Contents

[Introduction 2](#_Toc492299179)

[What is IntelliSense? 3](#_Toc492299180)

[IntelliSense for External APIs 5](#_Toc492299181)

[How to setup Visual Studio Code to provide IntelliSense for the Primer API 7](#_Toc492299182)

[Download Visual Studio Code 7](#_Toc492299183)

[Copy the Primer API IntelliSense File on to your Computer 7](#_Toc492299184)

[Tell Visual Studio the Location of the Primer API IntelliSense File 7](#_Toc492299185)

[How to setup Visual Studio Express to provide IntelliSense for the Primer API 9](#_Toc492299186)

[Download Visual Studio Express 9](#_Toc492299187)

[Copy the Primer API IntelliSense File on to your Computer 9](#_Toc492299188)

[Tell Visual Studio the Location of the Primer API IntelliSense File 9](#_Toc492299189)

[Add a Global Reference 9](#_Toc492299190)

[Add a Local Reference 11](#_Toc492299191)

[Add a \_references File 12](#_Toc492299192)

[Additional use for the \_references.js file 17](#_Toc492299193)

# Introduction

This document is a guide for setting up Intellisense in ***Visual Studio Express for Web*** and ***Visual*** ***Studio Code***. These are two separate editors created by Microsoft that can be used to write JavaScript.

Despite having similar names, they are actually quite different and we would recommend using ***Visual Studio Code*** as this is a much smaller download (MBs rather than GBs). It is a cross-platform editor which can be used on Windows, Linux or Mac (***Visual Studio Express can only be used on Windows***) and also has excellent documentation online.

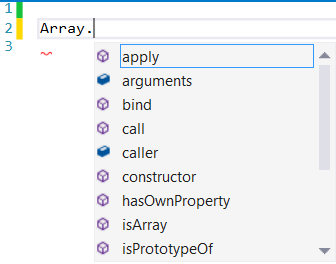
The document is split into two sections; the first showing how to setup ***Visual Studio Code*** and the other for ***Visual Studio Express for Web***.

## What is IntelliSense?

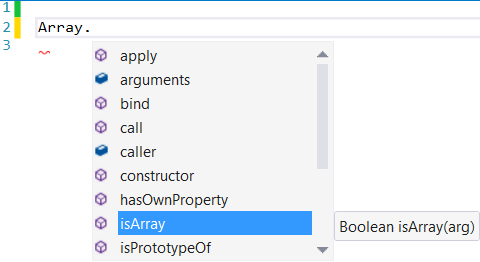
From the Microsoft MSDN website:

*‘IntelliSense helps you write code faster and with fewer errors by providing information while you code. As you write the script IntelliSense lists the objects, functions, properties and parameters that are available based on your current context. You can select a coding option from the pop-up list provided by IntelliSense to complete the code.’*

For example if you type ‘***Array.***’ IntelliSense will pop-up a list of the available static functions and properties for the Array object:



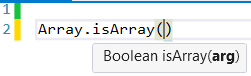
You can then carry on typing or use the arrow key to select the code you want:



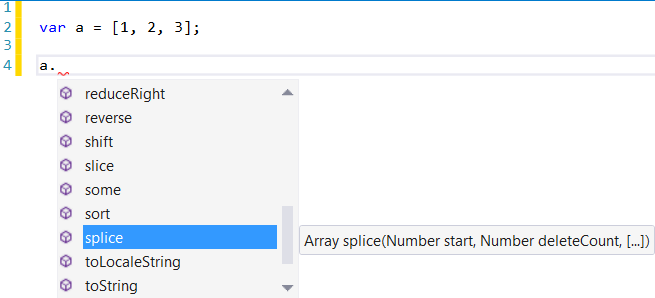
Then press the Tab key to auto-complete:



Because isArray() is a function, if you type ‘**(**‘ IntelliSense will display the parameters required and the return type. It can also show a description of the function and parameter (although it doesn’t for this function):



This also works for the member functions and properties of instances of objects. For example, if you assign an array to a variable and then use the variable, the IntelliSense knows that it is an instance of an Array so will pop-up the member functions and properties:



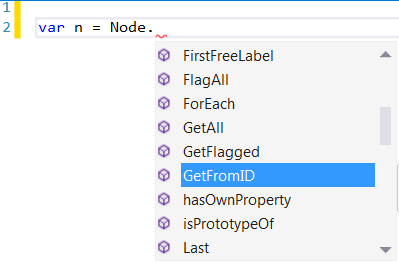
## IntelliSense for External APIs

IntelliSense support for the fundamental objects like Arrays, Strings etc. is useful as it makes writing scripts quicker since you don’t need to browse an external source for function names, parameters and properties, etc.

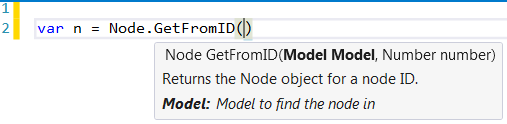
However, by default Visual Studio does not know about external APIs or libraries so it does not offer IntelliSense for the objects in the Primer API, e.g. if you type ‘***Node.***’ it does not know about the Node object so will not list any of the static functions like **GetAll()** or **GetFromID()**.

Luckily it is possible to point Visual Studio to a file containing information about the API so that IntelliSense works. Once this is set up IntelliSense is able to do things like this:

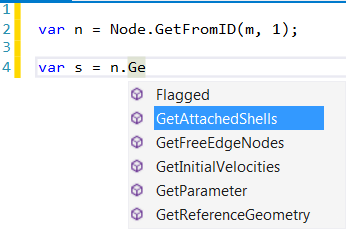
Show static functions for the Node object:



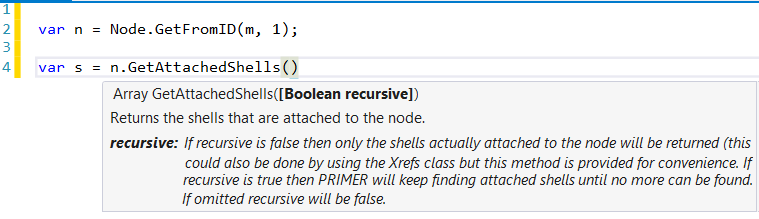
Show a description of the function and parameters, list the parameters and show the return type:



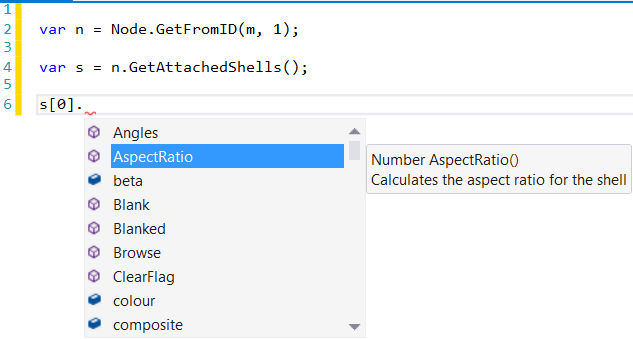
Show member functions (i.e. because **Node.GetFromID()** returns a Node object, Visual Studio knows ‘n’ is an instance of a Node so lists the appropriate member functions and properties):



Show a description of member functions and parameters, list the parameters and show the return type:



Because **GetAttachedShells()** returns an array of Shell objects IntelliSense knows that each element in the array ‘s’ is an instance of a Shell object so will show the appropriate member functions for a Shell:



Hopefully what has been shown is convincing enough to persuade you that Visual Studio IntelliSense will save you time writing your scripts. If it has, the following section shows you what you need to do to setup Visual Studio to get IntelliSense working for the Primer API.

# How to setup Visual Studio Code to provide IntelliSense for the Primer API

## Download Visual Studio Code

If you don’t already have it you’ll need to install ***Visual Studio Code***:

<https://code.visualstudio.com/>

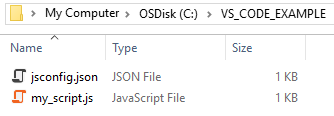
Documentation on how to use ***Visual Studio Code*** can be found at the same site.

## Copy the Primer API IntelliSense File on to your Computer

For ***Visual Studio Code*** you need to copy the **primer.d.ts** file from the **Oasys xx/Intellisense** installation directory to somewhere on your computer.

## Tell Visual Studio the Location of the Primer API IntelliSense File

You need to tell ***Visual Studio Code*** the location of the IntelliSense file by creating a ‘jsconfig.json’ file in the same directory as your JavaScript.js file, e.g.



This is what it needs to contain if your script is a single file:

**{**

**"compilerOptions": {**

**"checkJs": true,**

**"lib": [**

**"es5"**

**]**

**},**

**"include": [**

**"C:/primer\_js\_api/primer.d.ts",**

**"\*.js"**

**]**

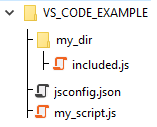
**}**

If you want to search directories recursively, use \*\*, e.g. “\*\*/\*.js” would search the current directory and any child ones for \*.js files.

You can copy and paste this into your file, changing ‘C:/primer\_js\_api/primer.d.ts’ to the location of the file on your computer.

If you now open ***Visual Studio Code***, go to *File->Open Folder* and select the folder containing the script. You should now be able to edit your script with IntelliSense working.

If your script contains multiple files, i.e. it uses the Use() function to include files, then you will have to modify the jsconfig.js file so ***Visual Studio Code*** knows about the other files. For example if your directory structure looks like this:



You would need to add a line to the file so it knows about the *included.js* file in the *my\_dir* directory:

**{**

**"compilerOptions": {**

**"checkJs": true,**

**"lib": [**

**"es5"**

**]**

**},**

**"include": [**

**"C:/primer\_js\_api/primer.d.ts",**

**"my\_dir/\*.js",**

**"\*.js"**

**]**

**}**

# How to setup Visual Studio Express to provide IntelliSense for the Primer API

## Download Visual Studio Express

If you don’t already have it you’ll need to install ***Visual Studio Express for Web***:

<https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx>

Make sure you get the Express edition and not Community as this has some caveats on what you can use it for if you belong to a commercial organisation.

Also make sure you get the ‘Express for Web’ version not ‘Express for Desktop’

## Copy the Primer API IntelliSense File on to your Computer

The Primer API IntelliSense **primer.intellisense.js** file can be found in the **Oasys xx/Intellisense** installation directory.

You need to copy this somewhere on your computer. The file is created automatically from comments in the Primer source code so we can easily update it as and when new features are added to the API.

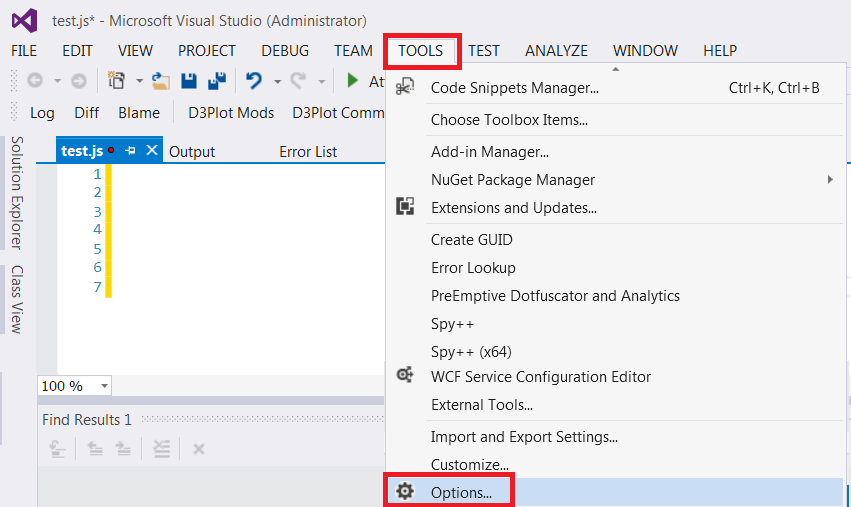
## Tell Visual Studio the Location of the Primer API IntelliSense File

There are three ways of telling Visual Studio the location of the IntelliSense File:

### Add a Global Reference

The simplest method is to add a reference to the IntelliSense file globally so that any JavaScript file opened in Visual Studio will know about the Primer API. Whilst it is the simplest method, it is the least flexible as you may not want it to know about the Primer API if you’re writing a T/HIS script for example.

1. Go to TOOLS->Options…

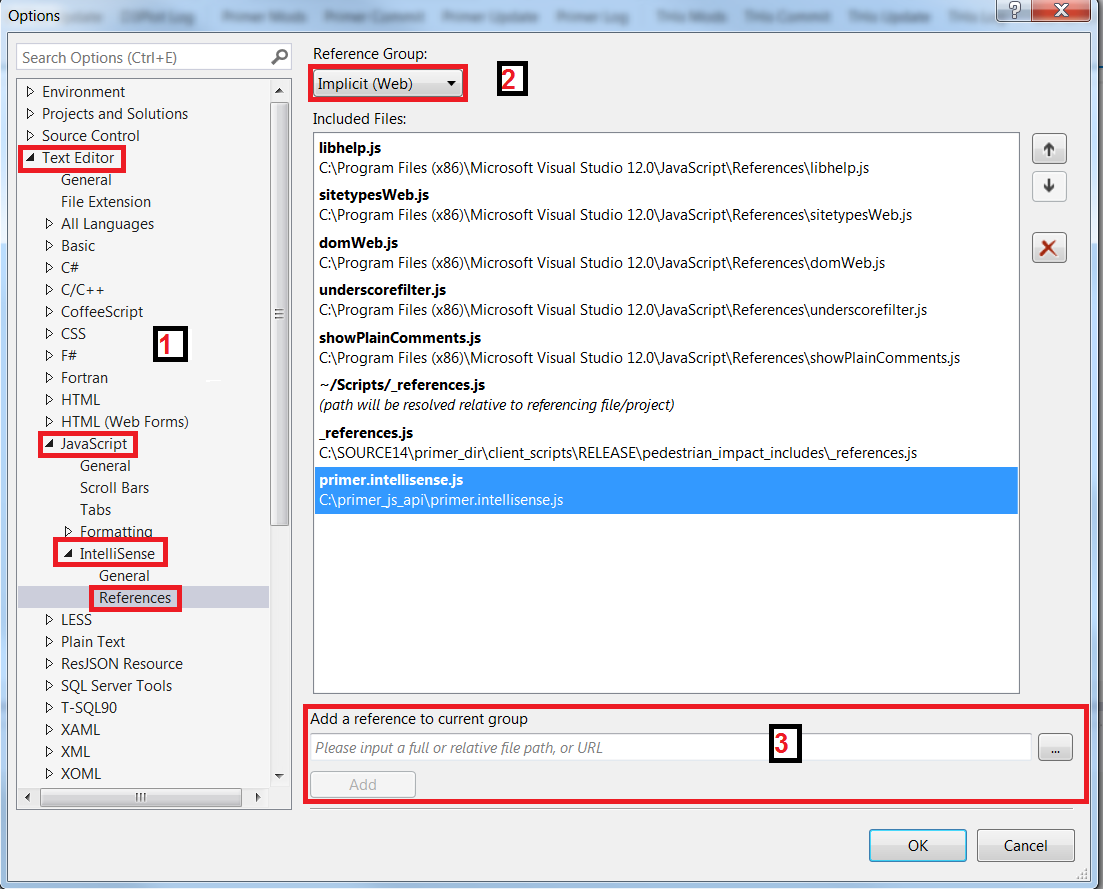


1. In the Options menu, select Text->JavaScript->IntelliSense->References.

Then change the Reference Group to ‘Implicit (Web)’.

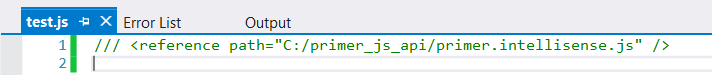
Then select the primer.intellisense.js file you copied to your computer.

Then press OK.



### Add a Local Reference

Alternatively you can add a reference directly in your JavaScript file using a special comment to point to the IntelliSense file (note the triple slash ‘///’):



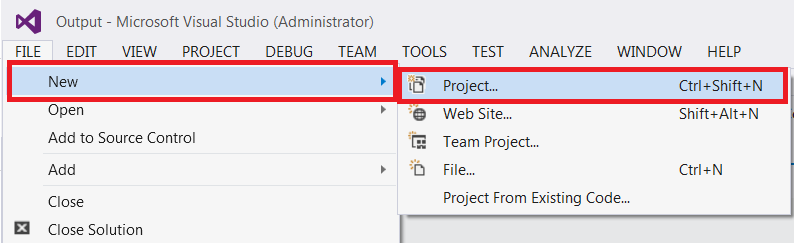
This is more flexible than the previous approach as it means if you’re writing a T/HIS JavaScript you won’t get the Primer API IntelliSense if you don’t add this line. However, you have to remember to add it to every file you want IntelliSense for.

### Add a \_references File

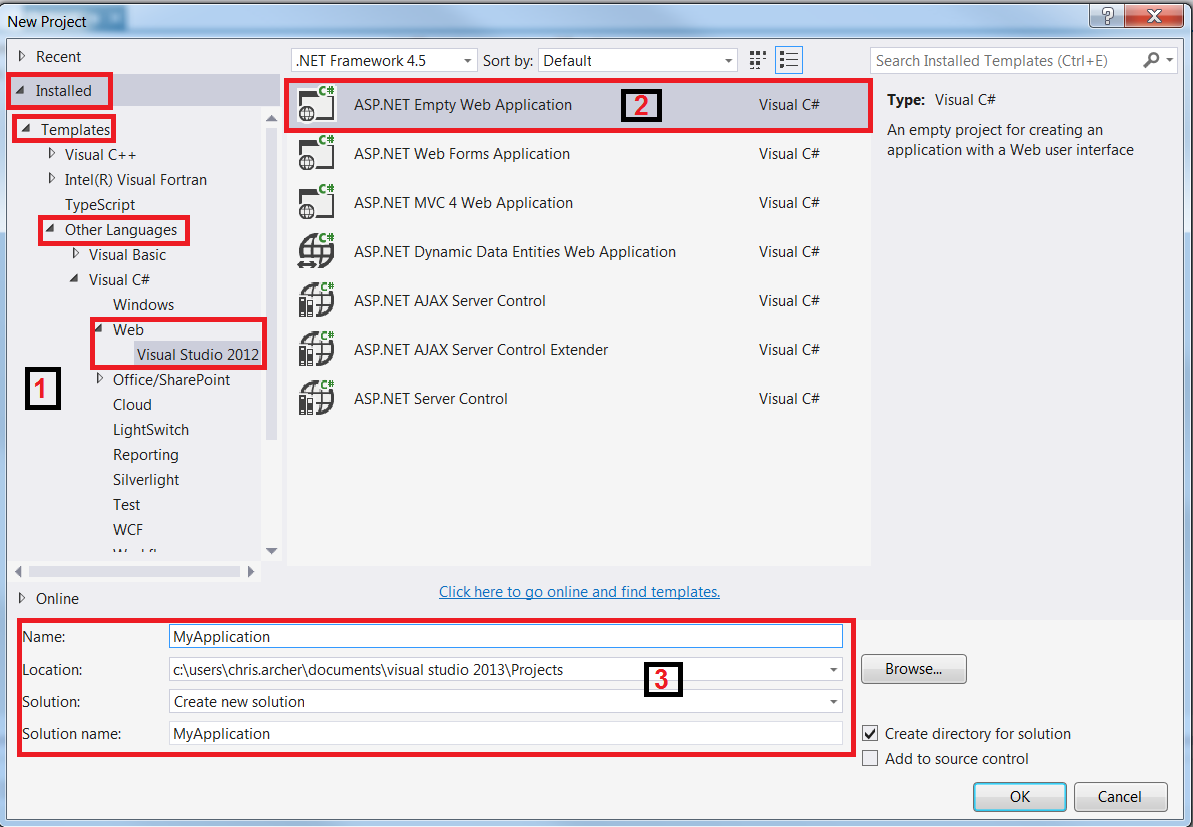
The third way is to create what is called a project in Visual Studio which is just a collection of files, e.g. if your script has multiple files then you will need to create a project. You can then add a file to the project to reference the IntelliSense file.

This is the most flexible method and is the recommended way to do it if your script has multiple files (i.e. it uses the Use() function to include files).

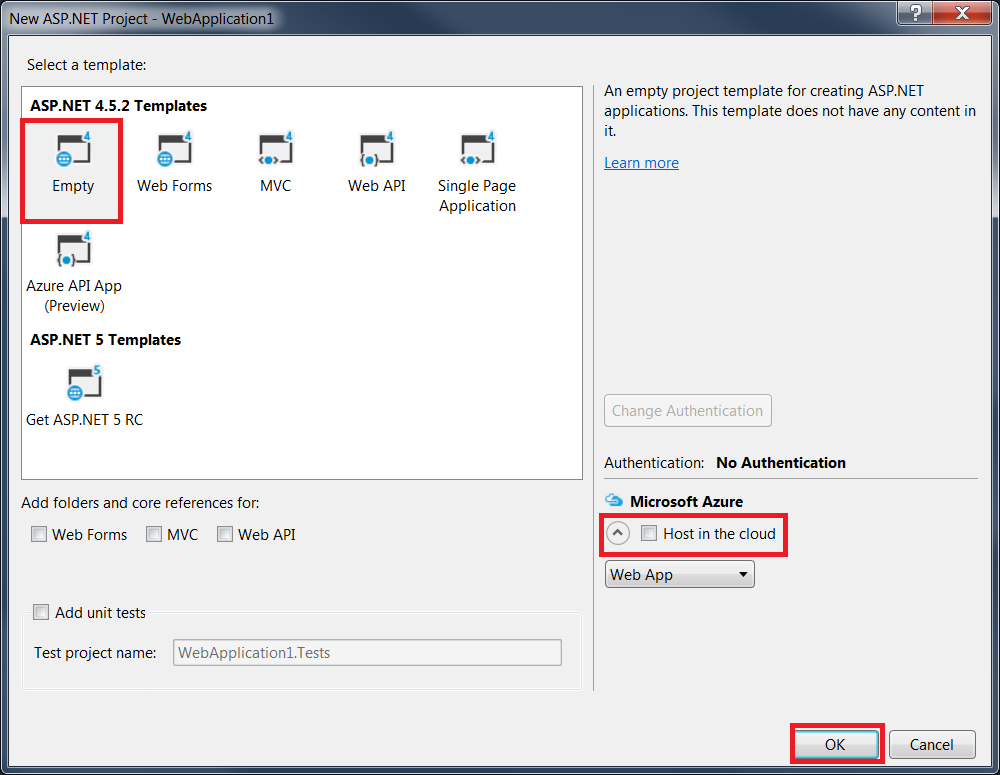
1. Go to FILE->New->Project…



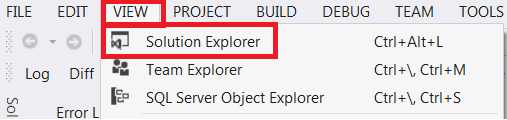
1. Select the Visual C# Web template, choose a location where you want to save it and choose a name for your project and then press OK:



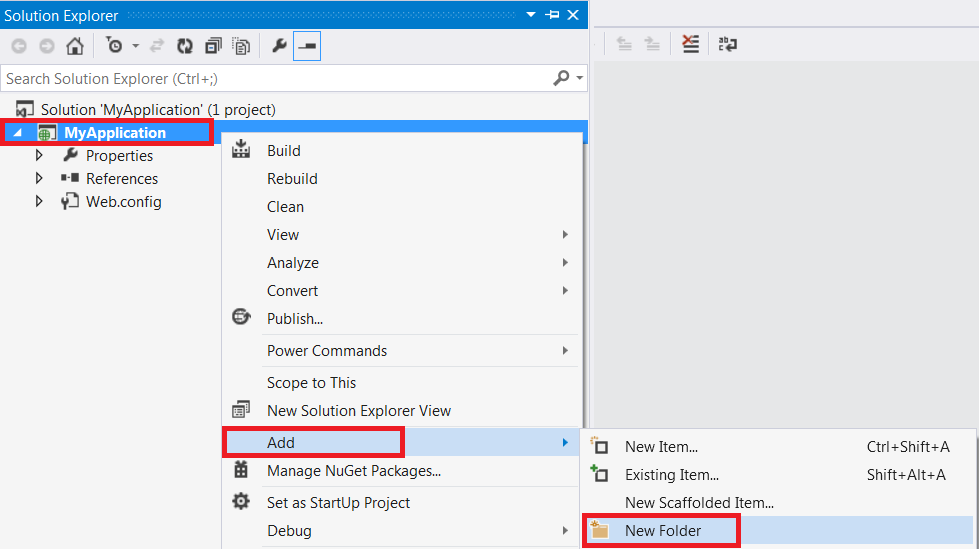
1. You may get a further window pop up (depending on the version of Visual Studio you have). If so, select the ‘Empty’ template and turn off ‘Host in the cloud’ then press OK:



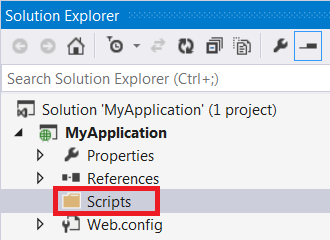
1. Select VIEW->Solution Explorer



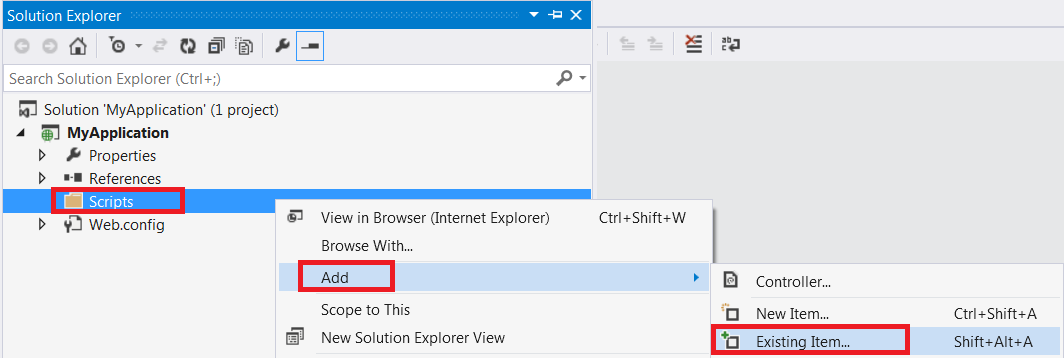
1. If a ‘scripts’ folder doesn’t already exist, right-click on your project and add a new folder. (If it does exist you may need to delete some of the .js files Visual Studio has automatically added into it).



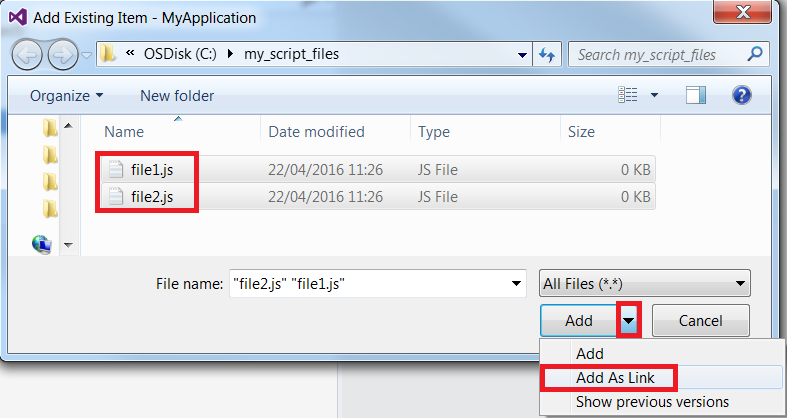
1. Rename it to ‘Scripts’.



1. You should now be able to add your script file(s) (which can live anywhere on your computer) to this directory e.g. if I have two files in a the directory C:\my\_script\_files, right-click on the Scripts folder and add the existing files:

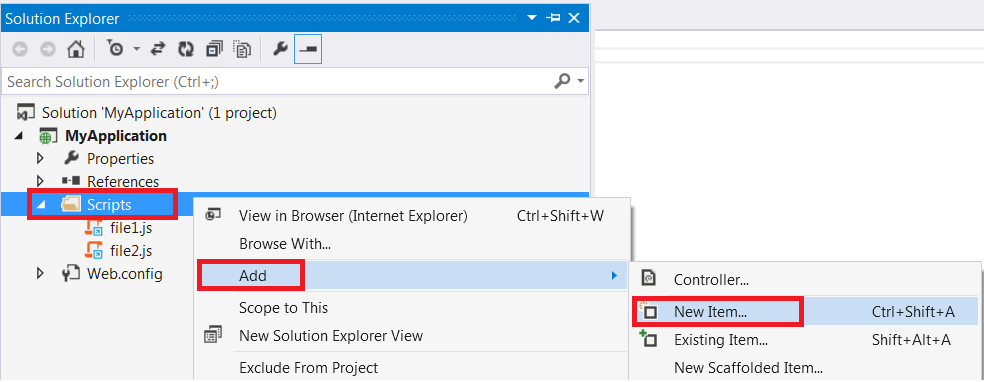


1. Select the files (using shift-click) and make sure you add them as a link:

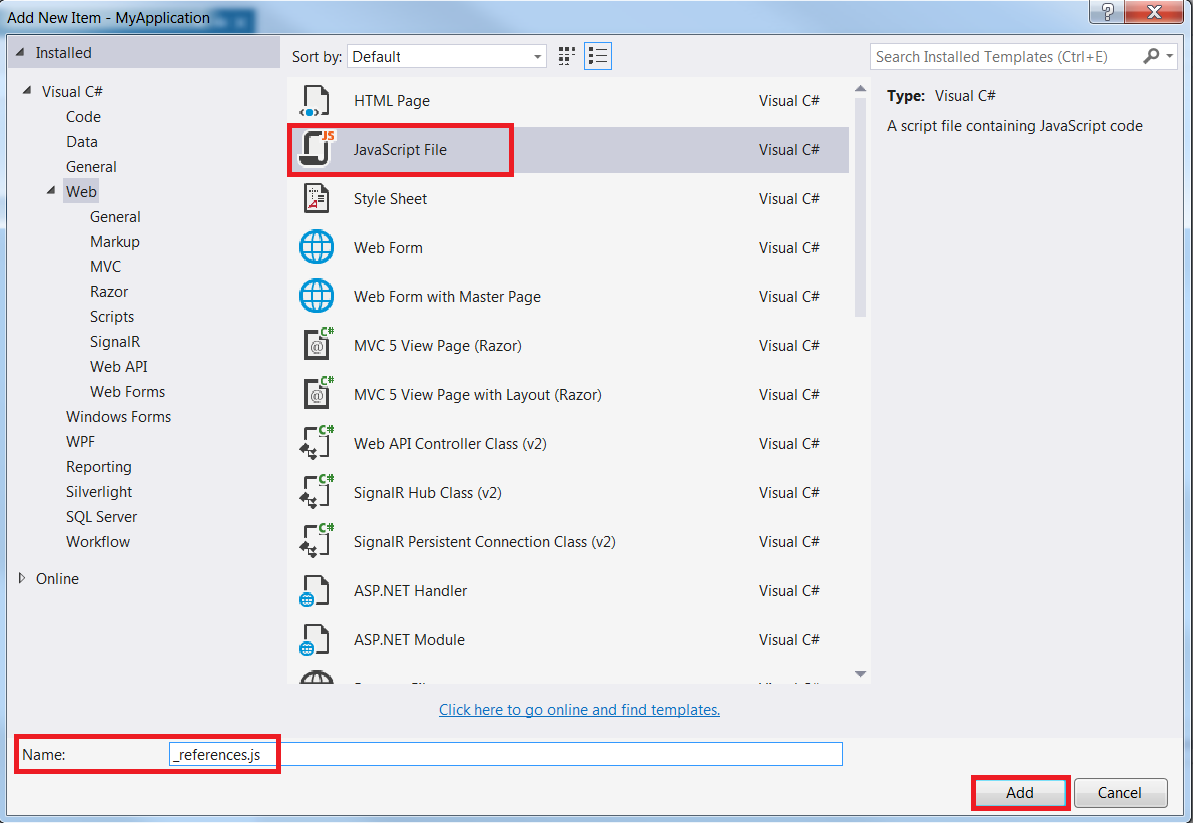


1. You should now be able to open the files in Visual Studio by double clicking on them. However, IntelliSense won’t yet work for the Primer API. You need to add an additional \_references.js file.

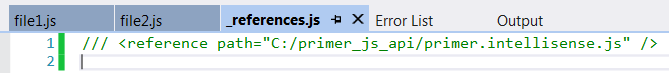
Right-click on the Scripts folder and add a new item:



1. Select the JavaScript file, name it ‘\_references.js’ and then press Add:



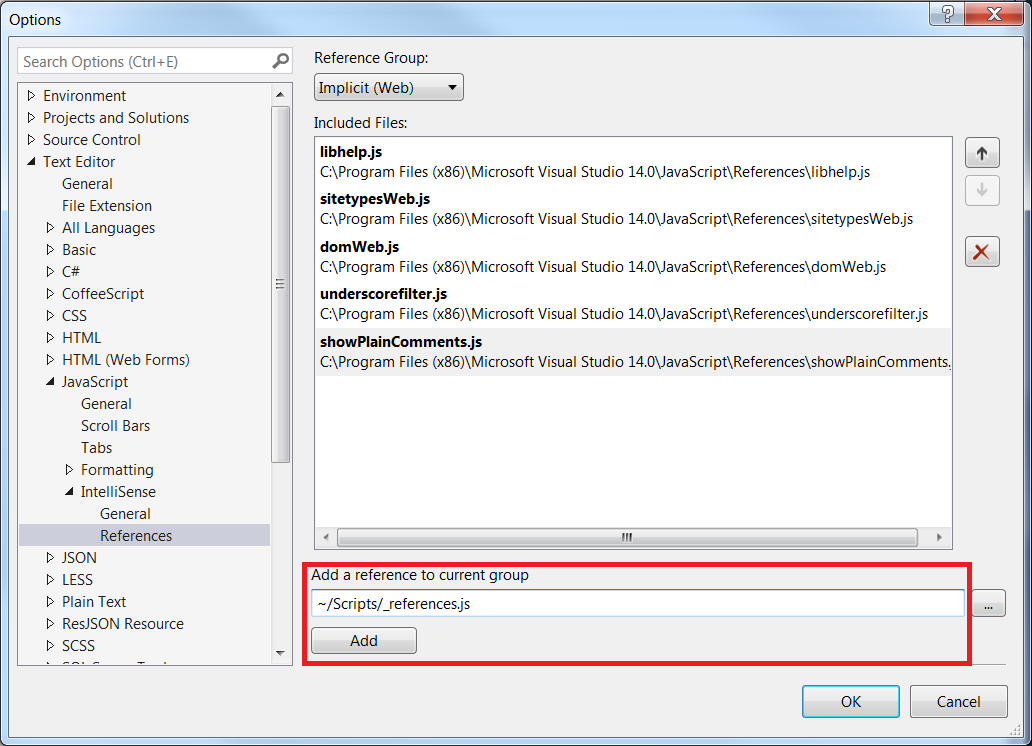
1. Add the reference to the IntelliSense file with the special comment. Note that you need to press return at the end of the line or it won’t work for some reason:



1. Go to TOOLS->Options and select Text->JavaScript->IntelliSense->References.

Then change the Reference Group to ‘Implicit (Web)’.

Then add a reference to ‘~/Scripts/\_references.js’

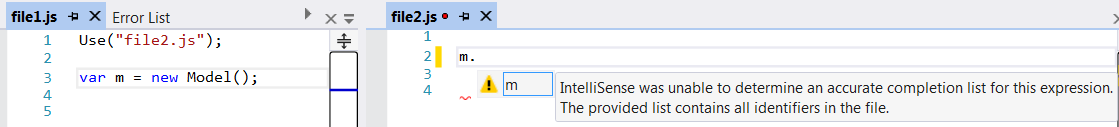


1. IntelliSense for the Primer API should now work in all the files in your project.

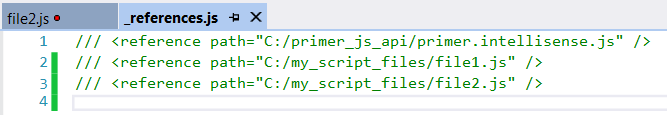
## Additional use for the \_references.js file

If your script contains multiple files and they are linked together using the Use() function, Visual Studio does not know which files are included by which files. This means if you have a global variable defined in one file and another file uses it, Visual Studio has no way of knowing what the variable type is so cannot provide sensible IntelliSense.

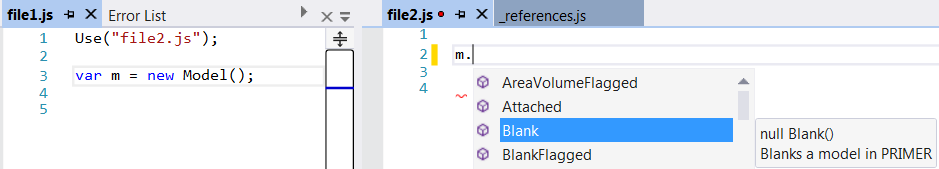
e.g. If file1.js includes file2.js and defines a global variable ‘m’ as a Model object, the second file does not know that ‘m’ exists or that it’s an instance of a Model object.



Luckily you can add a reference to the \_references.js file to resolve this problem:



It then knows about the variable defined in the other file:



This is why creating a \_references.js file is the recommended method as means you only have to update a single file to get IntelliSense working correctly.