Firstly, the sum up of mine is that I have pass all of unit test cases. Here I explain some of my insight ideas and the problems that I have faced.

The first procedure when I got this question is problem investigating. In my mind, it is a parsing string problem. It provides two parameters, one is a string named as Template, another is a corresponding struct like variable named as DataScource. For this parsed Template String, there contains multiple values or multiple parallel attributes because the exist of nested patterns. I believe strong robust programs can be parsed regardless the layers of nesting or the variety of attributes.

The second procedure is to reflect on the outcome scheme. My first idea is to create a class for property, so the varieties of property will be able to parse by using function overload, the normal content Strings in the Template will be parsed also. The next process is to build a tree-like structure for containing these properties, because of the exist of the nested parallel properties, the depth-first iterating can be used to pass through all the elements to obtain the result. Moreover, the purpose of this approach is to treat all the Strings as one object, but parsing Template meanwhile parsing DataSource will be required. When parsing the template, the name of required property was obtained, thus the corresponding value can be taken from the DataSource by reflection. The second idea is to use stack to substitute tree, this is because the stack has features of last in first out (first in last out) it will be able to have a deep-first implementation on the tree-like structure.

The third procedure would be the implementation. My plan is to implement the overall framework first and then enrich the features, so I wrote the pseudocode for overall framework in advance. I named the framework parsingTemplate. I can block some other unit test cases, this allows me to pass the enriched feature test cases after the pass of basic cases. I am used to make comment when I finished each class, and think about whether there are any codes can be reuse for similar methods.

Furthermore, for the problems that I have faced. The first one I came across during the implementation was I realized that I could not to use tree objects, conversely, this might be a little over-designed. Because it is parsing trees when parsing template Strings, so I wrote the parsing as a Recursion, and obtained the contents in deserved order. I parse each layer directly and return a final String, then saved the String into an Arraylist to converse to result String conveniently.

The second problem is when the template string is parsed, if parsing it by using “split(“ ”);” to get the string list. the string list received have the chance of containing cases such as "[with", "Contact]" and "[City][/with]". It could have some trouble obtaining the contents between “[with] [/with]” tokens, so in this case I treat it as a special case, unless the contains can be obtained directly and do not contain any tokens then perform normal splitting (“split(“ ”);”) to get each of the elements.

The third problem is based on the second case, because if people want to obtain the contents between two tokens, I implement this through subtract the length of the starting token to receive the length of the central contents. In this way, through “substring(starting token length, content length)” to obtain the contents. For the content length then I must know the ending token’s start index. Therefore, I applied an algorithm which is a string-matching algorithm, due to the simple case, I only use Brute Force matching. Due to the current case is simple, otherwise the KMP algorithm might be better for advance.

In addition, I have conquered is C# language tools, which I solve it by bring my Object-Orient Programming Knowledges and other languages experiences. There is some circumstance that I might not so well handled and probably not covering the whole contents due to the length, if there is any confusion, please discuss with me. I am also looking forward to correcting my mistakes if there happens to be any.