Test Plan

Overview

The purpose of the program is to simulate the way a cheque program converts a decimal number into the corresponding text.

For example:

Input: "123.45"

Output: "ONE HUNDRED AND TWENTY-THREE DOLLARS AND FORTY-FIVE CENTS"

In of Scope

- Unit Testing
- Functional Testing
- Performance Testing
- Integration Testing

Out of Scope

- Regression testing (N/A)

Environment

Attributes	Requirements
Operating system	Windows XP (32 and 64 bits) SP1 to Windows 8
.NET Framework	4.5

Approach

The purpose of this test plan is to validate the correct conversion of a decimal number into text format in terms of money accuracy.

Assumptions

- The program just accepts two decimal positions, considering that the maximum value for cents is represented in two digits.
  0.1 valid number in terms of money, representing ten cents
  0.01 valid number in terms of money, representing one cents
  00.001 invalid
- In terms of integers, a valid value should be a positive or negative number with 28 digits as maximum.

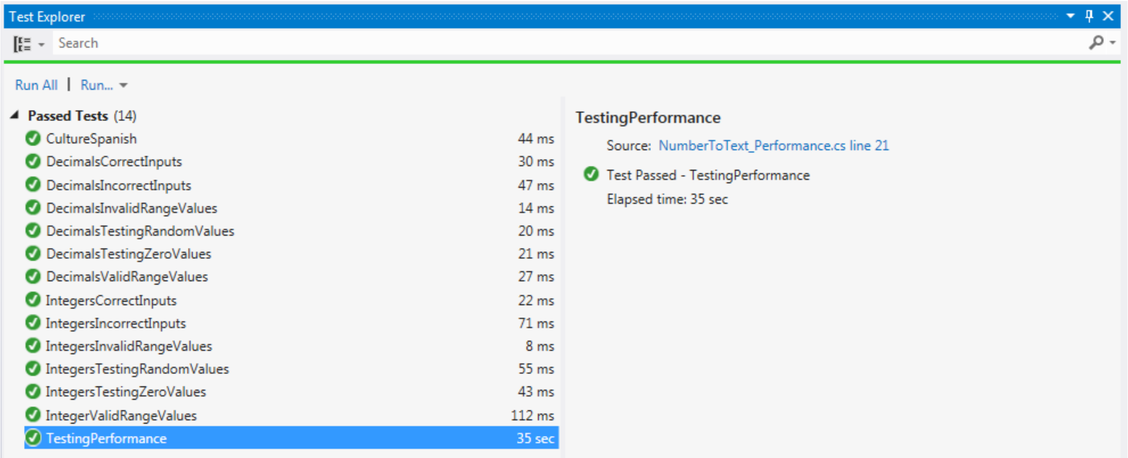
Testing Process

Unit Testing

In order to run the test methods, open the test explorer in Visual Studio and select "Run All". The test data files that are used in the test methods could be found in the following folder "source code\ConverterUnitTesting\TestData\", the methods cover the following scenarios:

Test Method	Data Source	Example
IntegersInvalidRangeValues	IntegersRangeData.xml	**Input**: 99999999999999999999999999999.0 **Expected Result:** `ArgumentOutOfRangeException`
IntegerValidRangeValues	IntegersRangeData.xml	**Input:** `100000.0` **Expected Result:** `one hundred thousand dollars and zero cents`
IntegersIncorrectInputs	IntegersInputData.xml	**Input:** 指事字 **Expected Result:** `ArgumentException`
IntegersCorrectInputs	IntegersInputData.xml	**Input:** `1025.63` **Expected Result:** `one thousand twenty five dollars and sixty three cents`
IntegersTestingRandomValues	IntegersValidData.xml	**Input:** `9.0` **Expected Result:** `nine dollars and zero cents`
IntegersTestingZeroValues	IntegersValidData.xml	**Input:** `02.0` **Expected Result:** `two dollars and zero cents`
DecimalsInvalidRangeValues	DecimalsRangeData.xml	**Input:** `0.999` **Expected Result:** `ArgumentOutOfRangeException`
DecimalsValidRangeValues	DecimalsRangeData.xml	**Input:** `0.99` **Expected Result:** `zero dollars and ninety nine cents`
DecimalsIncorrectInputs	DecimalsInputData.xml	**Input:** `15.96.48` **Expected Result:** `ArgumentException`
DecimalsCorrectInputs	DecimalsInputData.xml	**Input:** `0.5` **Expected Result:** `zero dollars and fifty cents`
DecimalsTestingRandomValues	DecimalsValidData.xml	**Input:** `0.9` **Expected Result:** `zero dollars and ninety cents`
DecimalsTestingZeroValues	DecimalsValidData.xml	**Input:** `-780000460000054600075470001.58` **Expected Result:** `minus seven hundred eighty septillion four hundred sixty quintillion fifty four trillion six hundred billion seventy five million`

		`four hundred seventy thousand one dollars and fifty eight cents`
CultureSpanish	SpanishData.xml	**Input:** `20450.25` **Expected Result:** `veinte mil cuatrocientos cincuenta dolares y veinte cinco centavos`
TestingPerformance	Automatic generation of 1000 decimal numbers	**Input:** **Expected Result:** ConverterPerformanceResults.txt



```
Passed Tests (14)
  CultureSpanish                    44 ms
  DecimalsCorrectInputs             30 ms
  DecimalsIncorrectInputs           47 ms
  DecimalsInvalidRangeValues        14 ms
  DecimalsTestingRandomValues       20 ms
  DecimalsTestingZeroValues         21 ms
  DecimalsValidRangeValues          27 ms
  IntegersCorrectInputs             22 ms
  IntegersIncorrectInputs           71 ms
  IntegersInvalidRangeValues         8 ms
  IntegersTestingRandomValues       55 ms
  IntegersTestingZeroValues         43 ms
  IntegerValidRangeValues          112 ms
  TestingPerformance                35 sec
```

```
TestingPerformance
  Source: NumberToText_Performance.cs line 21
  Test Passed - TestingPerformance
  Elapsed time: 35 sec
```

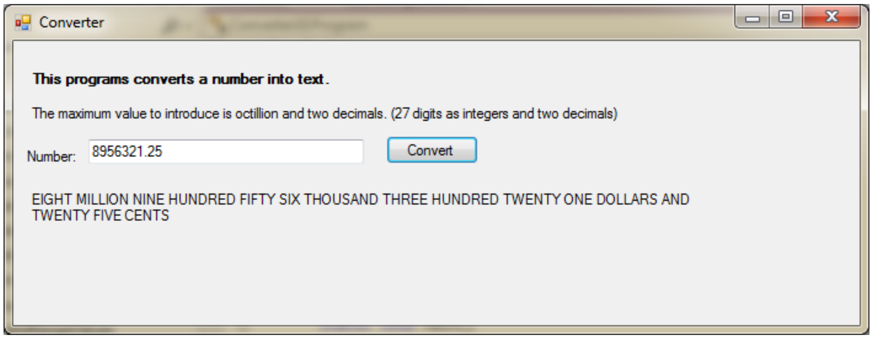
Unit testing methods

Functional Testing

Test.

The Converter desktop application was created to cover the functional testing of the cheque program. In order to run the application download the folder “application” and follow these instructions:

1. Run the executable file **ConverterUI.exe**
2. Type a decimal number on the text field.
3. Click the “Convert” button.



This programs converts a number into text.

The maximum value to introduce is octillion and two decimals. (27 digits as integers and two decimals)

Number: 8956321.25

EIGHT MILLION NINE HUNDRED FIFTY SIX THOUSAND THREE HUNDRED TWENTY ONE DOLLARS AND TWENTY FIVE CENTS

Converter desktop application

Note: The ConverterUI.exe needs the following components to work: ConverterResources.dll, Converter.dll

Expected Result.
English text format of the decimal number input.

Performance Testing

Test:

In order to test the performance of the converter; in the test explorer in Visual Studio run the “TestingPerformance “ function that generates 10,000 random numbers within the valid ranges.

Expected result:

A file named *ConverterPerformanceResults.txt* at the same level of the **ConverterUI.exe**

Integration Testing

Test:

The converter is a class library implemented in C#; however, it could be used in a desktop application as it’s currently shown in the functional testing.

Expected Result:

The user should be able to use the Converter.dll through the ConverterUI.exe interface.

Plan History

Version	Developer	Tester	Date	Comments
1.0	Brenda Jimenez		08/09/2013	Creation of the test plan.