Using Code OnTime Generator: Getting Started

Code OnTime Generator is a general purpose code generator for *Microsoft.NET*.

Most code generators on the market are based on a proprietary template technology modeled after *ASP.NET*. A typical code generator template is a code file written in a target programming language with embedded markup tags that contain code generator internal programming language statements and expressions. These statements are manipulating input parameters and data files to produce the dynamic portion of the template text when code generator processes the template.

This model is well understood and quite popular.

There are few weaknesses of this approach to code generation:

1. You are forced to use procedural approach to generate your programs. Every single aspect of the output program text must be explicitly defined in your template, which creates enormous complexity.
2. One template file supports only one programming language. If you do want to produce your code in more than one programming language then you have to create two separate templates, which usually turn out to be a maintenance nightmare.
3. It is very difficult to produce a well formatted output. If you ever tried to view *HTML* code generated by *ASP.NET* web forms then you would probably agree that it does not look pretty. Luckily we have amazing web browsers to sift through the mess and produce a pretty picture for us. You have to make sense of the mess and be a "browser" when you deal with program text produced by code generator. The code may work great but is usually a pain to read and work with.
4. Proprietary data models used by code generator projects are difficult to alter. You will do well if you access the model via standard objects built into code generator but any customization must be done by design tools that come with code generator.
5. Design environment of a typical code generator is inflexible and may or may not reflect your expectations. You are at the mercy of the vendor when the time comes to ask for improvements.

Code OnTime Generator eliminates these limitations.

Industry standard data presentation language *XML* is the data model language of code generation projects. Industry standard *XPath* and *XSLT* languages are natural companions to this data model language. Both, *XPath* and *XSLT* bring flexibility, power, and a bit of a magic in code generation process. *Java Script* and *DHTML* (also referred to as *AJAX)* are quite popular in a development community and are proving every day that there are very few things that you cannot do when you are building modern user interfaces with their help. Both technologies are in the foundation of the code generator design environment.

Here is how these technologies stack up against limitations that are found in a typical code generator.

Declarative Programming Model

XSLT is the best template technology out there designed specifically to take advantage of the flexibility built into *XML*. If you describe your project code generation requirements as an *XML* file then there is not a single piece of such specification, which is not a few key strokes away from the template developer writing *XPath* expressions. *XPath* is to *XML* what *SQL* is to *databases*. Learn to say what you want and *XPath* engine will get it for you.

Compare that with the laborious procedural statements and expressions in a typical code generator that you have to meticulously craft to get the data that you want. Imagine that you are writing an application that works with the database. What if instead of saying *select CustomerName from Customers where CustomerID = 'ABC'* you are forced to write code that opens a database file, opens the primary key index, creates a cursor, moves the cursor until your reach the desired record, declares a buffer to read the data out, and only then you can start using the result. Sounds like a nightmare. Typical code generator templates are going through similar pains when code based on any complex data model is to be generated.

Instead of writing code in a specific target language you declare the XML constructs that your program must be built from. Consider the following declaration.

```
<variableDeclarationStatement type="DbProviderFactory" name="factory">
  <init>
    <methodInvokeExpression methodName="GetFactory">
      <target>
        <typeReferenceExpression type="DbProviderFactories"/>
      </target>
      <parameters>
        <primitiveExpression value="{$DbProviderName}"/>
      </parameters>
    </methodInvokeExpression>
  </init>
</variableDeclarationStatement>
```

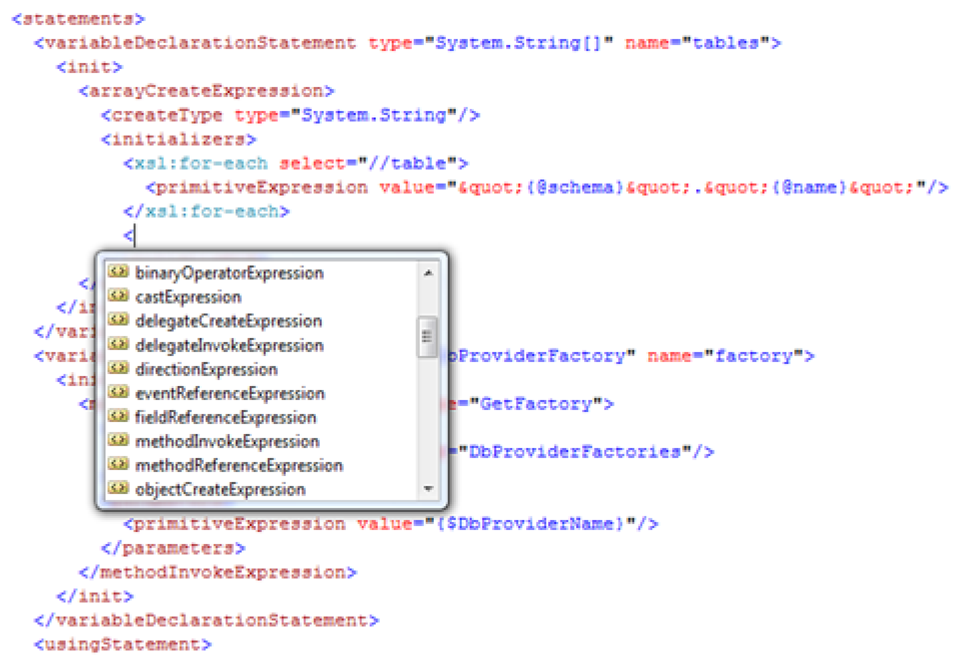
It sure reads like a book.

I am declaring a variable of type *DbProviderFactory* named *factory* and initializing it by invoking method *GetFactory* while targeting type *DbProviderFactories* and passing on parameter in a form of a primitive expression that drives its values from *XSLT* variable *DbProviderName*.

The result in a *C#* program will look like the one below if *DbProviderName* variable is equal to *System.Data.SqlClient*.

```
DbProviderFactory factory =
DbProviderFactories.GetFactory("System.Data.SqlClient");
```

Best of all you can rely on the power of *Visual Studio* to guide you in your quest of writing a code generation template. Complete *intellisense* support makes the process very simple.


```
<statements>
  <variableDeclarationStatement type="System.String[]" name="tables">
    <init>
      <arrayCreateExpression>
        <createType type="System.String"/>
        <initializers>
          <xsl:for-each select="//table">
            <primitiveExpression value="&quot;{@schema}&quot;.&quot;{@name}&quot;"/>
          </xsl:for-each>
          <
      </
    </i
  </var
  <vari                                          bProviderFactory" name="factory">
    <ini
      <                                         e="GetFactory">

                                               ="DbProviderFactories"/>

          <primitiveExpression value="{$DbProviderName}"/>
        </parameters>
      </methodInvokeExpression>
    </init>
  </variableDeclarationStatement>
  <usingStatement>
```

Code OnTime Generator features an automatic processor of output program code declarations that are translated with the help of *System.CodeDom* code generation library built into *Microsoft.NET*.

Support for Multiple Languages

Dependency on *System.CodeDom* provides a benefit of being able to generate the output code in any programming language supported by *Microsoft.NET*. We go a step further in overcoming some of the limitations of the *System.CodeDom* code generator and are providing additional constructs and full support for many language features for two programming languages: *Visual C#* and *Visual Basic.NET*.

High Quality Formatting

The produced code comes out with almost perfect formatting. You will likely say that it does look as if a human being has been crafting the lines of the output program text. Automatic formatting saves huge amount of time and gives you a peace of mind to known that the final code will come out alright.

Open User-Defined Data Model

There is a sample project *Hello World* that was designed to showcase the capabilities of the code generator. Here is project data model that can be found in *HelloWorld.Project.xml* file.

```
<?xml version="1.0" encoding="utf-8" ?>
<project xmlns="urn:schemas-codeontime-com:hello-world-project">
  <providerName>System.Data.SqlClient</providerName>
  <connectionString></connectionString>
</project>
```

The project will generate a program that opens a connection to any supported database, read its content, and print out a number of records in the database tables. All we need is the name of *ADO.NET* provider and a connection string. Absolutely arbitrary element names are included in our *XML* project database. There are no mandates or requirements that are imposed on your data by any features of the code generator. You decide what sort of data will be in the foundation of your code generation project.

Unlimited Flexibility Of Design Environment

Each project has file CodeOnTime.Project.xml that defines the steps that code generator must perform to produce the output. Here is a snippet from *Hello World* project.

```
<load path="$ProjectPath">
  <if test="not(file/@name='HelloWorld.Project.xml')">
    <copy input="$LibraryPath\HelloWorld.Project.xml"
          output="HelloWorld.Project.xml"/>
  </if>
</load>
<!-- opens the main project page -->
<navigate url="HelloWorld.Start.htm"/>
```

In this snippet the code generator will load the list of files in the output project folder.

If there is no *HelloWorld.Project.xml* there then the file is copied from the library folder to the project folder.

Next the project build script directs code generator to load *HelloWorld.Start.htm* file located in the project library folder.

Our code generator relies on *Java Script* and *DHTML*. Build-in web browser loads project pages and completely relies on this pages to display an appropriate user interface, update the data model, and signal when *navigate* step is over. All pages are relying on the standard *Microsoft.Ajax.js* library, which brings significant improvements in making it simpler to build UI with Java Script.

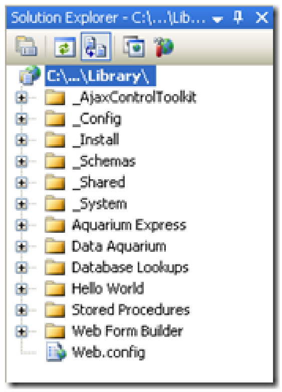
All code generation templates and included "as is" and can be used as a foundation for your own projects.

Code Generation Primer

Project *Hello World* is provided to give you a head-start with your own code generation ideas. Here is how you start development.

Run *Visual Studio 2008* or *Visual Web Developer 2008 Express Edition* and select *File|Open Web Site* command in the menu. Browse to *[My Documents]\Code OnTime\Library* folder and open it as if it were a web site.

Visual Studio will suggest converting this "site" to the web site that can run with *Microsoft.NET 3.5*. Press *OK* button. The *Solution Explorer* will display a similar set of files:



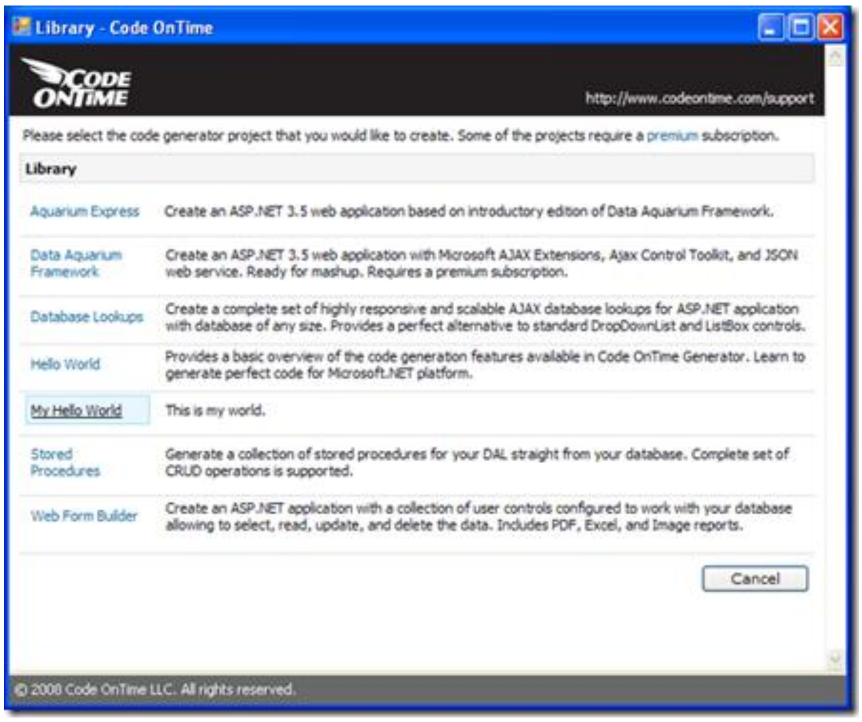
Files that start with underscore symbol are collections of system files that provide *XSD* schemas, shared *HTML* snippets and system *Java Script* files. One of the folders holds the entire *AjaxControlToolkit* in it, which is being reused in a few projects.

File *Web.config* has been created by *Visual Studio* and can be ignored.

Copy any project that you would like to use as your own template and paste it into the root of the library. Open *CodeOnTime.Project.xml* file and make changes at the top of the project to reflect the project name, and description. Also come up with a product ID of your own.

```xml
<information>
  <product>CNT200809-MHW</product>
  <title>My Hello World</title>
  <description>This is my world.</description>
  <vendor>Contoso</vendor>
  <website>http://www.contoso.com</website>
</information>
```

Run *Code OnTime Generator* and start creating a new project.



Please select the code generator project that you would like to create. Some of the projects require a premium subscription.

Library	
Aquarium Express	Create an ASP.NET 3.5 web application based on introductory edition of Data Aquarium Framework.
Data Aquarium Framework	Create an ASP.NET 3.5 web application with Microsoft AJAX Extensions, Ajax Control Toolkit, and JSON web service. Ready for mashup. Requires a premium subscription.
Database Lookups	Create a complete set of highly responsive and scalable AJAX database lookups for ASP.NET application with database of any size. Provides a perfect alternative to standard DropDownList and ListBox controls.
Hello World	Provides a basic overview of the code generation features available in Code OnTime Generator. Learn to generate perfect code for Microsoft.NET platform.
My Hello World	This is my world.
Stored Procedures	Generate a collection of stored procedures for your DAL straight from your database. Complete set of CRUD operations is supported.
Web Form Builder	Create an ASP.NET application with a collection of user controls configured to work with your database allowing to select, read, update, and delete the data. Includes PDF, Excel, and Image reports.

© 2008 Code OnTime LLC. All rights reserved.

You project will show up right there. Any future code generation library updates will not affect your project.

You can start making immediate improvements with full intellisense support of *Visual Studio*. Intellisense and hints are available in *XSLT stylesheets*, in *JavaScript* libraries, in *codedom* files, and in project build file.

If you are working on your own project and have a question that requires an answer then please don't hesitate to drop us a message at http://codeontime.com/contactus.aspx. We are here to help