M Kódgenerátor

Simon Balázs, 2011.

Tartalomjegyzék

[1 Bevezetés 2](#_Toc316583967)

[2 Telepítés 2](#_Toc316583968)

[2.1 Rendszerkövetelmények 2](#_Toc316583969)

[2.2 Telepítés 32 bites operációs rendszeren 2](#_Toc316583970)

[2.3 Telepítés 64 bites operációs rendszeren 2](#_Toc316583971)

[2.4 Telepítés ellenőrzése 2](#_Toc316583972)

[2.5 Eltávolítás 32 bites operációs rendszerről 3](#_Toc316583973)

[2.6 Eltávolítás 64 bites operációs rendszerről 3](#_Toc316583974)

[3 Használat 4](#_Toc316583975)

[3.1 Konzol alkalmazás 4](#_Toc316583976)

[3.2 Osztálykönyvtár 5](#_Toc316583977)

[3.3 Kódgenerátor fájl 5](#_Toc316583978)

[4 M Kódgenerátor 6](#_Toc316583979)

[4.1 Template-ek 6](#_Toc316583980)

[4.1.1 Változókra való hivatkozás 6](#_Toc316583981)

[4.1.2 Függvények és más template-ek hívása 7](#_Toc316583982)

[4.1.3 Indentálás 8](#_Toc316583983)

[4.1.4 Új sorok kezelése 8](#_Toc316583984)

[4.1.5 Vezérlés 9](#_Toc316583985)

[4.1.6 Fájlba írás 10](#_Toc316583986)

[4.2 Loop konstrukció 11](#_Toc316583987)

[4.3 Kommentek 13](#_Toc316583988)

[4.4 Using, import és include 13](#_Toc316583989)

[4.5 Konfiguráció 14](#_Toc316583990)

# Bevezetés

Ez a dokumentum a Microsoft M nyelvére épülő kódgenerátort írja le. A telepítési és használati útmutatót követően néhány példán keresztül a generátor nyelve kerül ismertetésre.

# Telepítés

## Rendszerkövetelmények

A környezet telepítéséhez az alábbi szoftvereknek már telepítve kell lenniük:

* .NET Framework 4.0
* Microsoft Visual Studio 2010 Professional or higher
* Microsoft SQL Server Modeling CTP - Nov 2009 Release 3 (formerly Oslo)
  + <http://www.microsoft.com/download/en/details.aspx?id=24113>

## Telepítés 32 bites operációs rendszeren

32 bites operációs rendszeren az alábbi lépéseket kell elvégezni a telepítéshez:

1. Győződjünk meg róla, hogy az **install32.bat** fájlban a **GAC\_DIR** és **OSLO\_DIR** jó helyre   
   mutatnak-e
2. Futtassuk az **install32.bat** fájlt adminisztrátori jogokkal
3. Futtassuk le a **Sb.OsloExtensions.VStudio.vsix** fájlt

## Telepítés 64 bites operációs rendszeren

64 bites operációs rendszeren az alábbi lépéseket kell elvégezni a telepítéshez:

1. Győződjünk meg róla, hogy az **install64.bat** fájlban a **GAC\_DIR** és **OSLO\_DIR** jó helyre   
   mutatnak-e
2. Futtassuk az **install64.bat** fájlt adminisztrátori jogokkal
3. Futtassuk le a **Sb.OsloExtensions.VStudio.vsix** fájlt

## Telepítés ellenőrzése

A telepítés sikerességének ellenőrzéséhez indítsuk el a **Visual Studio**-t majd válasszuk ki a **Tools / Extension Manager…** menüpontot. A megjelenő listában a **Visual Studio Oslo Extensions** csomagnak szerepelnie kell:

## Eltávolítás 32 bites operációs rendszerről

1. Indítsuk el a **Visual Studio**-t
2. Kattintsunk a **Tools / Extension Manager…** menüpontra
3. Válasszuk ki a **Visual Studio Oslo Extensions** csomagot a listából
4. Kattintsunk az **Uninstall** gombra
5. Lépjünk ki a **Visual Studio**-ból
6. Győződjünk meg róla, hogy az **uninstall32.bat** fájlban a **GAC\_DIR** és **OSLO\_DIR** jó helyre   
   mutatnak-e
7. Futtassuk az **uninstall32.bat** fájlt adminisztrátori jogokkal

## Eltávolítás 64 bites operációs rendszerről

1. Indítsuk el a **Visual Studio**-t
2. Kattintsunk a **Tools / Extension Manager…** menüpontra
3. Válasszuk ki a **Visual Studio Oslo Extensions** csomagot a listából
4. Kattintsunk az **Uninstall** gombra
5. Lépjünk ki a **Visual Studio**-ból
6. Győződjünk meg róla, hogy az **uninstall64.bat** fájlban a **GAC\_DIR** és **OSLO\_DIR** jó helyre   
   mutatnak-e
7. Futtassuk az **uninstall64.bat** fájlt adminisztrátori jogokkal

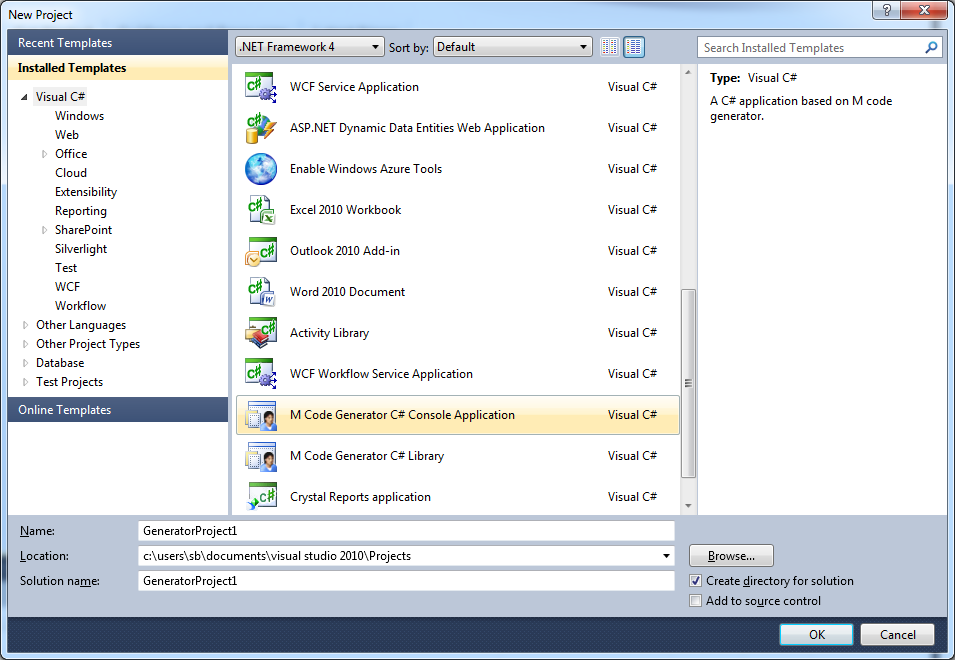
# Használat

A telepített csomagban a következő komponensek szerepelnek:

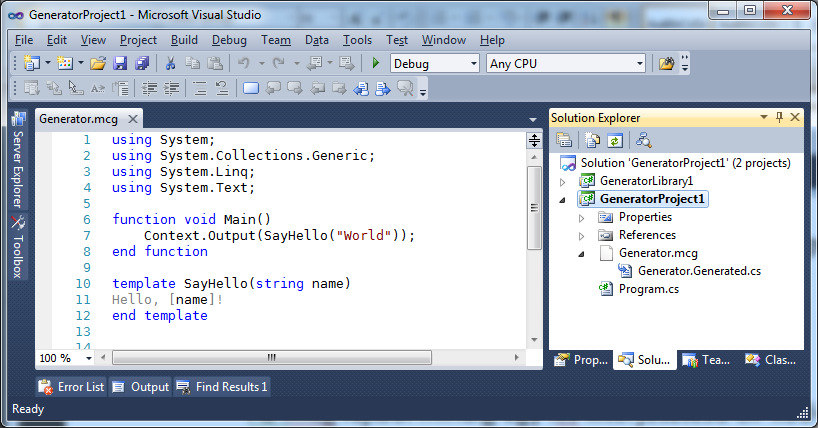
1. Konzol alkalmazás projekt
2. Osztálykönyvtár projekt
3. Kódgenerátor fájl

## Konzol alkalmazás

**File / New Project… / Visual C# / M Code Generator C# Console Application:**



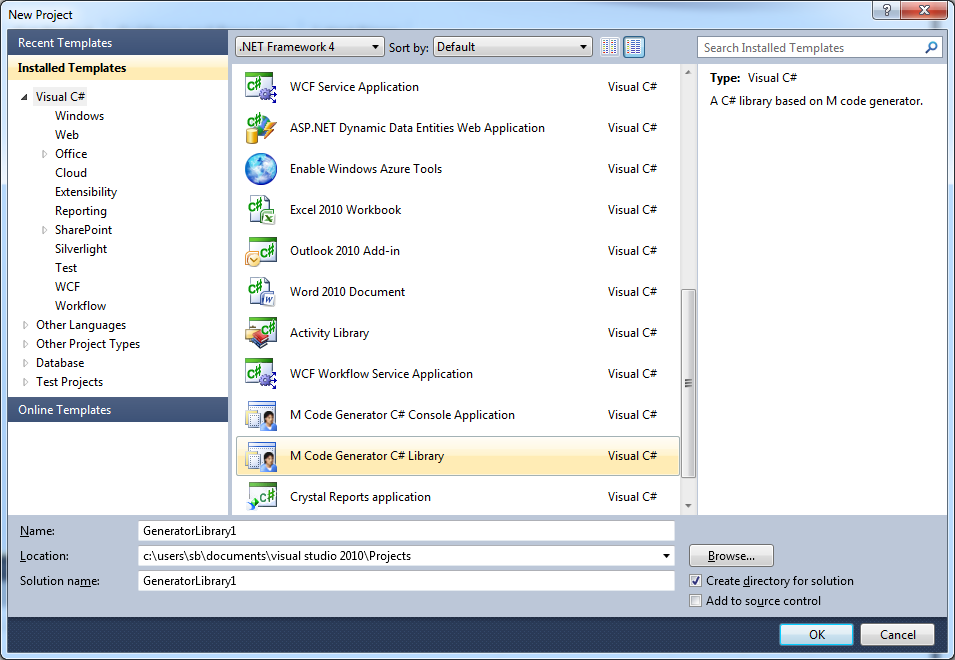
A létrehozást követően az **.mcg** kiterjesztésű kódgenerátor fájlt syntax highlighting segítségével szerkeszthetjük:



Az **.mcg** fájlból mindig egy **.cs** kiterjesztésű C# kód keletkezik, az ebben lévő osztályt kell a főprogramból példányosítani és meghívni.

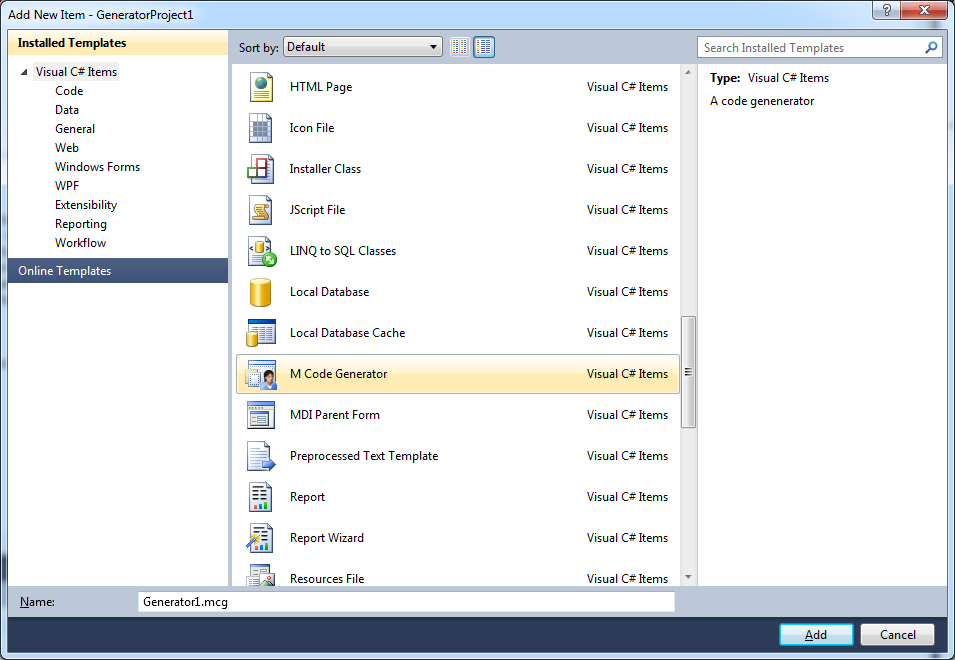
## Osztálykönyvtár

**File / New Project… / Visual C# / M Code Generator C# Library:**



## Kódgenerátor fájl

Egy létező projekthez: **Add / New Item… / Visual C# Items / M Code Generator**



# M Kódgenerátor

Ez a fejezet a kódgenerátort ismerteti példákon keresztül.

## Template-ek

Egy Template-et a template kulcsszóval lehet definiálni:

template <template neve>(<paraméterek>)

...

end template

Minden szöveg, ami a template kódján belül van, közvetlenül a kimenetre megy. A generálást szögletes zárójelek [ ] között lehet vezérelni.

### Változókra való hivatkozás

Változókra való hivatkozást szögletes zárójelek között lehet megtenni. Pl.

function void Main()

Context.Output(SayHello("World"));

end function

template SayHello(string name)

Hello, [name]!

end template

A generátor belépési pontja a void Main() függvény. A generátor meghívásához szükséges C# főprogam:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using OsloExtensions;

namespace GeneratorProject1

{

public class Program

{

public static void Main(string[] args)

{

try

{

List<object> instances = new List<object>();

GeneratorContext context = new GeneratorContext();

Generator generator = new Generator(instances, context);

generator.Execute();

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

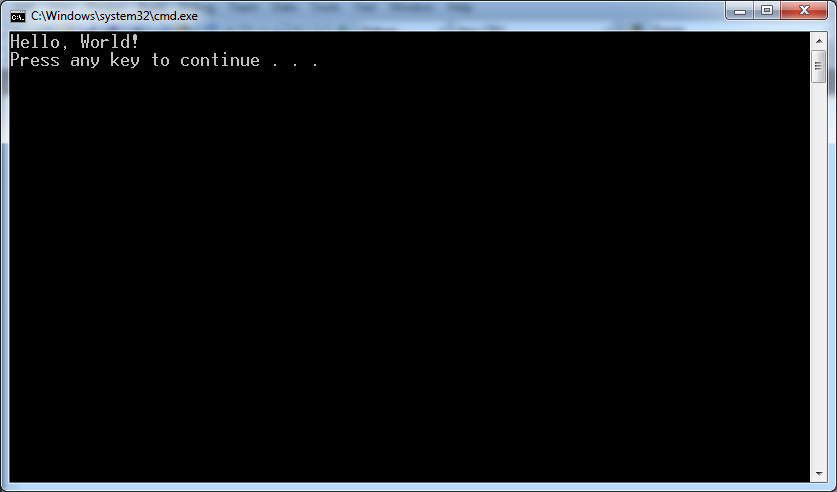
}

}

}

Az instances paraméter a kódgenerátor számára szükséges modellt biztosítja. A generátor ezt a modellt járja be és ez alapján készíti el a kimenetet. A context paraméter a generátor adminisztrációs feladatait látja el, például a fájlok létrehozása, kimenet formázása, stb.

A fenti generátor kimenete:



### Függvények és más template-ek hívása

Függvény vagy másik template hívása szögletes zárójelekben tehető meg:

function void Main()

Context.Output(MainTemplate());

end function

template t1(string a)

template [a]

end template

function string f1(string a)

return "function "+a;

end function

template MainTemplate()

Ez egy példakód, amely függvények és template-ek

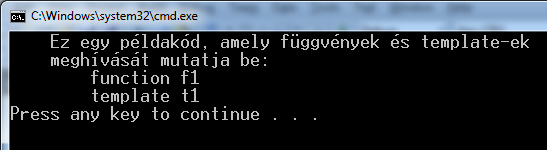
meghívását mutatja be:

[f1("f1")]

[t1("t1")]

end template

A kimenet:



### Indentálás

Ha egy template egy másik template-et hív, az intentálás megőrződik:

function void Main()

Context.Output(t1());

end function

template t1()

t1 - line 1

t1 - line 2

[t2()]

end template

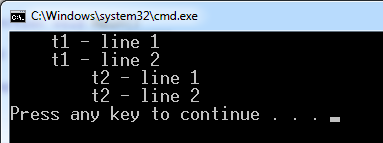
template t2()

t2 - line 1

t2 - line 2

end template

A kimenet:



### Új sorok kezelése

A template-ekben minden sor újsor karakterrel zárul. Ha egy template-ből másik template-et hívunk, akkor a hívott template utolsó sorának végén nem lesz figyelembe véve az újsor karakter:

function void Main()

Context.Output(t1());

end function

template t1()

Start

[t2()]

End

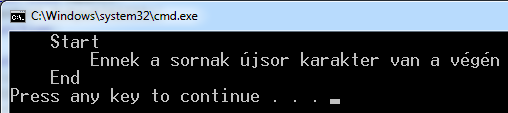
end template

template t2()

Ennek a sornak újsor karakter van a végén

end template

A kimenet:



Az üres sorokat a generátor nem veszi figyelembe. Explicit soremeléshez a kalap ^ karakternek kell a sor végén szerepelnie, illetve a soremelés megakadályozásához a visszaper \ karakter használandó:

function void Main()

Context.Output(t1());

end function

template t1()

Apple

^

Pear

Green

[empty()]

Blue

First, \

Second, \

Third

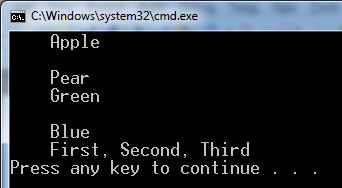
end template

template empty()

^

end template

A kimenet:



### Vezérlés

A feltételes kifejezések szintaxisa a következő:

if (feltétel)

...

end if

Vagy:

if (feltétel)

...

else

...

end if

Illetve:

if (feltétel1)

...

else if (feltétel2)

...

else if (feltétel3)

...

else

...

end if

Amennyiben ezeket vagy más vezérlő szerkezeteket template-en belül használunk, szögletes zárójelbe kell őket tenni:

function void Main()

Context.Output(t1(""));

Context.Output(t1("b"));

end function

template t1(string a)

start

[if (a == "b")]

[t2()]

[end if]

end

end template

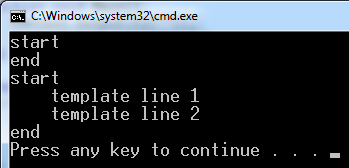
template t2()

template line 1

template line 2

end template

A kimenet:



A vezérlést célszerű mindig külön template sorba helyezni, különben a kimenet formátuma nem feltétlenül felel meg a várakozásoknak.

### Fájlba írás

A Context.SetOutput hívás segítségével a kimenetet fájlba irányíthatjuk, például:

function void Main()

Context.SetOutput("sample.txt");

Context.Output(Template());

end function

template Template()

Hello, World!

end template

## Loop konstrukció

A generátor legnagyobb erőssége a modellbejárás hatékony támogatása. A loop konstrukció segítségével egy sorban leírható sok egymásba ágyazott for-ciklus.

Vegyük a következő főprogramot:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using OsloExtensions;

using GeneratorLibrary1;

namespace GeneratorProject1

{

class A

{

public virtual string Name { get { return "A"; } }

}

class AB : A

{

public override string Name { get { return "AB"; } }

}

class AC : A

{

public override string Name { get { return "AC"; } }

}

class X

{

public virtual string Name { get { return "X"; } }

}

public class Program

{

public static void Main(string[] args)

{

try

{

List<object> instances = new List<object>();

instances.Add(new X());

instances.Add(new A());

instances.Add(new AB());

instances.Add(new AC());

instances.Add(new X());

GeneratorContext context = new GeneratorContext();

Generator generator = new Generator(instances, context);

generator.Execute();

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

A generátorból az Instances változón keresztül lehet elérni a modellt:

function void Main()

Context.Output(Template());

end function

template Template()

[loop(Instances->a:typeof(A))]

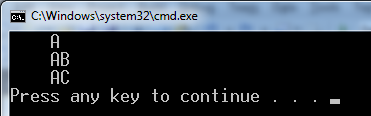
[a.Name]

[end loop]

end template

A nyíl operátor végigiterál a bal oldalán lévő kollekción és visszaadja a jobb oldalon specifikált elemeket. A typeof(A) segítségével választhatjuk ki a megfelelő típusú objektumokat. A kettőspont operator a loop bármelyik eleméhez csatolható, ennek segítségével elnevezhetjük az objektumokat és ezen a néven hivatkozhatunk rájuk a loop törzsében.

A fenti példa kimenete:



A loop fejlécében feltételes kifejezéseket is megadhatunk:

function void Main()

Context.Output(Template());

end function

template Template()

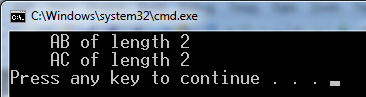
[loop(Instances->a:typeof(A) where a.Name.Length >= 2)]

[a.Name] of length [a.Name.Length]

[end loop]

end template

A kimenet:



A loop elemeit rendezni is lehet, például:

template Template()

[loop(Instances->a:typeof(A) where a.Name.Length >= 2 orderby a.Name)]

[a.Name] of length [a.Name.Length]

[end loop]

end template

Vagy pedig visszafelé sorrendben:

template Template()

[loop(Instances->a:typeof(A) where a.Name.Length >= 2 orderby a.Name descending)]

[a.Name] of length [a.Name.Length]

[end loop]

end template

A loop-nak is lehet else ága. Ez akkor fut le, ha a loop törzse sosem futna le. Például:

template Template()

[loop(Instances->a:typeof(A))]

[a.Name]

[else]

noname

[end loop]

end template

A loop végrehajtása során lehetőség van parancsokat is lefuttatni az aktuális ciklus sorszámától függően. Ehhez a loop kifejezés után pontosvesszővel elválasztva kell megadni a parancsokat. Az első pontosvessző utáni parancs a loop első lefutásakor jut érvényre, a második pontosvessző utáni a második lefutáskor, stb. A legutolsó pontosvessző utáni parancs a maradék lefutások mindegyikében végrehajtódik. Például egy operáció paraméterlistájának kigyűjtése úgy, hogy közöttük vessző a szeparátor:

function string GetParams(Operation op)

string result = "";

loop(op->p:Parameters; string delimiter = ""; delimiter = ", ")

result = result + delimiter + p.Name;

end loop

return result;

end function

## Kommentek

A forráskódban lehetőség van a Java-ban és C#-ban megszokott kommentek elhelyezésére. Template-ben nem szerepelhetnek kommentek, mindenhol máshol igen.

Például:

/\*

This is the Main function.

\*/

function void Main()

// Saying hello:

Context.Output("Hello World!");

end function

## Using, import és include

A .NET névterek használatához a using kulcsszót lehet alkalmazni:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

Egy másik kódgenerátor beimportálása az import kulcsszóval történik. A másik generátor függvényeit és sablonjait kvalifikálni kell az importált generátor nevével. Például:

import Generator1;

function void Main()

Context.Output(**Generator1.**SayHello("World"));

end function

## Konfiguráció

A generátor konfigurálása a using/import/inlcude rész után közvetlenül megadott configuration szekcióban történik. Például:

configuration

string NamespaceName = "MyNamespace";

string ClassName = "MyGenerator";

Type InstancesType = typeof(List<object>);

Type ContextType = typeof(GeneratorContext);

bool UseMcgLineNumbers = false;

properties MyProperties

string ProjectName = "DefaultProjectName";

bool GenerateBuildFile;

properties MyPropertyGroup

int IntProperty = 100;

end properties

end properties

end configuration

Az első öt sor rendre a következőket konfigurálja:

* az MCG kódból generált C# osztály névtere
* az MCG kódból generált C# osztály neve
* a generátorban a modell példányait tartalmazó lista típusa (a generátorból Instances néven lehet rá hivatkozni)
* a generátor kontextusa (a generátorból Context néven lehet hivatkozni)
* az MCG kódból generált C# osztályban szerepeljenek-e #line jelzések, amelyek segítségével a Visual Studio hibajelzése az MCG kódban fog megjelenni a generált C# kód helyett

Ez az öt beállítás opcionális, azonban a sorrendjük kötött.

Ha több különböző generátor fájlban ugyanazt a névteret és osztálynevet adjuk meg, akkor is helyes működést kapunk, ugyanis az MCG fájlból előállított C# osztályok partial class-ok, így egymás függvényeit meghívhatják. Arra azonban ügyelni kell, hogy a függvénynevek ne ütközzenek.

Az Instances és a Context típusát csak a Main() függvényt tartalmazó generátor rész veszi figyelembe, minden más esetben a konfigurációnak ez a két pontja nem érvényesül.

A properties rész arra szolgál, hogy a generátort kívülről lehessen paraméterezni. A properties kulcsszóval egy property csoportot lehet nyitni, a kulcsszó után meg kell adni a csoport nevét. A csoporton belül létrehozhatunk típus és név megadásával property-ket (rendelhetünk hozzájuk default értéket is), illetve hierarchikusan nyithatunk újabb property csoportokat.

A főprogramból így férhetünk hozzá a property-khez:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using OsloExtensions;

using MyNamespace;

namespace GeneratorProject1

{

public class Program

{

public static void Main(string[] args)

{

GeneratorContext context = new GeneratorContext();

MyGenerator generator = new MyGenerator(null, context);

generator.MyProperties.GenerateBuildFile = true;

generator.MyProperties.MyPropertyGroup.IntProperty = 200;

generator.Execute();

}

}

}

A generátorból magából pedig a következőképpen:

function void Main()

Context.Output(SayHello(MyProperties.ProjectName));

end function

template SayHello(string name)

Hello, [name]!

end template

Az eredmény:

