Get up to speed on developing your own Ghost adapters quick and easy.

GHOST NODES

Adapter Developer Guide

Version 1.1 January 2021

Ghost Nodes is a lightweight distributed integration platform, that is intended to be installed on existing hardware/infrastructure. The solution is built up of connected Gateways and Agents, where the Gateways are parenting one or more Agents or another Gateway. Ghost Studio is used to create, manage and deploy integrations in the distributed landscape. When an integration is deployed the individual adapters and processors that makes up the integration is pushed down to the Agents through the Gateways.

Ghost runtime is heavily dependent on adapters. An adapter is a software component that receive, send or process messages from a specific endpoint. All endpoint processing is done from adapters which communicates with external systems, applications, protocols etc. Adapters are bound together to form a workflow we call Links. A Link can have multiple bindings from a variety of adapters. Ghost supports three types of adapters: Source-, Destination- and CustomProcess-Adapter.

The adapter should be simple, with one single purpose only. To solve complex tasks you combine several adapters instead of building one that does everything.

Ghost provides some native adapters that supports various protocols such as: File, Ftp, SQL, Http, Mqtt, IBM MQ, Azure ServiceBus, Azure Storage, Smtp. (For a complete list of native adapters use the Ghost Studio)

If you are unable to locate an adapter to support your communication requirements, you can develop your own custom adapter.
This document will try to explain and guide you through the process of building your own custom adapters.

Contents

1: Pre-Requisites

To be able to develop custom adapters you will need:

- Windows 10
- .Net Core 3.1 SDK
- Visual Studio 2019
- Ghost Project Templates package

2: Adapter Types

Ghost have three types of adapters; Source-, Destination- and CustomProcess-Adapter. Each type have a specific role and implementation.

Source-Adapter:

This type of adapters will receive messages from an endpoint and will be the start of a Link. A Link can contain multiple sources that merge and/or splits to multiple adapters.

Destination-Adapter:

Destination adapters will send messages to an endpoint and can be seen as a final step in a Link.

CustomProcess-Adapter:

These adapters are used when a message needs to be modified in some way. Execution context depends on which Node the adapter is bound to. For example, if a custom adapter is bound to a destination that has a remote Agent it will be executed on the remote agent and not where the Source executes.

3: Request-Response

All adapters can act as a two-way adapter even if no code is written specifically for that. The Link can be configured to return custom content or even the incoming message. You can however override the “ReturnResponse” method to be able to write adapter specific code when returning messages.

4: Sync vs Async

Best practice when choosing which methods to override is to go with all Sync methods, or all Async. You can combine to override from both but it is not advised. All adapter methods have both implementations available.

5: Message Engine

The Message-Engine is the glue that controls and processes each adapter and will make sure that every message gets processed and routed according to the Link.
The engine will load each adapter and its dependencies in separate LoaderContext so no NuGet package version conflicts occurs.

6: Multithreading

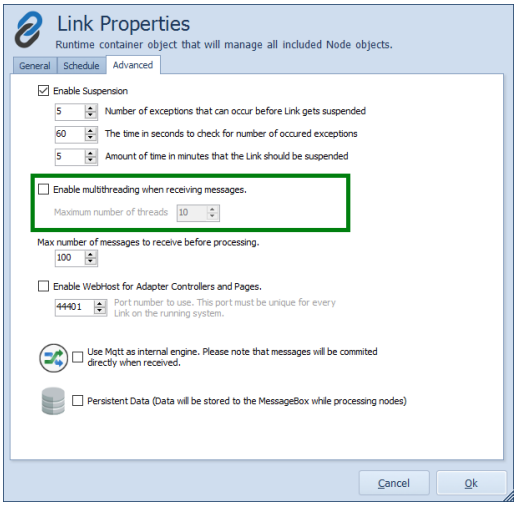
A source-adapter can be coded to support multiple threads. There is a special base property called "*SupportsMultiThreading*" which has to be set to true in the adapter constructor.

```
public Startup()
{
    Id = Guid.Parse("ABCDEFABC-1234-5678-0000-123456789123");
    Description = "My custom Source adapter for receiving messages.";
    SupportsMultiThreading = true;
}
```

Example of parallel code:

```
IEnumerable<string> matches = Directory.EnumerateFiles(m_NodeParams.Path);
if (base.MultiThreadingEnabled && base.MaxThreads > 1)
{
    var result = Parallel.ForEach(matches,
        new ParallelOptions
        {
            MaxDegreeOfParallelism = base.MaxThreads,
            CancellationToken = CancellationToken
        },
        (path, state, index) =>
        {
            byte[] data = System.IO.File.ReadAllBytes(path);
            MessageReceived(new AdapterMessage(data), path);
        });
}
```

To enable the multithreading, you will have to check the "Enable multithreading when receiving messages" check box in Link Properties.



There is also an option to set how many parallel threads that can run at once.

7: Ghost Project Templates

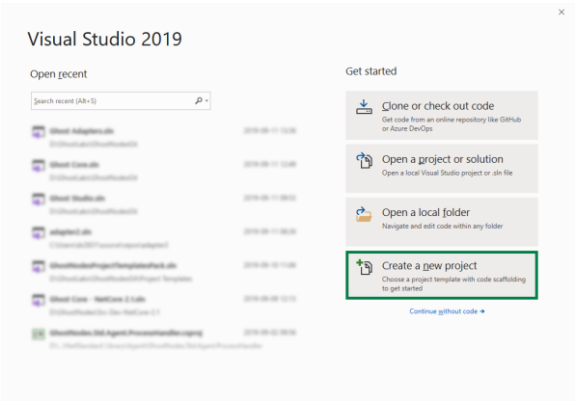
When developing custom adapters is can be very tricky to know what to include such as classes, dependencies, resources etc.

Just to make things easier we have developed a Visual Studio template package that will create skeleton adapter projects with everything already included. All you need to do is write you custom code for that specific adapter.

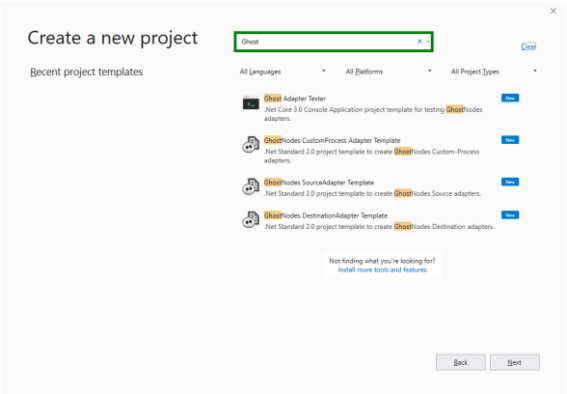
The package can be downloaded from the Ghost homepage: https://ghostnodes.com/

Once downloaded you need to install the package so it can be used from within Visual Studio.

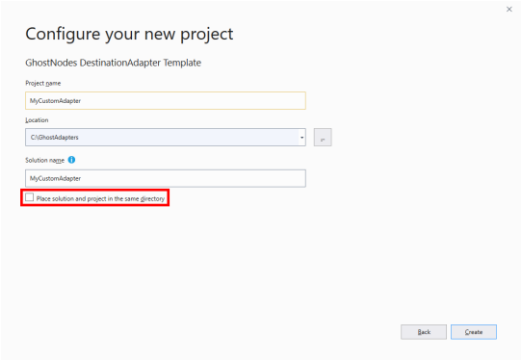
To create an adapter project open Visual Studio and select “Create a new project”



In the search box type “Ghost” and all adapter template projects will be visible.



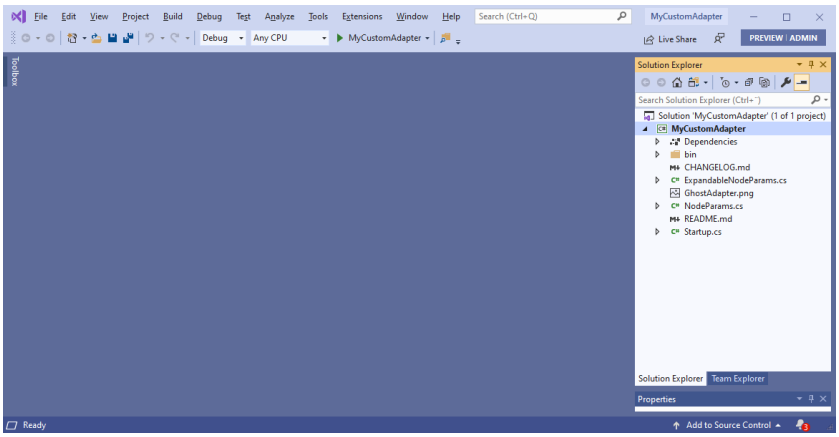
Select the type of project you want and click ‘Next’



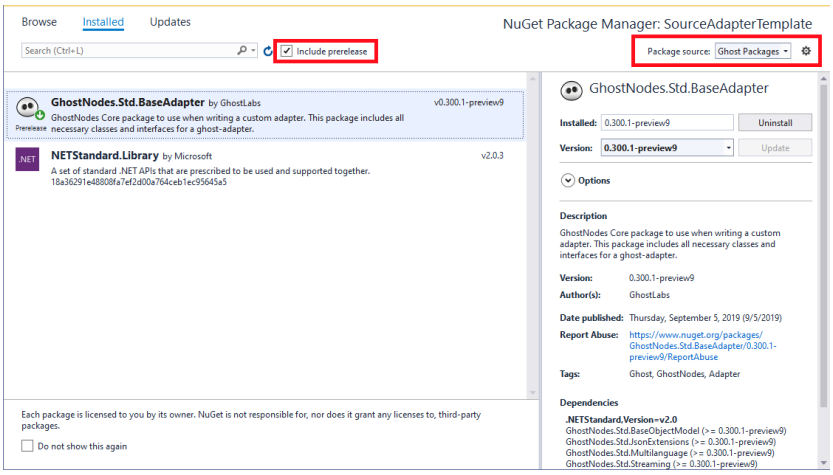
Name your project as the name of your adapter. Please do NOT use dots '.' in the name which will confuse the adapter discovery features later on.

Also it is not advised to check the "Place solution and project in the same directory". This is because you most certainly want to add an "Ghost Adapter Tester" project to the same solution so you can debug your adapter in Visual Studio.

Click 'Create' to finalize the wizard and you will end up with a skeleton project for your adapter.



Once the project is created please check that it uses the latest versions of Ghost packages.
Right click on the project and select "Manage NuGet packages…" menu item.
Make sure that "Include prerelease" is checked and that "Ghost Packages" package source is selected from the dropdown list.



8: Adapter files

The template project will generate some skeleton files and depending on the type of adapter the code within will wary.

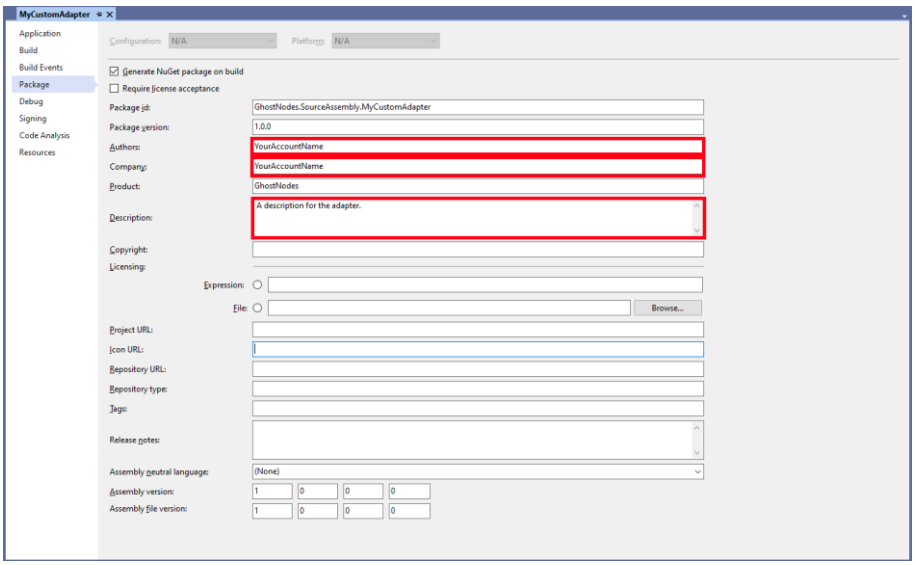
Generated files summary:

Filename	Description

CHANGELOG.md	This markdown file should contain version numbers and what significant changes has been made. This file must have: Build Action=**Content**
README.md	This markdown file should contain a description for the adapter. This file must have: Build Action=**Content**
ExpandableNodeParams.cs	Here you define all expandable params that the adapter will generate if any.
NodeParams.cs	Here you define all parameters that will control the adapter and its usage.
Startup.cs	Main class of the adapter. Here you select which overrides to use and call adapter specific code.
GhostAdapter.png	Adapter image for easy recognition. This file must have: Build Action=**Embedded resource**

9: Project Properties

In the project properties dialog it's important that you change some values for the Package. Authors and Company must be the same and be the same as the Account the adapter will deployed for. Also a decent Description is required.



10: Expandable Node Params

Expandable node params is a list value pairs that the adapter will populate for every processed message. The list of params will be passed with the message and can be used later on in other adapter configurations or in message routing.

11: Node Params

Node params contains all the parameters that the adapter need to have to be able to run as expected. Examples of parameters could be a connection string, a path, a threshold value.

11.1 Node Param Attributes

Every property included must have some attributes defined because they are used for auto-generating documentation/help for the adapter.

Mandatory Attributes

Attribute name	Description	Example
DisplayName	A readable string of the property which can contain spaces.	[DisplayName("Path To Files")]
Description	A description of the property.	[Description("Path to a local folder or a network share where files will be read from.")]
Category	Specifies the name of the category in which to group the property when displayed in Ghost Studio.	[Category("Mandatory")]
DefaultValue	The default value of the property used when auto generating documentation.	[DefaultValue("")]

Optional Attributes

Attribute name	Description	Example
IsNodeKey	Set this attribute on every property that combined will generate a unique id for every message.	[IsNodeKey(true)]
IsRefreshable	Set to true on properties that can be updated in runtime without restarting the process.	[IsRefreshable(true)]
AcceptDefaultValue	Tells the configuration if the default value is acceptable or needs to be changed before deploying.	[AcceptDefaultValue(false)]
EditorAttribute	Add a property editor for a complex property.	[EditorAttribute("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserUITypeEditor", "System.Drawing.Design.UITypeEditor")]
TypeConverter	Add a type converter for specified editor.	[TypeConverter("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserTypeConverter")]
SampleValues	A list of sample values that will be used when auto generating documentation.	[SampleValues(@"C:\Local\Folder", @"X:\Mapped\Folder", @"\\Server\Shared\Folder")]
PropertyOrder	Set this attribute with ascending values to sort properties. If not specified properties will be sorted by its name.	[PropertyOrder(10)]
MaxStringLength	Sets a max length of a string.	[MaxStringLength(64)]
ObscureText	This will obscure a text field hiding its characters. Should be used on password fields.	[ObscureText()]
OtherPropertyEnabled, OtherPropertyDisabled	A list of other properties that will be enabled/disabled when this property changes value. Properties of type: 'bool' will validate as is. 'string' will fail to validate on empty or null strings. All other types have to set the SuccessValue which determinates validation.	[OtherPropertyEnabled("ReplyPath,ReplyFilename")]
OtherPropertyVisibility	A list of other properties that will be shown/hidden when this property changes value. Properties of type:	[OtherPropertyVisibility("UseSystemProxy,UseWebProxy ")]

	'bool' will validate as is. 'string' will fail to validate on empty or null strings. All other types have to set the SuccessValue which determinates validation.	
OtherPropertyUnckeck	A list of properties that will be checked/unchecked depending in this property value.	[OtherPropertyUnckeck("UseWebProxy")]
MinIntValue	Only applies to 'int' properties.	[MinIntValue(1)]
MaxIntValue	Only applies to 'int' properties.	[MaxIntValue(100)]

Example of a complete NodeParams file:

```
[Export(typeof(INodeParametersBase))]
 public class NodeParams : NodeParametersBase<NodeParams>
 {
     public NodeParams()
     {
         // Mandatory
         Path = string.Empty;
         Filter = "*";

         // Optional
         SortOptions = FileSortOptions.None;
         MinFileAge = 0;
         IncludeSubFolders = false;
         IncludeEmptyFiles = false;

         // Response
         IsTwoWay = false;
         ReplyPath = string.Empty;
         ReplyFilename = string.Empty;
     }

     public static NodeParams Extract(List<Param> cfg)
     {
         return ExtractParams(cfg);
     }
     [IsNodeKey(true)]
     [IsRefreshable(true)]
     [DisplayName("Path")]
     [Description("Path to a local folder or a network share where files will be read from.")]
     [Category("Mandatory"), DefaultValue(""), AcceptDefaultValue(false)]
     [EditorAttribute("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserUITypeEditor",
"System.Drawing.Design.UITypeEditor)")]
     [TypeConverter("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserTypeConverter")]
     [SampleValues(@"C:\Local\Folder", @"X:\Mapped\Folder", @"\\Server\Shared\Folder")]
     [PropertyOrder(10)]
     public string Path { get; set; }

     [IsNodeKey(true)]
     [IsRefreshable(true)]
     [DisplayName("Filter"), Description("Collect only files that match specified Filter. Multiple
filter are separated by any of , ; | characters."), Category("Mandatory"), DefaultValue("*"),
AcceptDefaultValue(true)]
     [SampleValues(@"*.txt", @"*.xml|*.json", @"prefix*.txt,*postfix.txt")]
     [PropertyOrder(11)]
     public string Filter { get; set; }

     [IsRefreshable(true)]
     [DisplayName("SortOptions"), Description("The order the files in the Path folder will be picked
up."), Category("Mandatory"), DefaultValue("None"), AcceptDefaultValue(true)]
     [PropertyOrder(20)]
     public FileSortOptions SortOptions { get; set; }

     [IsRefreshable(true)]
     [DisplayName("Minimum File Age"), Description("Time in seconds on how old a file need to be
before it can be picked up."), Category("Optional"), DefaultValue(0), AcceptDefaultValue(true)]
     [PropertyOrder(23)]
     public int MinFileAge { get; set; }
```

```
        [IsRefreshable(true)]
        [DisplayName("Include sub-folders"), Description("Should sub-folders be included."),
Category("Optional"), DefaultValue(false), AcceptDefaultValue(true)]
        [PropertyOrder(24)]
        public bool IncludeSubFolders{ get; set; }

        [IsRefreshable(true)]
        [DisplayName("Include Empty Files"), Description("Set to True if empty files should be picked
up and processed."), Category("Optional"), DefaultValue(false), AcceptDefaultValue(true)]
        [PropertyOrder(25)]
        public bool IncludeEmptyFiles { get; set; }

        [IsRefreshable(true)]
        [DisplayName("Is Two Way"), Description("True if a response message should be returned."),
Category("Response"), DefaultValue(false), AcceptDefaultValue(true)]
        [PropertyOrder(30)]
        [OtherPropertyEnabled("ReplyPath,ReplyFilename")]
        public bool IsTwoWay { get; set; }

        [IsRefreshable(true)]
        [DisplayName("Reply Path"), Description("Path where response messages will be sent to."),
Category("Response"), DefaultValue(""), AcceptDefaultValue(true)]
        [Editor("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserUITypeEditor",
"System.Drawing.Design.UITypeEditor)")]
        [TypeConverter("GhostNodes.Win.AdapterUIEditors.UIEditors.FolderBrowserTypeConverter")]
        [PropertyOrder(31)]
        public string ReplyPath { get; set; }

        [IsRefreshable(true)]
        [DisplayName("Reply Filename"), Description("Filename format for response messages (Ex:
%guid%.txt)."), Category("Response"), DefaultValue(""), AcceptDefaultValue(true)]
        [PropertyOrder(32)]
        public string ReplyFilename { get; set; }
    }
```

12: Adapter Testing

There is a special project type “Ghost Adapter Tester” for testing and debugging an adapter included in the adapter package. When created you will simulate the message processing and build your Link from code instead from the Ghost Studio. You should create the Tester project in the same solution as the adapter you want to test.

The Link generation is done by creating a MockupSetup class in which you add adapters, create Nodes and do all the bindings that will form the Link.

There are two ways of adding an Adapter; by local code or by a NuGet package.

```
AdapterConfigInfo file_SourceAssembly = new AdapterConfigInfo()
{
    NuGetPkg = "GhostNodes.SourceAssembly.File"
};
```

OR

```
AdapterConfigInfo file_SourceAssembly = new AdapterConfigInfo()
{
    FullPath = @"C:\GhostAdapters\File\bin\Debug\netstandard2.0\" +
               "GhostNodes.SourceAssembly.File.dll",
};

setup.Adapters.Add(file_SourceAssembly);
```

When adapter is loaded from a NuGet package the adapter will be downloaded from the Ghost NuGet repository. This is so you may use existing adapters to build a Link with your own custom adapter you want to debug and test.

In the other option you need to point to the adapter assembly file which should be on your local file system.

To configure the adapter, you will need to create a NodeConfigInfo object that uses the adapter.

```
NodeConfigInfo file_SourceNode1 = new NodeConfigInfo()
{
    Name = "S1",
    NodeType = NodeType.Source,
    AdapterConfigInfo = file_SourceAssembly,
    Parameters = new List<Param>()
    {
        Param.Create("Path", @"C:\GhostTest\DropFolder")),
        Param.Create("UseMultiThreading", true),
        Param.Create("MaxThreads", 10),
    }
};
```

Here you will need to match the NodeType correctly and at a minimum add Params for all mandatory properties.

To finally build the Link you need to add and bind the Nodes.

```
setup.AddNode(file_SourceNode1);
setup.AddNode(envelope_CustomProcessNode1,
    new List<Guid>() { file_SourceNode1.UniqueId });
setup.AddNode(file_DestinationNode1,
    new List<Guid>() { envelope_CustomProcessNode1.UniqueId });
```

Then the Link is finally ready to be processed. This is done by starting the Message-Engine

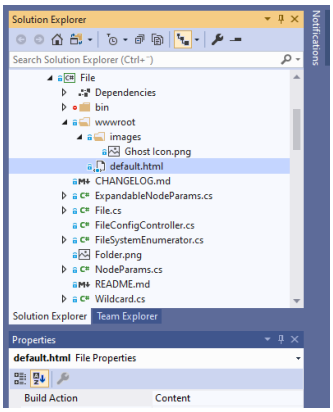
```
setup.StartEngine();
```

Set Adapter Tester project as “Startup Project” and set some breakpoints in the Adapter.
Hit F5 to start debugging…

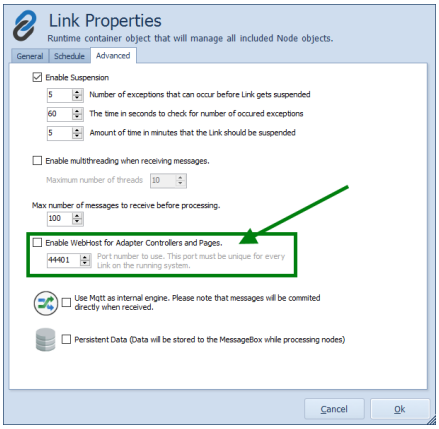
13: Embedded WebHost

Sometimes you might want to include web pages into your adapter. It could be anything from extra configuration possibilities to extended logging.

To solve this, we have made it possible to add a wwwroot folder to the adapter.
Include your html-, image-, css-files etc. as Content files. This will make sure the files will be included in the Adapter NuGet package.



Also the Link will have to be configured to enable WebHosts.



When the Message-Engine is started it will search all adapter in the Link for a wwwroot content folder. If one is found that adapter web-pages will be used for the WebHost.

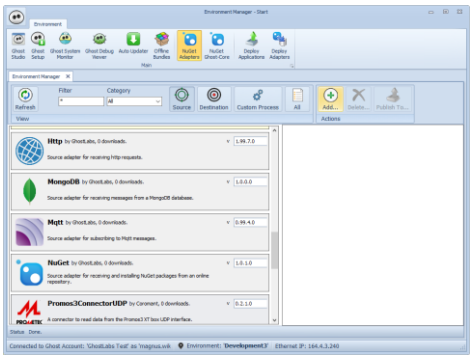
Search order is: Source-, Destination- and CustomProcess-Adapters.

14: Publishing Adapters

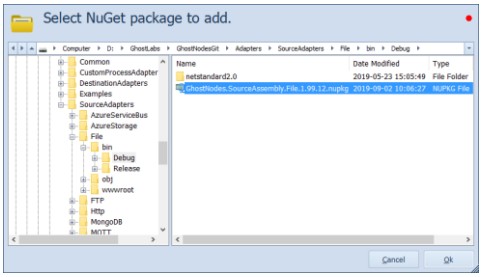
When an Adapter done, tested and bug free it needs to Published to the Ghost NuGet repository and Deployed to a Ghost Environment.

This is a two-step deal.

First you need to start Ghost Studio and login to the same Account as the adapter is built for. If Accounts mismatch you will NOT be able to publish anything. Click “Manage Environment”. Go to the “NuGet Adapter” and click “Add...” button.



Browse to the Adapter package and select Ok.

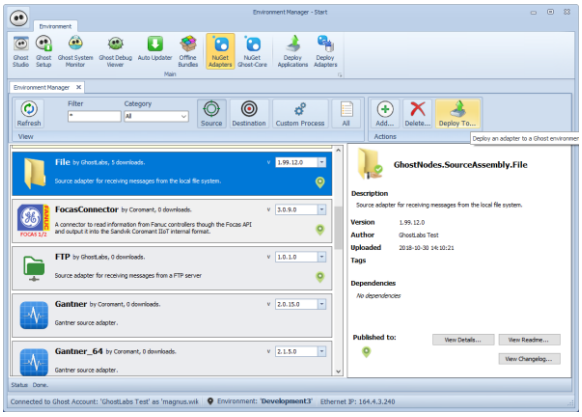


If everything is ok… the adapter package should be published to the NuGet repository.

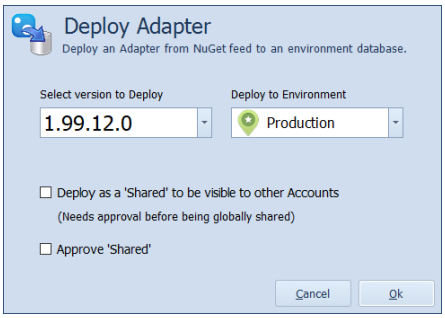
Once the package is in the repository it need to be deployed to an Environment. It will not be visible anywhere if this step is not done.

NOTE: When publishing an adapter to NuGet it will take a while (1-2 minutes) before it can be seen. So, it is important to Refresh the adapter list and make sure your adapter with the correct version is listed before you deploy it.

Once the correct version is available from the list click “Deploy To…”



In the “Deploy Adapter” dialog you select the version and environment the adapter should be deployed to.



Press ‘Ok’ and the adapter will be deployed and visible in the selected environment.

‘Deploy as Shared’ should be checked only if the adapter should be made public available. But the adapter has to be approved by Ghost before it becomes available to anyone.

15: Adapter Examples

We have a Git repository with some example adapters. These examples are not meant to be deployed only for educational purposes, but they are exact copies of the real adapters in use by Ghost.

The examples are:

Source Adapter: **File**
Destination adapter: **File**
Custom Process Adapter: **ReplaceText**

Git repo: https://github.com/GhostLabsAB/Examples