WORD COUNTING PROBLEM

# PROBLEM STATEMENT

Our users require a service to generate word counts. They would like to be able to enter a statement, which is then analyzed and added to the existing word count. The service needs to be able to handle multiple users adding statements at the same time. A user expects their entire statement to either be fully added or not at all (the count should never contain only some of the words from their statement).

The users would prefer that punctuation is removed, but this is optional for Phase 1. The count should contain one of the users’ complete statements, before the other user’s statement is added.

# SOLUTION APPROACH

I utilized the concept of layered architecture:

Layer 1: simple utility that accepts a user’s string statement, validates it and breaks it into the simple dictionary of counts, keyed by the trimmed word, which is also converted to the upper case (to remove double-entries into the dictionary). As the entered sentence may contain punctuation, the two identical words, one of which is punctuated and the other one is not, are considered as two distinct keys by the utility.

As the utility has to be easily tested, reliable and readable, I decided to implement it with the number of extension methods. As extensions methods are part of the library metadata, running them as some part of the unit tests is seamless. The extensions methods also provide high code readability, since a client can invoke those methods in chain, where the result of every operation is passed as an input into the next operation, creating a functional call similar to the one provided below. It’s also important that by changing the input parameter of an extension method, you could easily enforce the order of operations, which makes the coding style very close to a “domain language”. For example, in my implementation, it is impossible to apply the CountWordOccurances method BEFORE the ToWordArray method, as the former accepts an enumerable collection of words as its input, and such the collection is an output of the latter. On the other hand, if a client chose to generate the enumerable collection of words, she can still easily used the CountWordOccurances method independently, as it does not have any external dependencies on any other methods (note that it is OK for an extension method to have some internal dependencies on other extension methods, as the ToWordArray has such a dependency on Validate, as the user may chose not to / forget to apply the Validate before calling the ToWordArray:

“some string”.ToWordArray().CountWordOccurances()…;

Layer 2: wrapping WordCounter. This class:

1. Wraps the utility into a concise call.
2. Ensures execution strategy (please see the “Assumptions” section for more details).
3. Provides an implementation of the WCF service contract IWordCount.

Layer 3:

3.1. WCF service runner. This is a simple console application that runs a single instance of the WordCounter class as a WCF service, exposed via its associated IWordCount contract. This part of the architecture / infrastructure exists to:

1. Fulfill the “Create Service” user requirement
2. Increase testability and transparency of the architecture, as at this point we can test the whole service functionality without writing any additional code, but only by using a standard wcftestclient utility shipped as a part of the Visual Studio installation.

3.2. WordCountClient – this is a simple WPF application, allows to test the WordCount WCF service for those client, who does not have VS installed, including more complex scenarios (two clients entering their statements at the same time) unavailable in wcftestclient.

# ASSUMPTIONS

1. Out of order execution; that is, Client2 can receive the result of the execution of her statement faster than Client 1, provided that the execution and dictionary update is still transactional.
2. Client does not receive the copy of the service dictionary, but rather the string describing the state of the dictionary.
3. Implementation details of logging and configuration are left outside the scope. However, the solution provides placeholders for both, as well as the default implementation of the logger.

# TOOLS

1. Visual Studio 2012.
2. NUnit plugin for Visual Studio 2012 from

http://visualstudiogallery.msdn.microsoft.com/6ab922d0-21c0-4f06-ab5f-4ecd1fe7175d

# IMPLEMENTATION

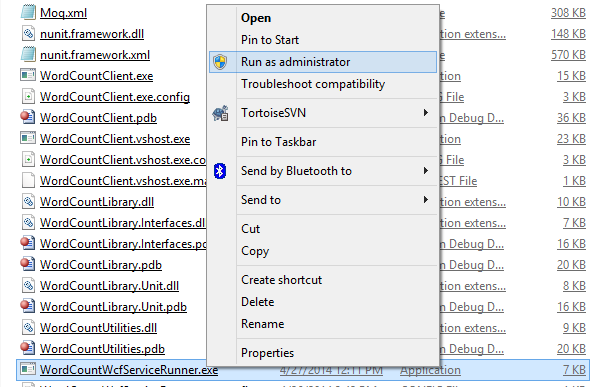
### SOLUTION STRUCTURE

WordCount folder contains a Build subfoder with a compiled version of the proposed solution, as well as solution itself (WordCount.sln).

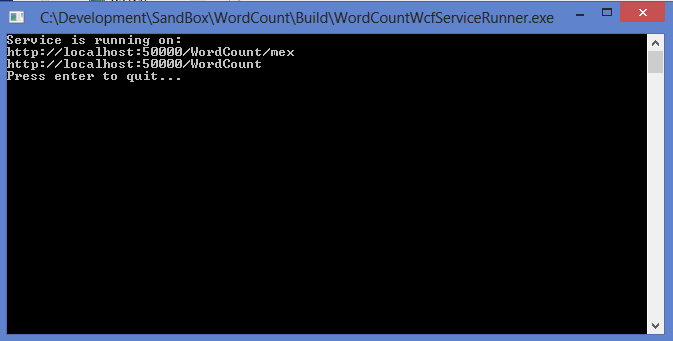
### HOW TO START WORD COUNT SERVICE AND CLIENT

The WordCount WCF service should be started first. If it is not, the wcftestclient or WordCountClient will simply display an error on attempt to connect to the service.

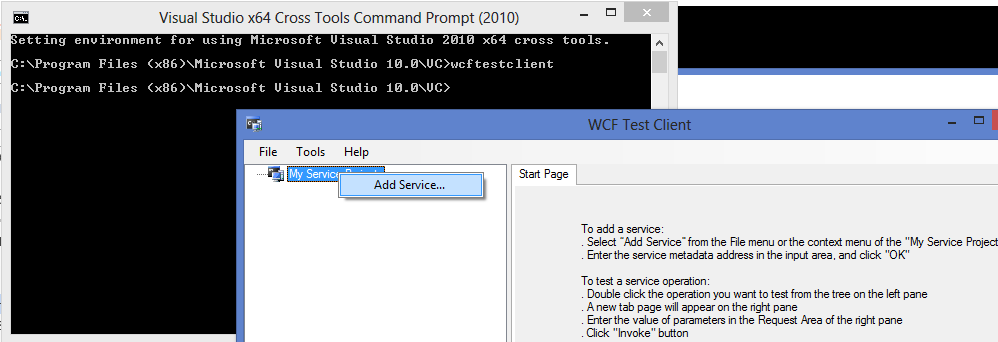
1. Switch to the Build subdirectory and find WordCountWcfServiceRunner.exe. Right-click on the executable and start it in the **administrator mode**:



If successful, the process should start and begin to listen on its associated endpoints: <http://localhost:50000/wordcount>. Note: the listening endpoint is not configurable!



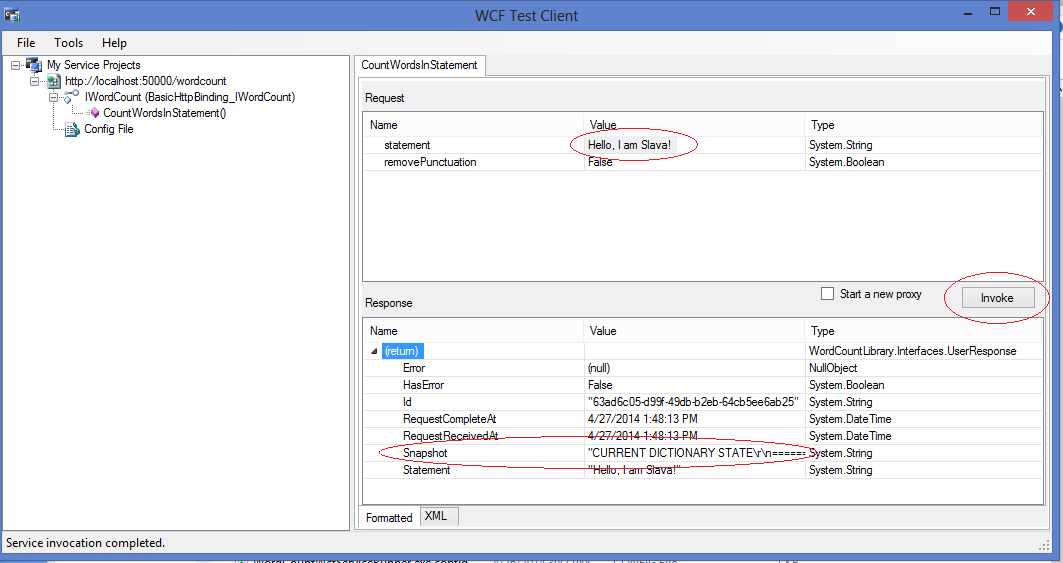
# At this point, you’re welcome to test the service using the standard wcftestclient utility. To to this, please open Visual Studio 2012 command prompt and type “wcftestclient”:



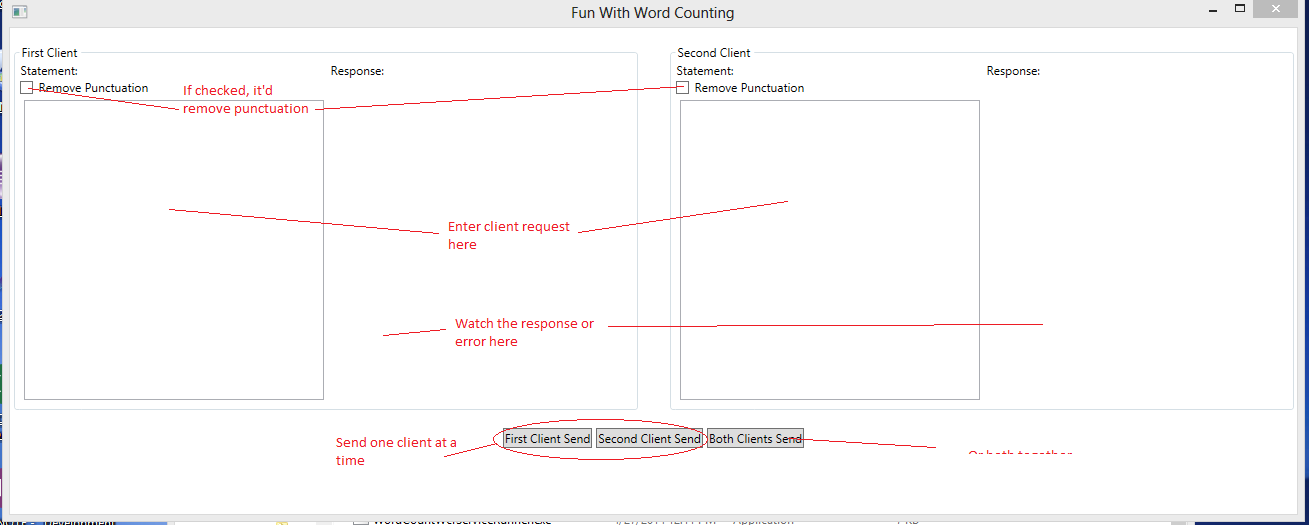
# Click on “Add Service…”, enter <http://localhost:50000/wordcount> and click “OK”:

# 

# You’re now connected to the service! Now, you can enter some phrase into the “statement” Value on the right pane of the service, press the “Invoke” button and obtain the result:

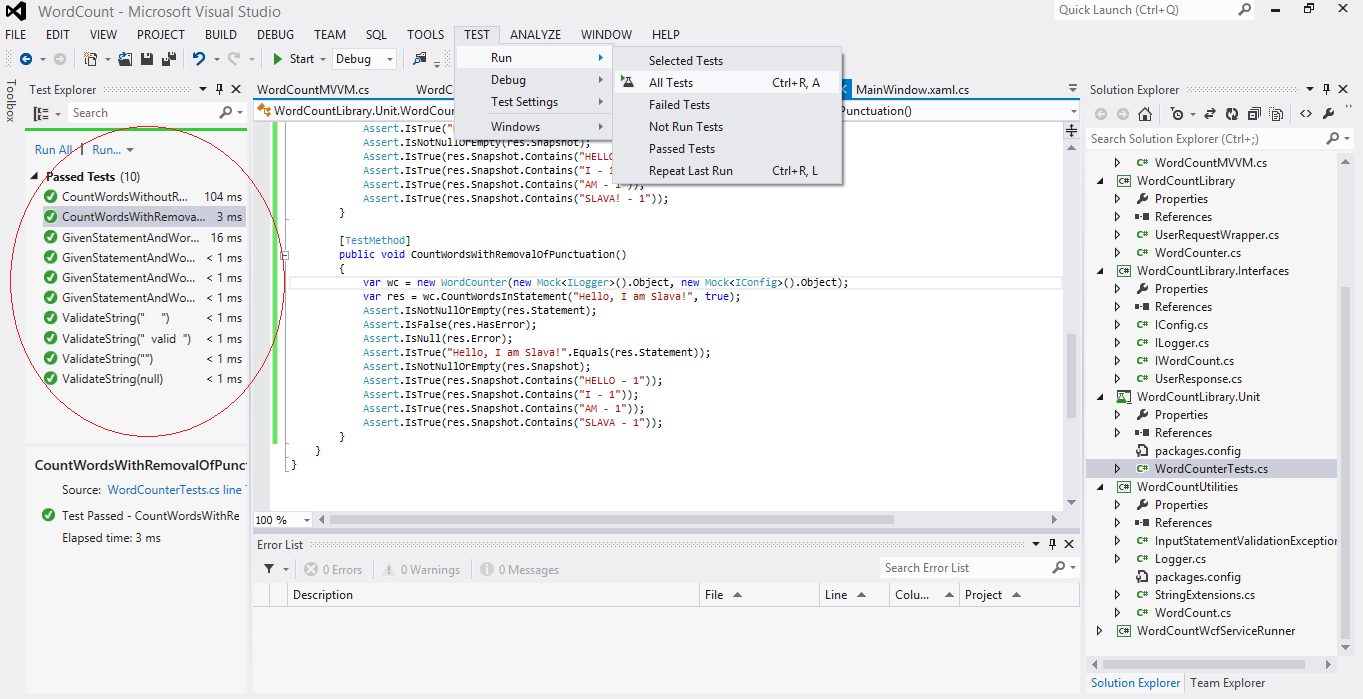


# To run and test the client, in the Build subdirectory, please find WordCountClient.exe and invoke it (no need to be in the administrator mode):



# TESTING

# Previous section corresponds to the integration part of testing. In order to run unit tests, please open Visual Studio, open the WordCount solution. All tests run and show green (succeeded):



### UNIT TESTING

#### StringExtensions

Validate – if supplied string is valid (not null, non-empty, not entirely from white spaces)

Check word split – correct generation of word dictionary

Check word occurances – correct generation and counting of word-keyed dictionary and correct counts for all keys.

WordCounter

Test wrapping call. Note that the invocation strategy is tested as a part of the integration testing.

### INTEGRATION TESTING

Please see the previous section for integration.