BuiltInTester Traceability
By the PHS IDT 2014 Team (phs_winter2014)

Jonathan Ni
Kent Ma
Diwakar Ganesan

Abstract. The purpose of this document is to outline the methods we used to test the required functionality of the Built-In Tester API, and outline examples of the capabilities of our software.

1. Introduction

We will begin by describing how we tested our program to ensure it met the minimum requirements given to us. Then, we will enumerate the extra features we added to our API, and how those were tested.

2. Traceability

1. The Built-in Testing API should be able to be added directly into the code that it is testing.

   To test this feature, we simply tried to instantiate a BuiltInTester object directly into production code:

```
BuiltInTester tester = new BuiltInTester(true);
```

2. The Built-in Testing API should be able to support the byte Java primitive type.

   To test this feature, we used the byteToBinytaryString function in the ByteUtility class. The test case we wanted to use was when the input to byteToBinytaryString was 12.

```
tester.expecting(b, (byte)12, "1100", "b", "byteToBinytaryString(byte)",
                 Byte.class, String.class).
```

   After the block of code, the following was called.

```
tester.log("b", binaryRepresentation);
```

   This was the output produced by BuiltInTester:

```
[I] Variable b PASSED in function byteToBinytaryString(byte) with value [12]. Logged
Output: [1100] --- Expected Output: [1100]
```

   This test demonstrates our API is capable of handling the byte data type.

3. The Built-in Testing API should be able to support the short Java primitive type.

   To test this feature, we wrote our own function, isTwo which returns whether a short argument is equal to 2.

```
import com.phs1437.debugger.BuiltInTester;
public static boolean isTwo(short num)
{
        BuiltInTester tester = new BuiltInTester(true);
        tester.expecting(num, 2, 2, “num”, “isTwo”, short.class, short.class}
        tester.log(“num”, num);
        return num == 2;
}
```

   Output:

```
[I] Variable num PASSED in function ‘isTwo’ with value [22]. Logged Output: [2] ---
Expected Output: [2]
```

   This verifies that our API can handle the short data type.

4. The Built-in Testing API should be able to support the int Java primitive type.

   To test this feature, we used the isEven function in the MathUtility class. The test case we used was for the number 2.

```
tester.expecting(numToCheck, 2, true, "numToCheck", "isEven",
              Integer.class, Boolean.class);
```

   After the block of code, the following value was logged:

```
tester.log("numToCheck", true);
```

   The following output was produced:

```
[I] Variable numToCheck PASSED in function 'isEven' with value [2]. Logged Output:
[true] --- Expected Output: [true]
```

   This test demonstrates our program can handle the int data type.

5. The Built-in Testing API should be able to support the long Java primitive type.

   To test this feature, we created a function returns the factorial of the input

```
import com.phs1437.debugger.BuiltInTester;
public static long factorial(long input)
{
        BuiltInTester tester = new BuiltInTester(true);
        long output = 0;
        tester.expecting(input, 10, 3628800, “input”, “factorial”, Long.class,
Long.class};
        for(int i = 1; i<=input; i++){
                output *= i;
        }
        tester.log(“input”, output);
        return output;
}
```

The code produced this output:

```
[I] Variable input PASSED in function 'factorial' with value [10]. Logged Output:
[3628800] --- Expected Output: [3628800]
```

This demonstrates our API can handle the long data type.

6. The Built-in Testing API should be able to support the float Java primitive type.

   To test this feature, we created a test function that takes in a float, and squares it.

```
import com.phs1437.debugger.BuiltInTester;
public static long squareFloat(float input)
{
        BuiltInTester tester = new BuiltInTester(true);
        tester.expecting(input, (float)25.0,  (float)625, "input",
                        "squareFloat", Float.class, Float.class);
        tester.log(“input”, Math.pow(input, 2));

        return Math.pow(input, 2);
}
```

The output produced for this test was:

```
[I] Variable input PASSED in function 'squareFloat' with value [25.0]. Logged Output:
[625] --- Expected Output: [625]
```

This demonstrates that our API is able to support the float data type.

7. The Built-in Testing API should be able to support the double Java primitive type.

   To test this feature, we created a test function which takes a double and converts it into a percentage.

```
import com.phs1437.debugger.BuiltInTester;
public static double toPercent(double decimal)
{
        tester.expecting(decimal,0.25, (float)25, “decimal”, “toPercent”, Float.class,
                                Float.class);
        tester.log(“decimal”, decimal*100);
        return decimal * 100;
}
```

The output produced was:

```
[I] Variable decimal PASSED in function ‘toPercent’ with value [0.25]. Logged Output:
[25.0] --- Expected Output: [25.0]
```

This demonstrates that our API is able to support the double data type.

8. The Built-in Testing API should be able to support the boolean Java primitive type.

   To test this feature, we created a simple function that performs the NOT operation on a boolean:

```
import com.phs1437.debugger.BuiltInTester;
public static boolean notOperator(boolean b)
{

        tester.expecting(b, true, false, “b”, “notOperator”, Boolean.class,
Boolean.class};
        tester.log(“c”, !b);
        return !b
}
```

   The output for this is as follows:

```
[I] Variable b PASSED in function 'notOperator' with value [true]. Logged Output:
[false] --- Expected Output: [false]
```

   This demonstrates that our API can handle the boolean data type.

9. The Built-in Testing API should be able to support the char Java primitive type.

   To test this feature, we created a function that converts a character to uppercase.

```
import com.phs1437.debugger.BuiltInTester;
public static char toUppercase(char c)
{

        tester.expecting(c, ‘e’, ‘E’, “c”, “toUppercase”, Char.class, Char.class};
        tester.log(“c”, c+32);
        return c+32;
}
```

   This test produced the following output

```
[I] Variable c PASSED in function 'toUppercase' with value [e]. Logged Output: [E] ---
```

```
Expected Output: [E]
```

This demonstrates that our API can handle the char data type.

10. The Built-in Testing API should be able to support the String Java type.

    To test this feature, we wrote a small function that returns the first character in a string.

```
import com.phs1437.debugger.BuiltInTester;
public static char getFirstCharacter(String s)
{

        tester.expecting(s, “IDT”, ‘I’, “s”, “getFirstCharacter”, String.class,
Char.class};
        tester.log(“s”, s.charAt(0));
        return s.charAt(0);
}
```

Here is the output:

```
[I] Variable s PASSED in function 'getFirstCharacter' with value [IDT]. Logged Output:
[I] --- Expected Output: [I]
```

This demonstrates that our API can support the String data type.

11. The Built-in Testing API should be able to support the int array (int[]) Java type.

    To test this feature, we wrote a sample program which contains the int[] data type:

```
import com.phs1437.debugger.BuiltInTester;
public static void main(String args[])
{
        int[] testarray = {1, 2, 3, 4, 5};
        tester.expecting(testarray, new int[] { 1, 2, 3, 4, 5 }, new int[] {1, 2, 3, 4,
5}, “shouldbeTrue”, “main”, Integer[].class, Integer[].class};
        tester.expecting(testarray, new int[] {1, 2, 3, 4, 5}, new int[] {5, 4, 3, 2,
1}, “shouldbeFalse”, “main”, Integer[].class, Integer[].class};
        tester.log(“shouldbeTrue”, testarray);
        tester.log(“shouldbeFalse”, testarray);
}
```

The output produced by this code is:

```
[I] Variable shouldBeTrue PASSED in function 'main' with value [1, 2, 3, 4, 5]. Logged
Output: [1, 2, 3, 4, 5] --- Expected Output: [1, 2, 3, 4, 5]

[I] Variable shouldBeFalse FAILED in function 'main' with value [1, 2, 3, 4, 5]. Logged
Output: [1, 2, 3, 4, 5] --- Expected Output: [5, 4, 3, 2, 1]
```

12. The Built-in Testing API should verify that a given block of code performs as expected.

    This function is verified through the implementation of other functions; by being able to verify each data type, the API can be used with any block of code to verify that it functions as expected.

13. The Built-in Testing API should produce a human readable report of results.

    Through the use of the Logger class, all data collected by the Built-In Testing API is formatted into a human-readable string:

```
        [Tag] Variable variablename PASSED/FAILED in function ‘functionID’ with value
[input]. Logged Output: [loggedValue]. Expected Output: [expectedOutput]
```

    The tags are:

    `[I]` - Information
    `[E]` - Error
    `[W]` - Warning

    Each time the `log` method is called, these human-readable strings are printed to stdout.

14. The Built-in Testing API should be configurable to enable or disable testing at runtime.

    By default, the Built-In Testing API is disabled if `true` isn't given to BuiltInTester's constructor. In addition, there are enable and disable methods:

    - `globalEnable()` and `globalDisable()` sets the state of `BuiltInTester` as a whole -- all `BuiltInTester` objects.
    - `disable()` and `enable()` set the state of individual `BuiltInTester` objects

    By having both global and local disables, the user can control both sections of code that are currently being monitored by `BuiltInTester`.