Arlo Developer Test (Backend)

## Instructions

* You will need access to a computer with [Microsoft Visual Studio](https://www.visualstudio.com/en-us/products/free-developer-offers-vs.aspx) (use free version if required).
* At the end of the test, if you have not completed the tasks, explain what you would have done next.
* This test is in C#, but if you think you can show off your skills better in another language, we’ll accept solutions in Java and Python as well.
* Send your results (a zip archive) to [careers@arlo.co](mailto:careers@artlo.co).

Thanks, and good luck!

## 

## 

## C#

Your task is to implement a string template engine in C# that substitutes tokens with values from a datasource, suitable for use in generating notification emails. Examples of input/output are shown below.

With this exercise, you will need to download our [archive](https://drive.google.com/file/d/0B1QexlklX4XKT0ZURVo1REJ5V3c/view?usp=sharing) with a Visual Studio project where you will write your implementation. The project contains isolated unit tests to help you with testing your implementation.

To produce output, the engine’s Apply() method is given two arguments:

* A string template (like Hello [Name])
* An object datasource (like { Name = “John” })

and will process these to produce a new string where *tokens* in the template are replaced with appropriate values from the datasource:

* Hello John

### Specification

The engine must support only three types of tokens/tags:

* Simple field name tokens: [*FieldName*]
* Spanned property name tokens: [*FieldA*.*FieldB*]
* STRETCH GOAL - Scope blocks: [with *FieldName*]...[/with]

The engine must be able to work with any valid template (not just the examples shown below), and any datasource.

***Simple token substitution [FieldName]***

|  |  |  |
| --- | --- | --- |
| **Template** | **DataSource** | **Output** |
| Hello [Name] | { Name = “John” } | Hello John |

***Span token substitution [FieldA.FieldB]***

|  |  |  |
| --- | --- | --- |
| **Template** | **DataSource** | **Output** |
| Hello [Contact.FirstName] [Contact.LastName] | {  Contact = new {  FirstName = “John”,  LastName = “Smith”  }  } | Hello John Smith |

***STRETCH GOAL - Scoped block token substitution [with FieldName]...[/with]***

The ‘with’ token is combined with an expression that defines a scope object that becomes the context for the block. It is similar to the JavaScript ‘with’ syntax.

All token expressions within the scope’s block should be evaluated with respect to this scope object.

Scopes can be nested.

|  |  |  |
| --- | --- | --- |
| **Template** | **DataSource** | **Output** |
| [with Contact]  Hello [FirstName] [LastName]  [with Organisation]  You are from [Name] in [City]  [/with]  [/with] | {  Contact = new {  FirstName = “John”,  LastName = “Smith”  Organisation = new {  Name = “Acme Ltd”  City = “Auckland”  }  }  } | Hello John Smith  You are from Acme Ltd in Auckland |

### 

### *Simple token substitution with format arguments [FieldName “format”]*

Tokens may contain a quoted format argument specifically for date formatting during substitution. For simplicity, assume you need to apply this only to DateTimeOffset types, and that the format specification is one natively supported by C#’s DateTimeOffset.ToString() -- you don’t need to write your own format parser.

If a format argument is supplied for a non-DateTimeOffset value (such as a property of type string or int), it should be ignored.

|  |  |  |
| --- | --- | --- |
| Template | DataSource | Output |
| The current date is [Today “d MMMM yyyy”] | { Today = DateTimeOffset.Now } | The current date is 7 December 1980 |

### 

### Task

* Along with these instructions, you have been provided a ZIP file containing a Visual Studio solution.
* The Visual Studio solution has two projects:
  + A core project -- where your implementation should be written
  + A unit test project -- that contains pre-written tests to verify the expected behavior.
* Your task is to write an implementation in the TemplateEngineImpl.cs file that meets the requirements above and passes all of the unit tests in TemplateParser.Test project.
* You can use integrated test tools (Test Explorer window) in Visual Studio if you wish to test your code against the existing unit tests.
* If you can’t get the integrated testing to work (or aren’t sure how to use this part of Visual Studio), the Test project is written as a simple console app that you can F5 run to test after each build.

### Constraints

* If a token cannot be evaluated (because it refers to a property that does not exist), it should be replaced with a blank string.
* If the syntax grammar is invalid, you may throw a parse exception.
* You may add as many additional classes as you like to the project.

### Hints

* This is fundamentally a parsing problem, so that should inform your solution
* While it’s possible to solve this by spamming string replaces, if you find yourself going down that route it might be best to take a step back
* This is a much simplified version of the parser that we use in our system to generate emails. The real version has support for looping/conditionals/etc. It would be great if your version had fundamentals you could build on later if you had to extend the functionality to include those advanced features.

### Delivery

## The output of the exercise should be a zip archive containing a VisualStudio solution that has passing unit tests. Also when you email your solution talk about what you did/didn’t like about what you did. If you have great ideas in your head we’d like to give you credit for those. We’d be interested in hearing you talk about what circumstances your solution would work well in and what solutions it wouldn’t.