLUnit dlc="${dlc.dir}">
      <propath>
            <pathelement location="C:/ABLUnit/src/project1" />
            <pathelement location="C:/ABLUnit/src/project2" />
      </propath>

       <batchtest todir="C:/ABLUnit/test-reports" format="xml">
           <fileset dir="C:/ABLUnit/src/" includes="**/*.cls" />
       </batchtest>
   </ABLUnit>
</target>
```

ABLUnit also supports command line tooling to run ABLUnit tests through proenv. For more details, see the product documentation.

4 Using OEMock in ABLUnit

In the above sections, we have seen how to migrate both tests and build scripts from OEUnit to ABLUnit. The next big question is what if your tests are dependent on OEMock. This section addresses how to continue to use OEMock after migrating to ABLUnit, as ABLUnit does not currently have a mocking framework.

This section discusses the necessary steps for making the OEMock framework available to the ABLUnit project. It focuses on using Progress Developer Studio for OpenEdge, as it is out of scope of this document to show you how to use proenv to execute the ABLUnit tests along with OEMock.

1. Download and extract the OEMock framework (https://github.com/msabbott/OEMock) to a directory, for example, C:\ABLUnit\OEMock-master.
2. Create an OpenEdge project of the ABLUnit type and add the OEMock framework to its PROPATH. The figure below shows the contents of PROPATH after adding the OEMock framework to the ABLUnit project:

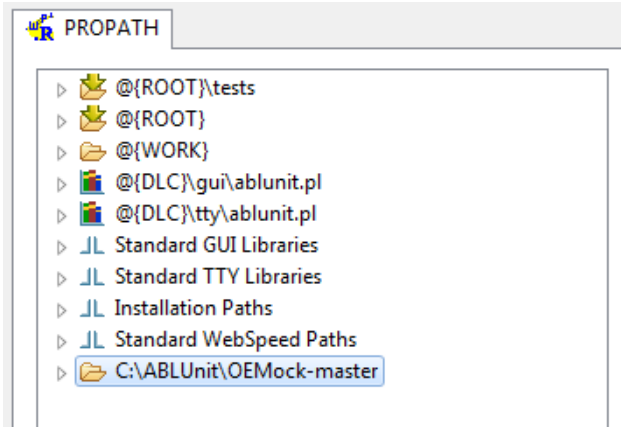

Figure 3: ABLUnit Project's PROPATH

Note: For more information on creating a project, see the product documentation.

With the above steps, you can now use mocking features from the OEMock framework along with the ABLUnit framework while writing the tests. You can even execute the samples supplied in the OEMock framework by copying them to the project and changing the assert APIs from OEUnit to ABLUnit in the samples.

5 Summary

OEUnit and ABLUnit are unit testing frameworks to develop repeatable tests for ABL sources. Each framework has its own strengths, and both frameworks can take advantage of the OEMock mocking framework. This document described how to migrate OEUnit test to the ABLUnit framework, and provided a variety of code samples to illustrate the migration.