Creating a Plugin

This article walks you through the process to create a plugin for HS4 that starts and stops with the system.
Creating a plugin that starts and stops with HomeSeer is a rather quick process. You can either start with the Hello World Plugin as a blank canvas to jump right into coding features and device management functionality, or you can follow the steps below to create your own dotNet solution from scratch.

On this page:

> The following instructions assume you are using Visual Studio 2019. If you prefer a different IDE, you are welcome to follow along using that.

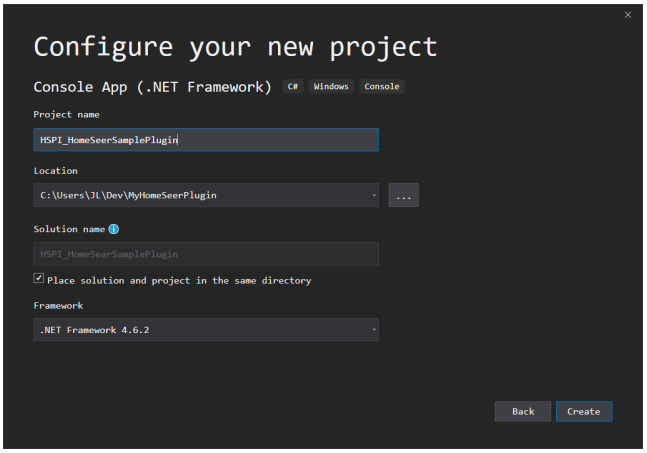
What you should know

You should be familiar with the following topics before developing a plugin for the HomeSeer platform:

- Visual Studio
- C# or Visual Basic
- The basics of creating a dotNET Framework console application
- NuGet dependency manager

Create a new project

Create a new console app project targeting .NET Framework 4.6.2 in your choice of language. (Examples are provided in C# and VB)



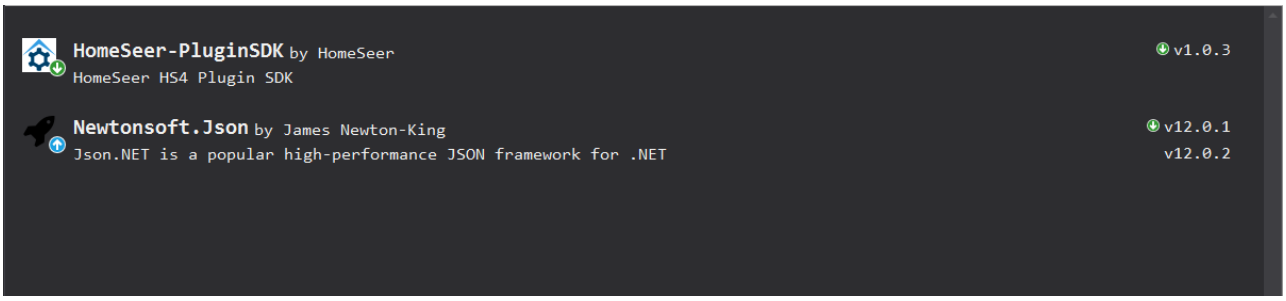
> The project's root namespace and assembly name must start with **HSPI_**
>
> *For example:* HomeSeerSamplePlugin should be HSPI_HomeSeerSamplePlugin

Add PluginSDK NuGet package

The PluginSDK is available through NuGet for inclusion into your plugin projects.

1. Right-click your project in the *Solution Explorer*
2. Select *Manage NuGet Packages...* from the menu
3. In the *Browse* tab, search for **homeseer pluginsdk**
4. Install the latest version of HomeSeer-PluginSDK to your project.
5. Clear the search box and click the *Installed* tab

The list of currently installed NuGet packages for your project should look like this:



In some older versions of Visual Studio, the default NuGet package source is not configured. Make sure the following package source is configured: https://api.nuget.org/v3/index.json

Define HSPI class

The HomeSeer platform utilizes reflection to facilitate communication between plugins and the system. When HomeSeer connects to a plugin, it looks for an HSPI class defined at the root of the assembly namespace.

Create a new class in your project called **HSPI** that inherits from AbstractPlugin.

C#

```
using HomeSeer.PluginSdk;

namespace HSPI_HomeSeerSamplePlugin {
    class HSPI : AbstractPlugin {

    }
}
```

VB

```
Imports HomeSeer.PluginSdk

Public Class HSPI
    Inherits AbstractPlugin

End Class
```

Implement required AbstractPlugin members

Override all of the required methods and properties in your HSPI class. Delete the default code in the three overridden methods that throw exceptions, and return true in OnSettingsChange. You will define these later on in the plugin development process.

C#

```
protected override void Initialize() {

}

protected override bool OnSettingChange(string
pageId, AbstractView currentView, AbstractView
changedView) {
    return true;
}

protected override void BeforeReturnStatus() {

}

public override string Id { get; }

public override string Name { get; }
```

VB

```
Protected Overrides Sub Initialize()

End Sub

Protected Overrides Function OnSettingChange
(pageId As String, currentView As AbstractView,
changedView As AbstractView) As Boolean
        Return True
End Function

Protected Overrides Sub BeforeReturnStatus()

End Sub

Public Overrides ReadOnly Property Id As String

Public Overrides ReadOnly Property Name As String
```

Name the plugin

The Name of the plugin is what is shown to users on the HomeSeer platform. It should be unique and represent what your plugin does.

Do not use any special characters in the name of your plugin except "-", ".", or " " (space), and please aim for 16 characters or less to ensure the name displays correctly on all display sizes.

C#

```
public override string Name { get; } = "HomeSeer
Sample Plugin";
```

VB

```
Public Overrides ReadOnly Property Name As String
        Get
                Return "HomeSeer Sample Plugin"
        End Get
End Property
```

Set the plugin Id

The Id property on the HSPI class is used by HomeSeer as the unique identifier for your plugin. It is recommended to use the name of your plugin, removing any special characters, and removing spaces or replacing them with dashes. Case is ignored. See the AbstractPlugin.Id Property documentation for more information.

C#

```
public override string Id { get; } =
"HomeSeerSamplePlugin";
```

VB

```
Public Overrides ReadOnly Property Id As String
    Get
        Return "HomeSeerSamplePlugin"
    End Get
End Property
```

This is all that is necessary to create and start a plugin on the HomeSeer platform. Your HSPI class should look like the class below.

C#

```
using HomeSeer.Jui.Views;
using HomeSeer.PluginSdk;

namespace HSPI_HomeSeerSamplePlugin {

    public class HSPI : AbstractPlugin {

        protected override void Initialize() {

        }

        protected override bool OnSettingChange
(string pageId, AbstractView currentView,
AbstractView changedView) {
            return true;
        }

        protected override void
BeforeReturnStatus() {

        }

        public override string Id { get; } =
"HomeSeerSamplePlugin";

        public override string Name { get; } =
"HomeSeer Sample Plugin";

    }

}
```

VB

```
Imports HomeSeer.Jui.Views
Imports HomeSeer.PluginSdk

Public Class HSPI
    Inherits AbstractPlugin

    Protected Overrides Sub Initialize()

    End Sub

    Protected Overrides Function OnSettingChange
(pageId As String, currentView As AbstractView,
changedView As AbstractView) As Boolean
        Return True
    End Function

    Protected Overrides Sub BeforeReturnStatus()

    End Sub

    Public Overrides ReadOnly Property Id As String
        Get
            Return "HomeSeerSamplePlugin"
        End Get
    End Property

    Public Overrides ReadOnly Property Name As
String
        Get
            Return "HomeSeer Sample Plugin"
        End Get
    End Property

End Class
```

Finish startup object

The last thing to do to get the plugin ready to build, is initialize and start the plugin in the application's Program.Main method. All of the code needed to start a connection between the plugin and HomeSeer is already taken care of in the AbstractPlugin.Connect Method; so the only thing you need to do in Program.Main is initialize an instance of the HSPI class and call Connect.

C#

```
namespace HSPI_HomeSeerSamplePlugin {

    internal static class Program {

        private static HSPI _plugin;

        public static void Main(string[] args) {

            _plugin = new HSPI();
            _plugin.Connect(args);
        }
    }
}
```

VB

```
Module Program

    Dim _plugin As HSPI

    Sub Main(ByVal args As String())

        _plugin = New HSPI()
        _plugin.Connect(args)

    End Sub
End Module
```

Add app.config file

All plugins must include an application config file to help in locating dependent assemblies. Without this file, your plugin may not work correctly. This is discussed in more detail in Plugin Packaging.

Add a new file to the project called **app.config** with the code below, using the Id of your plugin in the *privatePath* attribute of the *probing* tag.

app.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
        <startup>
                <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2"/>
        </startup>
        <runtime>
                <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
                        <probing privatePath="bin/HomeSeerSamplePlugin;bin/homeseer"/>
                </assemblyBinding>
        </runtime>
</configuration>
```

See the Microsoft docs on Configuring Apps by using Configuration Files for more information on the available tags.

Build and deploy

HomeSeer probes its install directory for EXE files that begin with **HSPI_** (which should automatically be the name of the application file produced by your project). There are a couple of different ways to go build and deploy a plugin to begin testing it with HomeSeer. You can build the project and then manually copy all of the files to their respective locations in the HomeSeer install directory, run your plugin remotely, configure the project build process to automatically copy everything to the HomeSeer install directory, or a combination of any of these methods; whichever works best for you. Prior to publishing your plugin though, it is always recommended to test a replicate user experience by deploying the plugin directly to the HomeSeer install directory and letting HomeSeer initialize the connection just like if you installed it from the store to your system for the first time.

Issue HS-4

There is currently a bug in assembly management when it comes to facilitating custom liquid tags in HTML feature pages that prevents a plugin EXE from being deleted or overwritten unless the HomeSeer system is stopped.

Manual deployment

- Pros
  - Allows you to accurately represent the user experience by letting HomeSeer initialize the connection to the plugin
  - No need to clean up any unintended files copied during the build process
- Cons
  - Can be time consuming and tedious unless you have a post-build script
  - Must start and stop the plugin through HomeSeer
  - Can't debug with breakpoints

To manually deploy your plugin to the HomeSeer install directory:

1. Build your project (do not build directly to the HomeSeer install directory)
2. Copy the outputted EXE file *(HSPI_HomeSeerSamplePlugin.exe)* and the app.config *(HSPI_HomeSeerSamplePlugin.exe.config)* files for your project to the root of the HomeSeer install directory. This is the same folder that HS4.EXE is in.
3. Copy any additional assemblies to a corresponding bin path within the HomeSeer install directory matching the Id of your plugin. This should match the directory defined in your plugin's **app.config** file. *(bin\HomeSeerSamplePlugin\)*
4. Copy any HTML files to a corresponding path under the html folder in the HomeSeer install directory matching the Id of your plugin. *(html\HomeSeerSamplePlugin\)*
5. Navigate to the *Manage Plugins* page in HomeSeer - Your plugin should now be in the list of installed plugins
6. Start your plugin

Remote debugging

- Pros
  - Full debugger suite available through IDE
  - Fast development iterations because there's no need to do any file cleanup every build
- Cons
  - Does not accurately represent a user experience
  - You still have to manually copy over HTML files

To run your plugin remotely:

1. Build your project (do not build directly to the HomeSeer install directory)
2. Copy any HTML files to a corresponding path under the html folder in the HomeSeer install directory matching the Id of your plugin. *(html\HomeSeerSamplePlugin\)*
3. Make sure HomeSeer is started
4. Start your plugin through the IDE

Deploying directly to HomeSeer

- Pros
  - Accurately represents the user experience
  - Full debugger suite available through IDE
  - Required files (HTML and assemblies) are automatically deployed to the right place
- Cons
  - Subject to problems when an assembly is copied to the HomeSeer install directory unintentionally (usually due to the "Copy Local" flag being set to true on an assembly reference)
  - Rebuilding sometimes requires you to completely stop HomeSeer

To set up direct deployment:

1. Unload the project in your IDE
2. Edit the project file
3. Within the *project* element, and underneath the first *import* element, add a new *PropertyGroup* element with a *Label* of "HomeSeer Properties" that includes two child elements called *HomeSeerRoot and PluginId*. The *HomeSeerRoot* element should be a relative path from the plugin project to the HomeSeer install directory, and the *PluginId* element should be the Id of your plugin without any HSPI_ prefix.

```
<PropertyGroup Label="HomeSeer Properties">
    <PluginId>HomeSeerSamplePlugin</PluginId>
        <HomeSeerRoot>..\Homeseer\Homeseer\</HomeSeerRoot>
</PropertyGroup>
```

4. Copy the entire *PropertyGroup* element for the debug build and paste it below the same element to create a new build configuration.
5. Replace the **Debug** text in the *Condition* attribute with **DirectToHs**
6. Change the *OutputPath* element content to **$(HomeSeerRoot)**

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'DirectToHs|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>none</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>$(HomeSeerRoot)</OutputPath>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <Prefer32Bit>false</Prefer32Bit>
    <DebugSymbols>false</DebugSymbols>
</PropertyGroup>
```

7. Scroll down to the bottom of the project file, and, at the end of the *ItemGroup* elements, add a new *ItemGroup* that points to the local HTML folder for your project

```
<ItemGroup>
    <HtmlFiles Include="html\*">
      <InProject>false</InProject>
    </HtmlFiles>
</ItemGroup>
```

8. Finally, as the last item in the *Project* element, add a *Target* element that handles copying the HTML files to the HomeSeer html folder as expected like so

```
<Target Name="AfterBuild" Condition=" '$(Configuration)|$(Platform)' == 'DirectToHs|AnyCPU' ">
    <Copy SourceFiles="@(HtmlFiles)" DestinationFolder="$(OutputPath)\html\$(PluginId)" />
</Target>
```

9. Save the project file and reload the project
10. Build your project using the new **DirectToHs** build configuration
11. Start your plugin through the HomeSeer UI

Going further

Now that you have built and deployed your plugin to HomeSeer, you can start it up and see it run alongside HomeSeer on the *Manage Plugins* page in the web UI. The plugin currently doesn't do anything in its current state, but, what it does from here, is entirely up to you.