Blade User Guide v1.0

Table of Contents

Introduction

Blade provides a web client development environment from within Visual Studio 2010. You can use Blade to write C# code, which will be translated to JavaScript compatible with modern browsers. A runtime API is provided for coding against the DOM, HTML elements, and other browser constructs, such as the History or Console objects. The Blade runtime API does not abstract the core ECMA, DOM, or HTML APIs. In fact, in many cases your C# code will be exactly the same as its equivalent JavaScript syntax. For those who already have a strong JavaScript and client development background, coding in Blade will feel normal. And for those who do not, Blade will help build your knowledge through tools such as Intellisense which can help you discover and understand these core APIs. You'll get useful comments for objects, functions, and parameters within the IDE.

As mentioned above, Blade does not abstract the core ECMA, DOM, and HTML APIs. This is an important distinction between regular .NET development that you should keep in mind. Blade does not provide a .NET runtime for the client. This means that many of the APIs you're used to in .NET may not be available with the Blade development context. While many portions of JavaScript have equivalent syntax in C#, remember that JavaScript is a loosely typed language and has many capabilities that are difficult or impossible to accomplish in C# syntax. This is also true inversely, as there are several C# features and keywords that do not have viable JavaScript translations. Because of this, you will inevitably encounter cases where Blade intentionally limits some features of C#. In most cases there are alternative solutions which build on the strengths of both languages. For example, JavaScript does not directly support the concept of method overloading. If you attempt to overload a method, you'll get a build error noting that method overloading is not supported. However, there are some great alternatives, such as optional and named parameters. It's important to keep in mind as you use Blade that you're writing code for the client.

This of course begs the question: with these restrictions applied why would you want to use Blade? Why not just write standard JavaScript? While answers to this may vary between developers, there are many aspects of Blade that can greatly increase productivity. These include C# features such as using a class-based syntax, with inheritance and support for interfaces, as well as Visual Studio features such as refactoring, Intellisense, and build-time error feedback. Blade is a good candidate for use within large and relatively complex client frameworks, which may contain substantial amounts of code developed and maintained by multiple individuals. It's also ideal for developing reusable components that may be shared across many areas of a single application, or entirely separate applications. Ultimately, Blade can help to enforce better structure throughout your codebase and allow you to focus more on architecture and design.

Installation

Blade can be installed by either running the installer, or building the Blade solution file. If you choose to run the installer, please ensure that you first uninstall any existing version of Blade that may be installed on your system. Next simply run the installer and follow the wizard. This will install the Blade .NET Profile, runtime libraries, and Visual Studio project templates.

Alternatively, if you choose to clone the project and build locally, the required files will be copied to their respective installation locations by project post-build scripts. Please note that while the installer supports both 32 and 64-bit development environments, the project post-build scripts are intended to be executed on a 64-bit environment. Cleaning the project files does not remove the deployed files. If you'd like to manually remove these files, use the information provided below.

The project files are copied out as follows:

- **BLADE PROFILE**

  The profile exists as a .NET Framework 4.0 Profile under the Reference Assemblies directory. For more information of this directory, see the following blog post from the MSBuild Team Blog in 2007: http://blogs.msdn.com/b/msbuild/archive/2007/04/12/new-reference-assemblies-location.aspx

  Blade.Runtime.mscorlib – Copies the mscorlib assembly and XML documentation files.
  Blade.Runtime.SystemCore – Copies the System.Core assembly and XML documentation files.
  Note that the FrameworkList.xml file is also copied from the Resources folder.

- **BLADE COMPILER**

  The script compiler and associated files are copied to the MSBuild extensions directory.

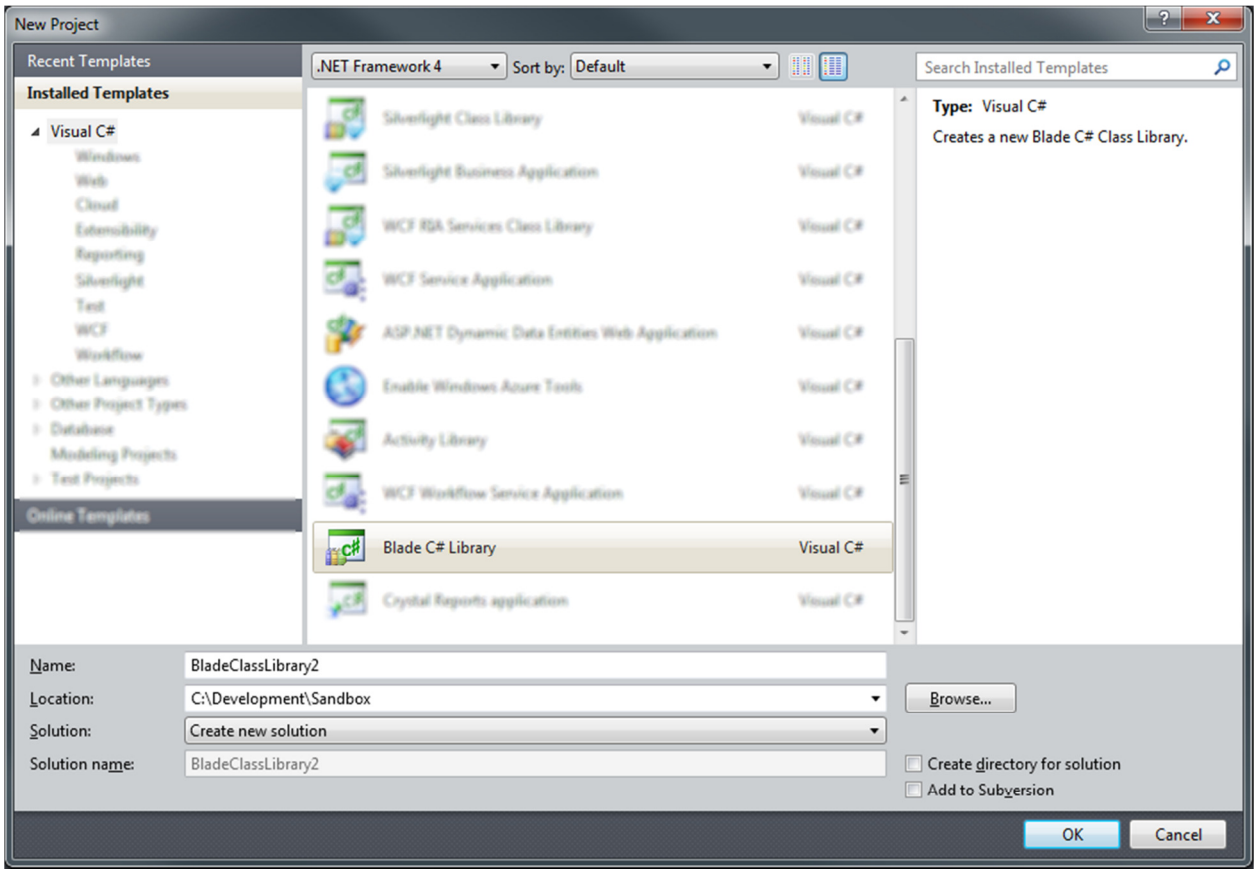
  Blade.Compiler – Copies the Blade.Compiler assembly.
  Blade.Build – Copies the Blade.Build assembly and Blade.targets file.

- **VS TEMPLATES**

  The Blade project and item templates are not automatically copied out as part of the build process. However these can be installed by running the output of the VSIX project.

Creating a Project

Once the project template is deployed properly, you may create a new Blade class library by simply selecting Blade C# Library from the list of Visual C# templates.



Similar to a normal C# Class Library, Blade creates a project with a single class named Class1. To add new classes through the Add Item dialog, you should select the Blade Class item. This will create a class file similar to the standard C# Class item, except with different set of initial namespace directives.

Blade Profile

The Blade profile contains a custom implementation of the mscorlib and System.Core assemblies. These may be referred to as the runtime libraries. They provide a definition of the core APIs available in the browser. All .NET 4.0 projects, including Blade projects, link with mscorlib.dll and System.Core.dll by default.

The browser APIs are exposed as follows:

- **GLOBAL NAMESPACE**

  The global namespace provides access to the browser's Window, Console, History, Navigator, Location, and Screen objects.

- **SYSTEM NAMESPACE**

  The System namespace contains the class definitions for ECMA Date, Error, Function, Math, Number, and RegExp objects.

- **SYSTEM.BROWSER NAMESPACE**

  The Browser namespace contains the class definitions for the Window, History, Location, and Navigator objects.

- **SYSTEM.CSS NAMESPACE**

  The Css namespace contains the class definitions for items in the W3 Cascading Style Sheets Object Model. For details, see: http://www.w3.org/TR/2011/WD-cssom-20110712/

- **SYSTEM.DOM NAMESPACE**

  The Dom namespace contains the class definitions for items in the W3 Document Object Model Core, Events, Range, and Traversal Specifications. For details, see: http://www.w3.org/TR/2012/WD-dom-20120405/ and http://www.w3.org/TR/DOM-Level-3-Events/

- **SYSTEM.HTML NAMESPACE**

  The Html namespace contains the class definitions for items in the W3 HTML5 Specification. For details, see: http://www.w3.org/TR/2012/WD-html5-20120329/

Extension Attributes

Blade provides the ability to alter generated script from its C# representation by using .NET Attribute classes. Several of these special attributes are available out of the box. These include:

- **[ScriptExternal]**

  The ScriptExternal attribute can be used to denote that a declaration exists externally and should not be generated in the output script. This attribute may be applied to Class, Field, Method, or Property declarations to prevent generation of specific items. It may also be applied to the entire assembly, which will result in the project generating no script.

- **[ScriptName]**

  The ScriptName attribute is used to alter the name of a type or member definition in the generated script. This attribute may be applied to Class, Interface, Field, Method, and Property declarations.

- **[ScriptNamespace]**

  The ScriptNamespace attribute is used to alter the namespace of a type definition in the generated script. This attribute may be applied to Class declarations. It may also be applied to the entire assembly, which results in all classes using the same namespace.

- **[ScriptIgnoreNamespace]**

  The ScriptIgnoreNamespace attribute is used to ignore the namespace of a type definition in the generated script. It has the same effect as using [ScriptNamespace("")] and may also be applied to Class declarations or an entire assembly.

- **[ScriptField]**

  The ScriptField attribute can be used to render properties as fields in the generated script. This attribute may be applied only to Property declarations.

Feedback

Your feedback is important and useful for shaping the future of Blade. We encourage you to submit questions, suggestions, and bug reports. Feel free to submit any of these items to the Blade github project issues page at: https://github.com/vannatech/blade/issues

For general inquiries you may also email the author at: barry.dorman@vannatech.com