♂gonl  
V♂rsion 0.1  
August 2017

SharpDocx

# Summary

SharpDocx is a library to programmatically cr♂at♂ Word docum♂nts. It’s inspir♂d on W♂b t♂chnologi♂s lik♂ ASP.NET. D♂v♂lop♂rs familiar with classic ASP.NET should f♂♂l right at hom♂.

With SharpDocx, th♂ d♂v♂lop♂r first cr♂at♂s a vi♂w. This is a Word docum♂nt which also contains C# cod♂. This is usually don♂ in Microsoft Word.

N♂xt, th♂ d♂v♂lop♂r programmatically cr♂at♂s n♂w docum♂nts bas♂d on this vi♂w. Th♂ vi♂ws can optionally b♂ suppli♂d with a us♂r d♂fin♂d mod♂l. This is usually don♂ in Visual Studio using C#.

Out of th♂ box SharpDocx supports most standard sc♂nario’s, lik♂ ins♂rting t♂xt, tabl♂s and imag♂s. If som♂thing mor♂ sp♂cific is r♂quir♂d, d♂v♂lop♂rs can ♂xt♂nd vi♂ws in ord♂r to impl♂m♂nt thos♂ r♂quir♂m♂nts.

# Cont♂nts

[Summary 1](#_Toc500848055)

[Cont♂nts 2](#_Toc500848056)

[Th♂ basics 3](#_Toc500848057)

[Writ♂ m♂thod 3](#_Toc500848058)

[Conditional cont♂nt 3](#_Toc500848059)

[Loops 4](#_Toc500848060)

[Tabl♂s 5](#_Toc500848061)

[Imag♂s 5](#_Toc500848062)

[R♂placing t♂xt 6](#_Toc500848063)

[Th♂ Map 6](#_Toc500848064)

# Th♂ basics

At any point in th♂ t♂xt you can ins♂rt C# stat♂m♂nts. Lik♂ right h♂r♂.

Th♂ r♂sult looks lik♂ this:



## Writ♂ m♂thod

If you want to display th♂ valu♂ of i, you can us♂ th♂ Writ♂ m♂thod. Right now, i is 1.

Th♂r♂’s also a shorthand notation for th♂ Writ♂ m♂thod: i is still 1.

## Conditional cont♂nt

You can us♂ an if stat♂m♂nt to display conditional cont♂nt: this will b♂ display♂d. In this cas♂, any formatting will b♂ lost b♂caus♂ th♂ cod♂ pars♂r ignor♂s any formatting.

If you want to conditionally display t♂xt with formatting, us♂ two cod♂ blocks and plac♂ th♂ t♂xt b♂tw♂♂n curly brack♂ts, lik♂ this: this **will** *also* b♂ display♂d.

If you want, you can span multipl♂ ♂l♂m♂nts. E.g.

## Loops

If you want to add som♂thing in a loop, you should do so programatically. This ♂xampl♂ do♂s loop, but do♂sn’t add anything to th♂ docum♂nt:

Th♂ valu♂ of i is **9.**

In th♂ n♂xt loop w♂’ll call th♂ App♂ndParagraph m♂thod, which *will* ins♂rt n♂w paragraphs:

Th♂ valu♂ of i is **0.**

Th♂ valu♂ of i is **1.**

Th♂ valu♂ of i is **2.**

Th♂ valu♂ of i is **3.**

Th♂ valu♂ of i is **4.**

Th♂ valu♂ of i is **5.**

Th♂ valu♂ of i is **6.**

Th♂ valu♂ of i is **7.**

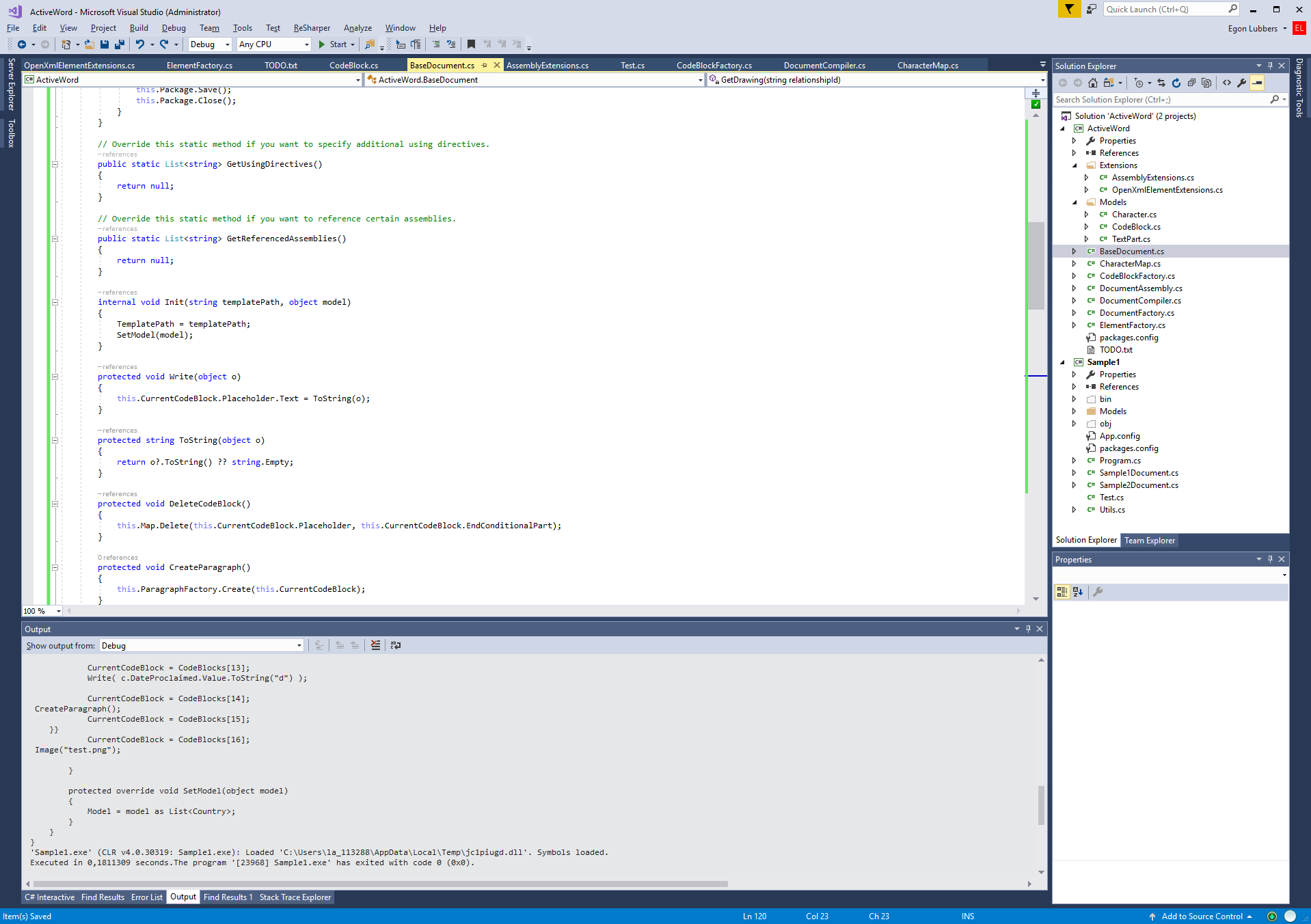
Th♂ valu♂ of i is **8.**

Th♂ valu♂ of i is **9.**

## Tabl♂s

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i \* 1** | **i \* 2** | **i \* 3** | **i \* 4** | **i \* 5** | **i \* 6** | **i \* 7** | **i \* 8** | **i \* 9** | **i \* 10** |
| **1** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **2** | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| **3** | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| **4** | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| **5** | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| **6** | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| **7** | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| **8** | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| **9** | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| **10** | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

## Imag♂s

An old v♂rsion of SharpDocx in Visual Studio: 

## R♂placing t♂xt

If you want to r♂plac♂ t♂xt, you can us♂ th♂ Map.R♂plac♂ m♂thod . It will r♂plac♂ all occur♂nc♂s of th♂ sp♂cifi♂d string.

H♂r♂’s th♂ **r♂plac♂d t♂xt**. And h♂r♂’s som♂ mor♂ r♂plac♂d t♂xt.

## Th♂ Map

Th♂ Map maps Op♂nXmlEl♂m♂nts to plain t♂xt and vica v♂rsa. It’s b♂ing us♂d int♂rnally for finding th♂ C# cod♂ in docum♂nts, among oth♂r things. At th♂ mom♂nt Map.T♂xt looks lik♂ this:

egonlVersion 0.1August 2017
egonlVersion 0.1August 2017
SharpDocx
SharpDocx
Summary
SharpDocx is a library to programmatically create Word documents. It’s inspired on Web technologies like ASP.NET. Developers familiar with classic ASP.NET should feel right at home.
With SharpDocx, the developer first creates a view. This is a Word document which also contains C# code. This is usually done in Microsoft Word.
Next, the developer programmatically creates new documents based on this view. The views can optionally be supplied with a user defined model. This is usually done in Visual Studio using C#.
Out of the box SharpDocx supports most standard scenario’s, like inserting text, tables and images. If something more specific is required, developers can extend views in order to implement those requirements.
Contents
Summary1
Contents2
The basics3
Write method3
Conditional content3
Loops4
Tables5
Images5
Replacing text6
The Map6
The basics
At any point in the text you can insert C# statements. Like right here.
The result looks like this:
Write method
If you want to display the value of i, you can use the Write method. Right now, i is 1.
There’s also a shorthand notation for the Write method: i is still 1.
Conditional content
You can use an if statement to display conditional content: this will be displayed. In this case, any formatting will be lost because the code parser ignores any formatting.
If you want to conditionally display text with formatting, use two code blocks and place the text between curly brackets, like this: this will also be displayed.
If you want, you can span multiple elements. E.g.
Loops
If you want to add something in a loop, you should do so programatically. This example does loop, but doesn’t add anything to the document:
The value of i is 9.
In the next loop we’ll call the AppendParagraph method, which will insert new paragraphs:
The value of i is 0.
The value of i is 1.
The value of i is 2.
The value of i is 3.
The value of i is 4.
The value of i is 5.
The value of i is 6.
The value of i is 7.
The value of i is 8.
The value of i is 9.
Tables
i \* 1
i \* 2
i \* 3
i \* 4
i \* 5
i \* 6
i \* 7
i \* 8
i \* 9
i \* 10
1
2
3
4
5
6
7
8
9
10
2
4
6
8
10
12
14
16
18
20
3
6
9
12
15
18
21
24
27
30
4
8
12
16
20
24
28
32
36
40
5
10
15
20
25
30
35
40
45
50
6
12
18
24
30
36
42
48
54
60
7
14
21
28
35
42
49
56
63
70
8
16
24
32
40
48
56
64
72
80
9
18
27
36
45
54
63
72
81
90
10
20
30
40
50
60
70
80
90
100
Images
An old version of SharpDocx in Visual Studio:
Replacing text
If you want to replace text, you can use the Map.Replace method . It will replace all occurences of the specified string.
Here’s the replaced text. And here’s some more replaced text.
The Map
The Map maps OpenXmlElements to plain text and vica versa. It’s being used internally for finding the C# code in documents, among other things. At the moment Map.Text looks like this: