Building, Integrating, and Deploying ClearScript

# Introduction

Welcome to ClearScript!

ClearScript is a library that allows you to add scripting to your .NET applications. It supports [JScript](https://msdn.microsoft.com/en-us/library/hbxc2t98(v=vs.84).aspx) and [VBScript](https://msdn.microsoft.com/en-us/library/t0aew7h6(v=vs.84).aspx) out of the box and in theory can work with other [Windows Script](http://msdn.microsoft.com/en-us/library/bb871518.aspx) engines.

ClearScript 5 adds support for the [V8](https://developers.google.com/v8/) high-performance open-source JavaScript engine. It allows you to use V8 with the same managed API and host integration features as JScript and VBScript. V8 provides better performance, however, and is more suitable for multi-threaded applications and asynchronous server-side scripting.

# ClearScript NuGet package

Now that an [official ClearScript NuGet package](https://www.nuget.org/packages/Microsoft.ClearScript) is available, you can simply add it to your project and skip to [Section VII](#_Debugging_with_ClearScript_2) below.

**Important:** The ClearScript NuGet package requires the Visual C++ Redistributable libraries. Download and install both the 32-bit and 64-bit versions: [[x86]](https://aka.ms/vs/16/release/vc_redist.x86.exe) [[x64]](https://aka.ms/vs/16/release/vc_redist.x64.exe)

**Important:** If you’re adding the ClearScript NuGet package to an ASP.NET project, you must exclude ClearScript’s mixed-mode assemblies from ASP.NET compilation. To do so, merge the following into your **Web.config** files:

<system.web>

<compilation>

<assemblies>

<remove assembly="ClearScriptV8-64" />

<remove assembly="ClearScriptV8-32" />

</assemblies>

</compilation>

</system.web>

# Building ClearScript

The ClearScript source code is hosted in a [Git](http://www.git-scm.com/download/win) repository. If you’re installing Git for the first time, select the options **Use Git from the Windows Command Prompt** and **Checkout Windows-style, commit Unix-style line endings**. If you already have Git installed, use [git-config](https://git-scm.com/docs/git-config) to ensure that the **core.autocrlf** variable is set to **true**.

Use [git-clone](https://git-scm.com/docs/git-clone) to download the [ClearScript source code](https://github.com/Microsoft/ClearScript) into a convenient directory.

The provided project and solution files support [Visual Studio 2019](https://www.visualstudio.com/downloads/) Version 16.4 or later. They produce architecture-neutral managed libraries that target .NET Framework 4.5 and .NET Core 3.1. ClearScript does not support older environments. The output directories are **bin\[Debug|Release]** and **bin\[Debug|Release]\netcoreapp3.1**.

There are two ways to build ClearScript – with and without V8 support. If you don't need V8 support, simply build the **ClearScript.NoV8** solution using Visual Studio. Note that this solution does not include test projects.

In order to build full ClearScript with V8 support, you must first acquire, build, and import V8:

1. **Note:** This procedure and the V8Update script are provided for your convenience. ClearScript does not include V8 source code, nor does it come with any third-party software required to download and build V8. Rights to V8 and its dependencies are provided by their respective rights holders.
2. **Important:** Ensure that the path to your ClearScript root directory does not contain spaces or non-ASCII characters.
3. **Important:** The V8 build now requires a 64-bit operating system. Your Visual Studio installation must include the **.NET desktop development** and **Desktop development with C++** workloads. Once built, ClearScript can still be deployed in a 32-bit environment.
4. **Important:** Your Windows SDK installation must include the **Debugging Tools for Windows** feature. Here’s how to add it to your installation:
   1. Launch Control Panel and click **Programs** → **Programs and Features**.
   2. Right-click your Windows SDK version and select **Change**.
   3. Click **Next**, check **Debugging** **Tools** **for** **Windows**, and click **Change**.
5. Open a [Visual Studio 2019](https://www.visualstudio.com/downloads/) developer command prompt and run the **V8Update** script from your ClearScript root directory:

**C:\ClearScript> V8Update [/N] [Debug|****Release] [Latest|Tested|<Revision>]**

This script downloads the V8 source code and its dependencies, builds 32-bit and 64-bit V8 shared libraries, and imports the results into ClearScript. It requires approximately 8GB of additional disk space and does not perform any permanent software installation on your machine.

The optional **/N** flag causes V8Update to reuse previously downloaded files if possible. It's useful for switching between Debug and Release versions of V8 and for testing local V8 modifications.

Specifying **Debug** or **Release** is optional; the default is **Release**. The selected V8 variant will then be used for all ClearScript configurations.

By default, **V8Update** builds a V8 revision that has been tested with the current version of ClearScript. If you'd like to use a specific revision instead, place the desired branch name, commit ID, or tag on the **V8Update** command line. Browse [here](https://chromium.googlesource.com/v8/v8.git) to view V8's revision history.

You are now ready to build the full **ClearScript** solution using Visual Studio 2017 or 2019.

**Note:** The first time you open the solution, Visual Studio may prompt you to upgrade one or more projects to the latest platform toolset or .NET Framework. We recommend that you select **Cancel** or **Don't Upgrade**. If you encounter issues building ClearScript’s unmanaged projects (**ClearScriptV8-32** and **ClearScriptV8-64**), ensure that their **Windows SDK Version** properties are set to an installed version of the Windows SDK.

**Optional:** The ClearScript repository includes the [ClearScript Library Reference](https://microsoft.github.io/ClearScript/Reference/html/R_Project_Reference.htm) in HTML and Compiled HTML (.CHM) formats. If you'd like to rebuild these files, use [Sandcastle Help File Builder](https://github.com/EWSoftware/SHFB) with the provided project files (**ClearScript\doc\[Web]Reference.shfbproj**).

# Building strong-named ClearScript assemblies (optional)

ClearScript now includes optional support for building strong-named assemblies. Use the following one-time procedure to enable this feature:

1. If the ClearScript solution is open in Visual Studio, close it.
2. Generate a cryptographic key pair in your ClearScript root directory:

**C:\ClearScript> sn -k ClearScript.snk**

1. Open the ClearScript solution in Visual Studio.
2. Click **Build** → **Transform All T4 Templates**.
3. Rebuild the solution.

Once you've performed these steps, the ClearScript assemblies you build will have strong names.

# Integrating and deploying ClearScript with your application

Once you've built ClearScript, here's how to add it to your application:

1. Right-click your project in Visual Studio and select **Add Reference**.
2. In the Reference Manager window, click **Browse** and locate your ClearScript output directory (see above). Select **ClearScript.dll**, click **Add**, and then click **OK** to exit Reference Manager.
3. **Important:** If you're using V8, you must also copy the following files from your ClearScript output directory to your application's directory:

**ClearScriptV8-32.dll**

**ClearScriptV8-64.dll**

**v8-ia32.dll**

**v8-x64.dll**

**v8-base-ia32.dll**

**v8-base-x64.dll**

**v8-libcpp-ia32.dll**

**v8-libcpp-x64.dll**

For ASP.NET projects, we recommend that you add these assemblies as content files at the root of your web application and set their **Copy to Output Directory** properties to **Do not copy**.

1. **Important:** If Visual Studio is not installed on your deployment machine, you must install both 32-bit and 64-bit Visual C++ Redistributable packages:

Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019: [[x86]](https://aka.ms/vs/16/release/vc_redist.x86.exe) [[x64]](https://aka.ms/vs/16/release/vc_redist.x64.exe)

# Machine-level (global) deployment (.NET Framework only)

You can now deploy ClearScript at the machine level, making it available to all locally installed applications. For this to work, your ClearScript assemblies must have strong names. If you’re building ClearScript yourself, see [Section IV](#_Building_strong-named_ClearScript) for more information. The [official ClearScript NuGet package](https://www.nuget.org/packages/Microsoft.ClearScript) contains strong-named assemblies.

To deploy ClearScript globally, install its managed assemblies in the [Global Assembly Cache](https://docs.microsoft.com/en-us/dotnet/framework/app-domains/gac):

**C:\ClearScript\bin\Release> gacutil /i ClearScript.dll**

**C:\ClearScript\bin\Release> gacutil /i ClearScriptV8-32.dll**

**C:\ClearScript\bin\Release> gacutil /i ClearScriptV8-64.dll**

Additionally, you must copy ClearScript’s V8 assemblies to the system folder:

On 32-bit Windows:

**C:\ClearScript\bin\Release>** **copy v8\*-ia32.dll %SystemRoot%\System32**

On 64-bit Windows:

**C:\ClearScript\bin\Release> copy v8\*-x64.dll %SystemRoot%\System32**

**C:\ClearScript\bin\Release> copy v8\*-ia32.dll %SystemRoot%\SysWOW64**

**Important:** The commands above require an elevated command prompt.

# Debugging with ClearScript and V8

V8 does not support standard Windows script debugging. Instead, it implements its own WebSocket-based debugging protocol. A convenient way to debug JavaScript code running in V8 is to use the Visual Studio Code IDE:

1. Install and launch [Visual Studio Code](https://code.visualstudio.com/).
2. Set up one or more ClearScript V8 debug configurations:
   1. Click **File** → **Preferences** → **Settings** to open your user settings.
   2. Locate or search for the **Launch** configuration and click **Edit in settings.json**.
   3. Add the following section to the file:

**"launch": {**

**"version": "0.2.0",**

**"configurations": [**

**{**

**"name": "Attach to ClearScript V8 on port 9222",**

**"type": "node",**

**"request": "attach",**

**"protocol": "inspector",**

**"address": "localhost",**

**"port": 9222**

**}**

**]**

**}**

* 1. You can specify additional configurations for different hosts, port numbers, and other options. See [here](https://code.visualstudio.com/docs/nodejs/nodejs-debugging) for more information.
  2. Click **File** → **Save**.

1. Enable script debugging in your application by invoking the [**V8ScriptEngine**](https://microsoft.github.io/ClearScript/Reference/html/T_Microsoft_ClearScript_V8_V8ScriptEngine.htm) constructor with **[V8ScriptEngineFlags.EnableDebugging](https://microsoft.github.io/ClearScript/Reference/html/T_Microsoft_ClearScript_V8_V8ScriptEngineFlags.htm)** and a TCP port number that matches one of your debug configurations. The default port number is 9222.
2. If you’d like to debug your application remotely, you must also do the following:
   1. Construct your script engine with the additional flag [**V8ScriptEngineFlags.EnableRemoteDebugging**](https://microsoft.github.io/ClearScript/Reference/html/T_Microsoft_ClearScript_V8_V8ScriptEngineFlags.htm).
   2. **Important:** If necessary, configure your firewall to allow incoming connections to your TCP port.
3. Attach the Visual Studio Code debugger to your application:
   1. Click **View** → **Debug** to bring up the Debug view.
   2. Select the appropriate debug configuration at the top of the Debug Side Bar.
   3. Click **Debug** → **Start Debugging**.

**Note:** You can also attach Visual Studio to your application for simultaneous debugging of script, managed, and native code.

# Acknowledgments

We’d like to thank:

1. [The V8 team](https://code.google.com/p/v8/people/list).
2. [Kenneth Reitz](http://kennethreitz.org) for generously providing the [**Httpbin**](http://httpbin.org) service.