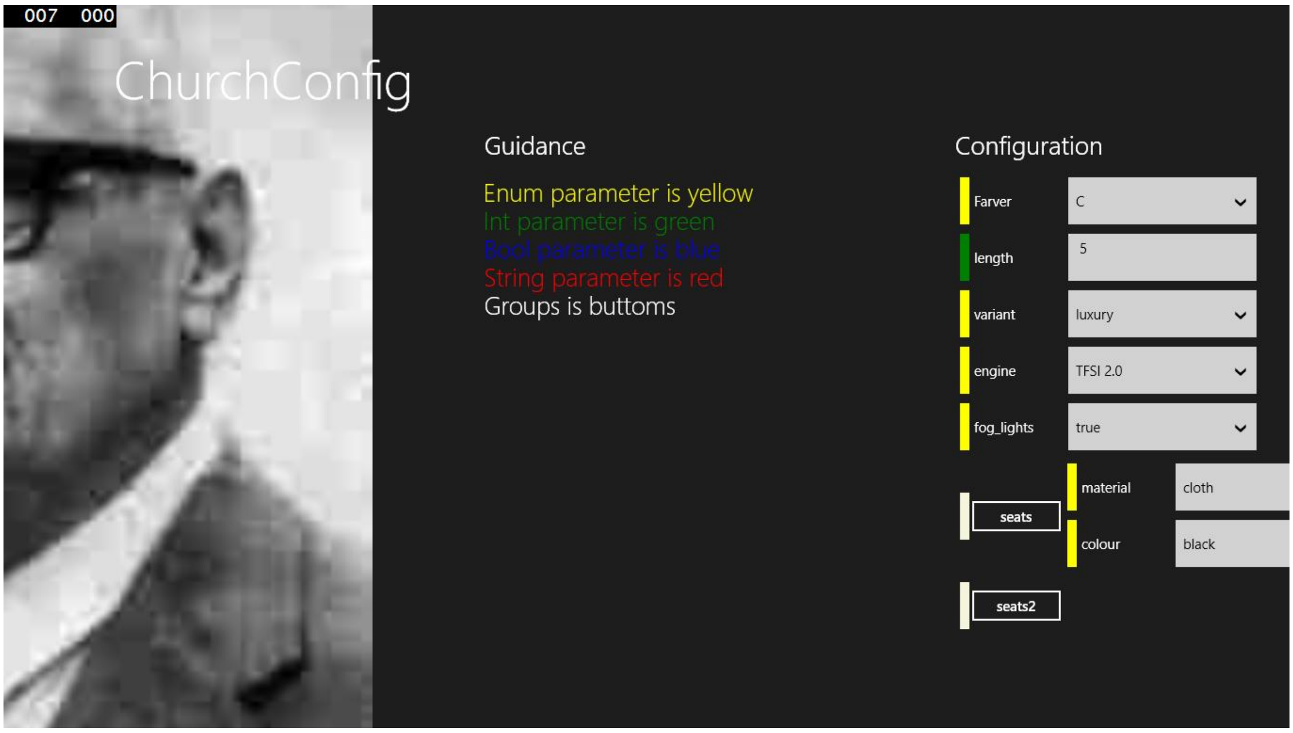
**Contents**

Configurator App(Windows store app) ........................................................... 1
Code generator ................................................................................... 2
Code generation output ........................................................................ 2

## Configurator App(Windows store app)

Image of the CS configurator application with the 'VW' test configuration



Guidance

Enum parameter is yellow
Int parameter is green
Bool parameter is blue
String parameter is red
Groups is buttoms

Configuration

| | |
|---|---|
| Farver | C |
| length | 5 |
| variant | luxury |
| engine | TFSI 2.0 |
| fog_lights | true |
| material | cloth |
| colour | black |

https://github.com/smdp2015/project/tree/master/Smdp2015DotNetClient

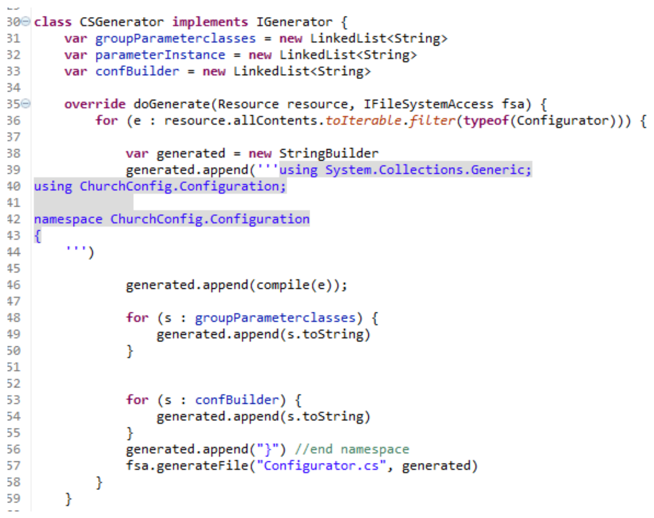
FileDescription

ChurchConfig / Configuration / Configurator.cs – generated code
ChurchConfig / Configuration / CommonConfig.cs – BaseClasses to the generated code
ChurchConfig / ConfigControl.xaml – configurator usercontrol
ChurchConfig / HubPage.xaml – Application mainpage

## Code generator

https://github.com/smdp2015/project/tree/master/dk.itu.smdp2015.church.configurator.syntax/src/dk/itu/smdp2015/church/generator/CSGenerator.xtend

All in one file.



```
class CSGenerator implements IGenerator {
    var groupParameterclasses = new LinkedList<String>
    var parameterInstance = new LinkedList<String>
    var confBuilder = new LinkedList<String>

    override doGenerate(Resource resource, IFileSystemAccess fsa) {
        for (e : resource.allContents.toIterable.filter(typeof(Configurator))) {

            var generated = new StringBuilder
            generated.append('''using System.Collections.Generic;
using ChurchConfig.Configuration;

namespace ChurchConfig.Configuration
{
    ''')

            generated.append(compile(e));

            for (s : groupParameterclasses) {
                generated.append(s.toString)
            }


            for (s : confBuilder) {
                generated.append(s.toString)
            }
            generated.append("}") //end namespace
            fsa.generateFile("Configurator.cs", generated)
        }
    }
```

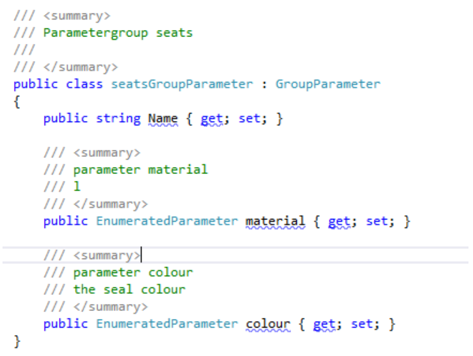
The generator collects codes in three linkedLists.

- GroupParameterClasses contains generated Groupparameter classes.
- parameterInstance contains parameter instanciation.
- confBuilder contains code to create the configuration instance.

# Code generation output

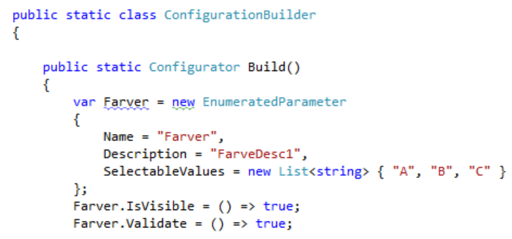
## Structure of the generated code

1.

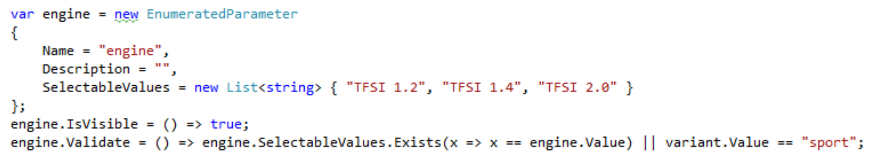


```
/// <summary>
/// Parametergroup seats
///
/// </summary>
public class seatsGroupParameter : GroupParameter
{
    public string Name { get; set; }

    /// <summary>
    /// parameter material
    /// 1
    /// </summary>
    public EnumeratedParameter material { get; set; }

    /// <summary>
    /// parameter colour
    /// the seal colour
    /// </summary>
    public EnumeratedParameter colour { get; set; }
}
```

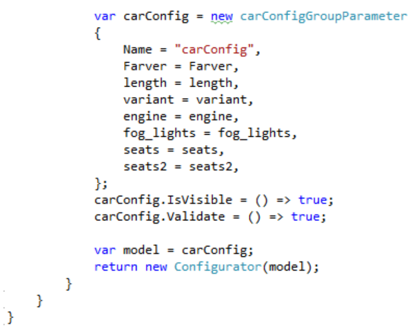
All group parameters are classes, because they all are different from each other.



```
public static class ConfigurationBuilder
{

    public static Configurator Build()
    {
        var Farver = new EnumeratedParameter
        {
            Name = "Farver",
            Description = "FarveDesc1",
            SelectableValues = new List<string> { "A", "B", "C" }
        };
        Farver.IsVisible = () => true;
        Farver.Validate = () => true;
```



```
var engine = new EnumeratedParameter
{
    Name = "engine",
    Description = "",
    SelectableValues = new List<string> { "TFSI 1.2", "TFSI 1.4", "TFSI 2.0" }
};
engine.IsVisible = () => true;
engine.Validate = () => engine.SelectableValues.Exists(x => x == engine.Value) || variant.Value == "sport";
```



```
            var carConfig = new carConfigGroupParameter
            {
                Name = "carConfig",
                Farver = Farver,
                length = length,
                variant = variant,
                engine = engine,
                fog_lights = fog_lights,
                seats = seats,
                seats2 = seats2,
            };
            carConfig.IsVisible = () => true;
            carConfig.Validate = () => true;

            var model = carConfig;
            return new Configurator(model);
        }
    }
}
```

In the static build method, all “value parameters” is created with baseclasses. The base classes is defined in CommenConfig.cs.

All parameters and parametergroups are instantiated so validation and Isvisible methods can reference other parameters. They are all in global scope.

The static method ConfigurationBuilder.Build() creates an instance of the configuration model.