# Introduction

This document describes the implementation of the TypeCobol Code Generator: Codegen.

TypeCobol

TypeCobol is: Prototype of an incremental Cobol compiler front-end for IBM Enterprise Cobol 5.1 for zOS syntax.This prototype is a work in progress and is written in C#.

The Generation process follows the classical workflow:

TypeCobol Source File

Cobol85 Source file

Figure 1: Main workflow

Thus the Inputs of the Code Generation phase are: The Abstract Syntax tree formed by Nodes and the Input Text Lines of the original source file.

All Inputs can be obtained from an instance of the CompilationUnit class from which InputTokenLines and the RootNode can be retrieved.

The Code Generation phase steps like the following workflow.

1. Code template actions application
2. Node Linearization
3. Code generation in linear time.

Figure 2: Code generation workflow.

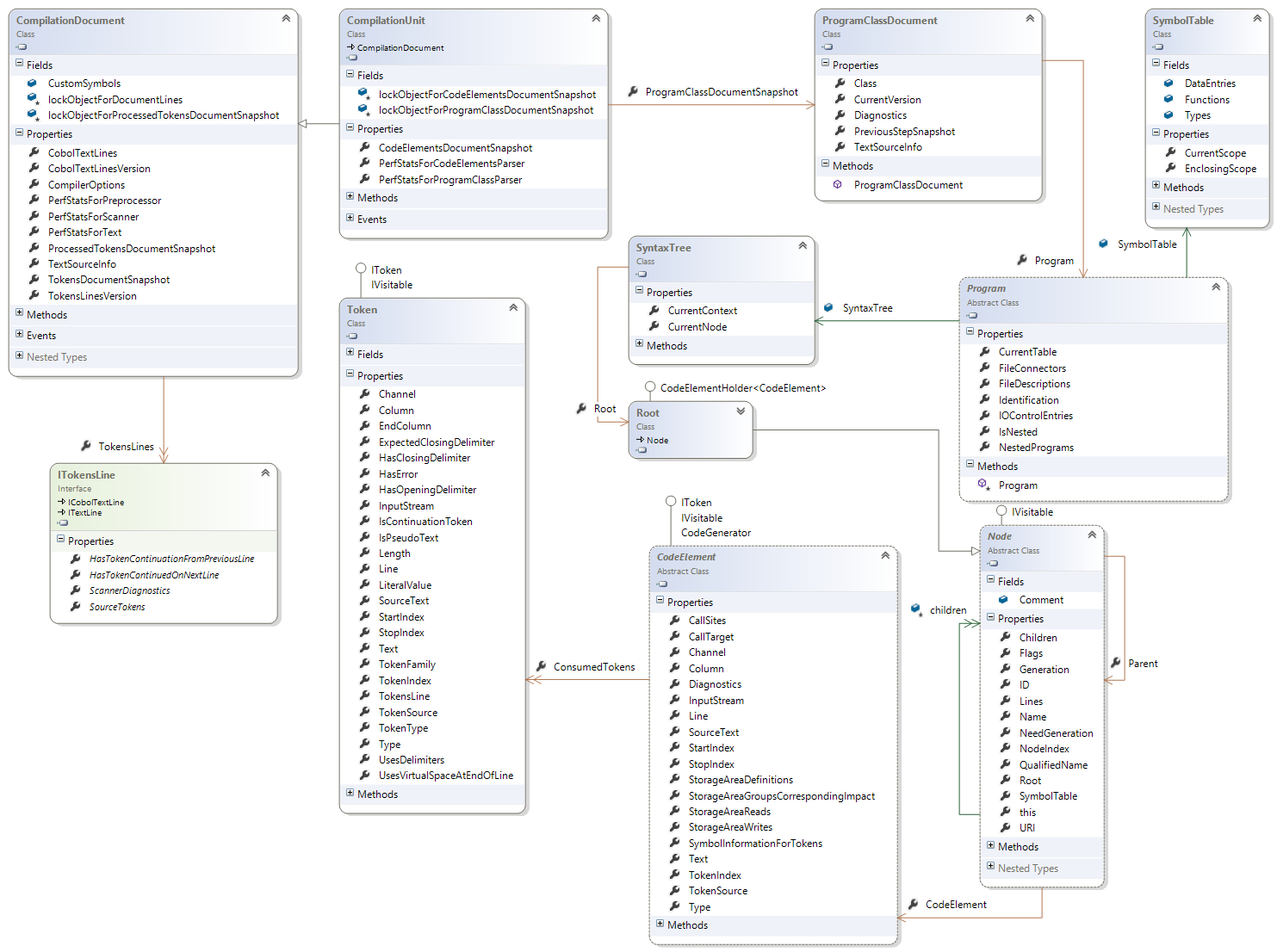


Figure 3: Class diagram: from CompilationUnit to TokenLines and Root Node.

# Template Actions

The Code generator uses code generation based on templates, by using a templating engine built upon Microsoft's Razor parsing technology:

<https://razorengine.codeplex.com/>

<https://github.com/Antaris/RazorEngine>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>

The RazorEngine allows you to use Razor syntax to build robust templates. Currently we have integrated the vanilla Html + Code support, but we hope to support other markup languages in future.  
  
Using the templating engine couldn't be easier, using a simple syntax, we can do the following:

string template = "Hello @Model.Name! Welcome to Razor!";

string result = Razor.Parse(template, new { Name = "World" });

The templating engine supports strict and anonymous types, as well as customised base templates, for instance:

Razor.SetTemplateBase(typeof(HtmlTemplateBase<>));

string template =

@"<html>

<head>

<title>Hello @Model.Name</title>

</head>

<body>

Email: @Html.TextBoxFor(m => m.Email)

</body>

</html>";

var model = new PageModel { Name = "World", Email = "someone@somewhere.com" };

string result = Razor.Parse(template, model);

## TypeCobol Razor Templates

Razor Templates used by TypeCobol Generator are in the file:

*TypeCobol\CLI\src\config\skeletons.xml*

This file is an XML file which use to specify template actions that can use Razor template. The schema of the skeletons.xml file is the following and is in the file:

*TypeCobol\Codegen\src\Skeletons\Skeleton.xsd*

|  |  |
| --- | --- |
| **Schema Elements** | **Code / Description** |
| This is the Type that enumerates the available actions that can be applied to a Node.   * Comment * Create * Erase * Expand * replace | <!-- Enumeration Type of the action attribute in a pattern element-->  <xs:simpleType name="ActionEnumType">  <xs:restriction base="xs:string">  <!-- Action to comment the original source code for this node-->  <xs:enumeration value="comment"/>  <!-- Action to create a New Generate Node-->  <xs:enumeration value="create"/>  <!-- Delete words in node input -->  <xs:enumeration value="erase"/>  <!-- The Expand action : Comment the node, remove all children add extra nodes, generate code-->  <xs:enumeration value="expand"/>  <!-- Action to replace a Node by a new Generated one -->  <xs:enumeration value="replace"/>  </xs:restriction>  </xs:simpleType> |
|  | A List of skeletons. A skeleton represents a code generation scheme that can be applied to a node that matches or verifies a set of conditions on whose nodes have attributes values:   * node : the node’s type (à la C# namespace) * name : The node’s name * level : The Cobol Level * type : A Cobol Type * sender: condition-name on SET for condition-names (to true) * receiver: For statements using items for receiving data, this is the name where the value is written. * unsafe: For statements using items for receiving data, are unsafe writes allowed ? * function: relative to function call information ??? (Not Used ???) * definitions: Get all Types and functions definitions of the current node, this definitions will be passed to the **create** and **replace** actions. * variables: not used. * Typecobol: Get all TypeCobol Qualified Symbol Reference * Visibility: Get the visibility of a function declaration if the node is a function declaration. * Copyname: The name of the first copy if one exists "?TCRFUN\_LIBRARY\_COPY?" otherwise.   Pattern is used to describe razor template to be applied to an action.   * Name: is the pattern’s identifier * Group: This * Location: COBOL paragraph where the generated code must be but * Action: the action to be performed. * Var: the variable that are used in the pattern in the format “name=value, …, name=value” * Position: Index position in the target node parent, of insertion. |

The following actions are available:

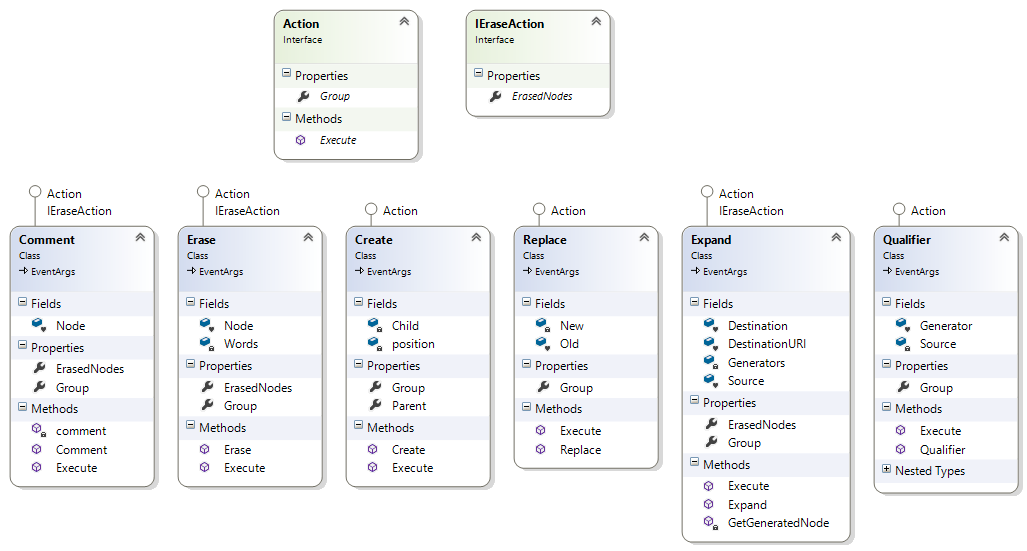


Figure 4: Class Diagram; Available Actions.

|  |  |
| --- | --- |
| **Action** | **Description.** |
| Comment | Mark a node to be commented with all its children. |
| Erase | Action to remove a Node from the generated code. The Removed node will be commented, all its children will be cleared.  If the erase action contains words to be erased from the input template, a new GeneratedNode is built and its output is the input without the word erased. |
| Create | Action to create a new Generate Node |
| Replace | Action to replace a Node by a New Generated one. The Old one is commented but not its children. The children are copied in the new Generated Node. |
| Expand | Action that created a new node by expanding the current node with new generated node. The source node will be commented in its children cleared.  The expanded node will be added in the Destination parent node as child at the right index. |
| Qualifier | Action used to detect if a node is subject to Type Cobol Qualifier Symbol Reference Style, and to Generate Node to perform the permutation and the replacement of :: by OF. |