Building, Integrating, and Deploying ClearScript

# Welcome to ClearScript!

ClearScript is a library that allows you to add scripting to your .NET applications. It supports [V8](https://v8.dev/) (Windows, Linux, macOS) and [JScript](https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/scripting-articles/hbxc2t98(v=vs.84))/[VBScript](https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/scripting-articles/t0aew7h6(v=vs.84)) (Windows only).

# ClearScript NuGet package

Now that official ClearScript NuGet packages are available, you can simply add one to your project and skip to [Section VII](#_Debugging_with_ClearScript_2) below.

## Windows

Use Visual Studio’s [NuGet Package Manager](https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio) or [**dotnet add package**](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package) to add the [Microsoft.ClearScript](https://www.nuget.org/packages/Microsoft.ClearScript) (x86/x64) or [Microsoft.ClearScript.win-arm64](https://www.nuget.org/packages/Microsoft.ClearScript.win-arm64) package to your project. Remember to use the **-v** option on the **dotnet** command line to add pre-release packages.

## Linux

Use [**dotnet add package**](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package) to add the [Microsoft.ClearScript.linux-x64](https://www.nuget.org/packages/Microsoft.ClearScript.linux-x64), [Microsoft.ClearScript.linux-arm](https://www.nuget.org/packages/Microsoft.ClearScript.linux-arm), or [Microsoft.ClearScript.linux-arm64](https://www.nuget.org/packages/Microsoft.ClearScript.linux-arm64) package to your project. Remember to use the **-v** option on the **dotnet** command line to add pre-release packages.

## macOS

Use [**dotnet add package**](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package) to add the [Microsoft.ClearScript.osx-x64](https://www.nuget.org/packages/Microsoft.ClearScript.osx-x64) package to your project. Remember to use the **-v** option on the **dotnet** command line to add pre-release packages.

# Building ClearScript

## Windows

**Important:** This procedure requires a 64-bit Windows operating system running on x64 hardware.

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. Use a directory at or near the root of a drive to avoid path length issues.

The provided project and solution files support [Visual Studio 2022](https://visualstudio.microsoft.com/downloads/) and [Visual Studio 2019](https://visualstudio.microsoft.com/vs/older-downloads/) Version 16.4 or later. They produce architecture-neutral managed libraries that target .NET Framework 4.5, .NET Framework 4.7.1, .NET Core 3.1, and .NET 5.0. ClearScript does not support older environments. The output directory is **bin\[Debug|Release]**.

**Important:** Ensure that [.NET 5.0 SDK](https://dotnet.microsoft.com/download/visual-studio-sdks) is installed before building ClearScript.

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.

To build full ClearScript with V8 support, you must first acquire and build V8:

1. **Important:** This procedure currently requires [Visual Studio 2019](https://visualstudio.microsoft.com/vs/older-downloads/). The V8 build does not yet support newer versions.
2. **Important:** Ensure that the path to your ClearScript root directory does not contain spaces or non-ASCII characters.
3. **Important:** Your Visual Studio installation must include the **.NET desktop development**, **Desktop development with C++**, and **Universal Windows Platform Development** workloads, as well as the **C++ ARM build tools** and **C++ ARM64** **build tools** components.
4. **Important:** The V8 build also requires [Windows 10 SDK](https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk/) version 10.0.19041.0 or later. Your Windows SDK installation must include the **Debugging Tools for Windows** feature.
5. Open a [Visual Studio 2019](https://visualstudio.microsoft.com/vs/older-downloads/) developer command prompt and run the **V8Update** script from your ClearScript root directory:

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

This script downloads the V8 source code and its dependencies and builds x86, x64, and arm64 V8 libraries in the specified configuration. It requires approximately 12GB 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 testing local V8 modifications.

Specifying **Debug** or **Release** is optional; the default is **Release**. If you need both variants, run the script twice as follows:

C:\Path\To\ClearScript> **V8Update && V8Update /N Debug**

**Note:** Depending on your PC hardware and internet connection, this step could take up to a half-hour or longer.

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 2019 or 2022.

**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.win-x86**, **ClearScriptV8.win-x64**, and **ClearScriptV8.win-arm64**), 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**).

## Linux

**Important:** This procedure requires a 64-bit Linux operating system running on x64 hardware.

Here’s how to acquire and build V8 and ClearScript on Linux:

1. Install the following packages: [Git](https://www.git-scm.com/download/linux), [.NET 5.0 SDK](https://dotnet.microsoft.com/download/dotnet/5.0), [Clang](https://clang.llvm.org/), [GNU Make](https://www.gnu.org/software/make/), [Python 2](https://www.python.org/downloads/release/python-2718/), and [pkgconf](http://pkgconf.org/). For each of these, check your package manager first, and then the official website.
2. **Important:** To avoid V8 build issues, ensure that the **python** command launches Python 2.
3. 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. Avoid very long directory paths.
4. **Important:** Ensure that the path to your ClearScript root directory does not contain spaces or non-ASCII characters.
5. Run the following command from your ClearScript root directory:

user@host:/path/to/ClearScript$ **make -f Unix/Makefile [DEBUG=1] [CPU=[arm|arm64]]**

Specifying **DEBUG=1** is optional; the default configuration is **Release**. The output directory is **bin/[Debug|Release]**.

Specify **CPU=arm** to cross-build for the arm architecture. This requires arm C++ cross-build tools, such as [g++-10-arm-linux-gnueabihf](https://packages.ubuntu.com/focal/g++-10-arm-linux-gnueabihf) on Ubuntu.

Specify **CPU=arm64** to cross-build for the arm64 architecture. This requires arm64 C++ cross-build tools, such as [g++-aarch64-linux-gnu](https://packages.ubuntu.com/focal/g++-aarch64-linux-gnu) on Ubuntu.

## macOS

Here’s how to acquire and build V8 and ClearScript on macOS:

1. Install [Xcode](https://developer.apple.com/xcode/) and [.NET 5.0 SDK](https://dotnet.microsoft.com/download/dotnet/5.0).
2. **Important:** To avoid V8 build issues, ensure that the **python** command launches Python 2.
3. 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. Avoid very long directory paths.
4. **Important:** Ensure that the path to your ClearScript root directory does not contain spaces or non-ASCII characters.
5. Launch the [Terminal](https://en.wikipedia.org/wiki/Terminal_(macOS)) app. **Important:** If you’re on an M1 machine, open a Rosetta shell as follows:

user@host:~$ **arch -x86\_64 zsh**

1. Run the following command from your ClearScript root directory:

user@host:/path/to/ClearScript$ **make -f Unix/Makefile [DEBUG=1] [CPU=[arm64]]**

Specifying **DEBUG=1** is optional; the default configuration is **Release**. The output directory is **bin/[Debug|Release]**.

Specify **CPU=arm64** to cross-build for the arm64 (M1) architecture.

# Building strong-named ClearScript assemblies (optional, Windows only)

ClearScript 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:\Path\To\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

## Windows (x86/x64)

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.Core.dll**, **ClearScript.Windows.dll**, **ClearScript.Windows.Core.dll**, and **ClearScript.V8.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.win-x86.dll**

**ClearScriptV8.win-x64.dll**

## Windows (arm64)

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

C:\Path\To\MyApp> **dotnet add reference D:\Path\To\ClearScript\NetCore\ClearScript.V8**

**Important:** If your application fails to load ClearScript, ensure that the following ClearScript output files are present in your application’s output directory:

**ClearScript.Core.dll**

**ClearScript.V8.dll**

**ClearScriptV8.win-arm64.dll**

## Linux

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

user@host:/path/to/MyApp$ **dotnet add reference /path/to/ClearScript/NetCore/ClearScript.V8**

**Important:** If your application fails to load ClearScript, ensure that the following ClearScript output files are present in your application’s output directory:

**ClearScript.Core.dll**

**ClearScript.V8.dll**

**ClearScriptV8.linux-[x64|arm64].so**

## macOS

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

user@host:/path/to/MyApp$ **dotnet add reference /path/to/ClearScript/NetCore/ClearScript.V8**

**Important:** If your application fails to load ClearScript, ensure that the following ClearScript output files are present in your application’s output directory:

**ClearScript.Core.dll**

**ClearScript.V8.dll**

**ClearScriptV8.osx-[x64|arm64].dylib**

# Machine-level (global) deployment (.NET Framework only, x86/x64)

You can 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 assembly in the [Global Assembly Cache](https://docs.microsoft.com/en-us/dotnet/framework/app-domains/gac):

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

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

On 32-bit Windows:

C:\Path\To\ClearScript\bin\Release>**copy ClearScriptV8.win-x86.dll %SystemRoot%\System32**

On 64-bit Windows:

C:\Path\To\ClearScript\bin\Release> **copy ClearScriptV8.win-x64.dll %SystemRoot%\System32**

C:\Path\To\ClearScript\bin\Release> **copy ClearScriptV8.win-x86.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 highlighted 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. You can add the flag [**V8ScriptEngineFlags.AwaitDebuggerAndPauseOnStart**](https://microsoft.github.io/ClearScript/Reference/html/T_Microsoft_ClearScript_V8_V8ScriptEngineFlags.htm) to have the script engine stop and wait for a debugger connection before executing script code. Once attached, the debugger will be in a breakpoint state.
3. 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.
4. Attach the Visual Studio Code debugger to your application:
   1. Click **View** → **Run** to bring up the Run Side Bar.
   2. Click **Run** → **Start Debugging** to attach the debugger to your application.

**Note:** If you’re having issues with remote debugging, try disabling the **debug.JavaScript.usePreview** setting.

**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:

* [The V8 team](https://code.google.com/p/v8/people/list).
* [The Jekyll team](https://jekyllrb.com/team/).
* [Kenneth Reitz](http://kennethreitz.org) for generously providing the [**Httpbin**](http://httpbin.org) service.
* [Michael Rose](https://mademistakes.com/) for generously providing the [So Simple](https://mmistakes.github.io/so-simple-theme/) Jekyll theme.