Using the Object Gateway for .NET with Ensemble

Version 2018.1
2024-05-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using the Object Gateway for .NET with Ensemble
Ensemble Version 2018.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book explains how to enable easy interoperation between Ensemble and Microsoft .NET Framework components. The Object Gateway for .NET can instantiate an external .NET object and manipulate it as if it were a native object within Ensemble. This book uses the shorter term, .NET Gateway.

This book contains the following chapters:

- .NET Gateway Architecture
- Using the .NET Gateway in a Production

For a detailed outline, see the table of contents.

The following books provide related information:

- *Ensemble Best Practices* describes best practices for organizing and developing Ensemble productions.
- *Developing Ensemble Productions* explains how to perform the development tasks related to creating an Ensemble production.
- *Configuring Ensemble Productions* describes how to configure the settings for Ensemble productions, business hosts, and adapters. It provides details on settings not discussed in this book.
- *Using the Caché Gateway for .NET* explains how to instantiate an external .NET object and manipulate it as if it were a native object within Caché.

For general information, see the *InterSystems Documentation Guide*.

1

.NET Gateway Architecture

The Object Gateway for .NET provides an easy way for Ensemble to interoperate with Microsoft .NET Framework components. The .NET Gateway can instantiate an external .NET object and manipulate it as if it were a native object within Ensemble.

The external .NET object is represented within Ensemble by a "wrapper" or "proxy" class. The proxy object appears and behaves just like any other Ensemble object, but it has the capability to issue method calls out to the Common Language Runtime (CLR), either locally or remotely over a TCP/IP connection. Any method call on the proxy object triggers the appropriate class method inside the CLR.

You can use the .NET Gateway to create proxy classes for custom .NET components. However, the most powerful feature of the .NET Gateway is that it easily creates proxy mappings to entire .NET application interface specifications, such as ADO, Remoting, ASP.Net, etc. Ensemble can then use these applications without having to generate new proxy classes.

Note: The generated proxy class is an Ensemble class. You can access it with code written in either Caché Basic or ObjectScript. The examples in this document use ObjectScript.

In general, the best approach to using the .NET Gateway is to build a small wrapper class that exposes just the functionality you want and then create a proxy for this wrapper. This makes the API between Ensemble and the .NET framework very clean and eliminates potential issues dealing with how to map more esoteric features to a proxy object.

The *Using the Caché Gateway for .NET* guide provides complete technical details of using the .NET Gateway. The next chapter shows how to specifically use the .NET Gateway in Ensemble.

2

Using the .NET Gateway in a Production

There are different ways to invoke the basic .NET Gateway **Start**, **Connect**, **Import**, **Disconnect**, and **Stop** commands. Practical approaches to this functionality include:

- Creating a Business Service
- Calling Business Service Methods
- Creating a Business Operation
- Calling API Methods
- Using the Command Prompt
- Using the .NET Gateway Wizard

This chapter describes each approach and explains how to work with it. Related topics include:

- Error Checking
- Troubleshooting

2.1 Creating a Business Service

While it is possible to start the .NET Gateway server from the command prompt, the simplest way to use the .NET Gateway with an Ensemble production is to configure the EnsLib.DotNetGateway.Service class as a business service within the production. You can only do this if the .NET Gateway server is on the local machine where you are running Ensemble.

Otherwise, you need to start the .NET Gateway server from the command prompt. For details, see the Using the Command Prompt section.

To configure the EnsLib.DotNetGateway.Service class as a business service:

1. From the Management Portal main menu, choose **Productions**.
2. Find your production in the list and click **Configure** beside its name.
3. Click **Add Service** to start the Business Service Wizard.
4. Click **Other** and in the **ServiceClass** list, click **EnsLib.DotNetGateway.Service**.
5. Click **OK** to display the updated production diagram that now contains the .NET Gateway business service. Click the **EnsLib.DotNetGateway.Service** box to configure it.

The wizard fills in the associated .NET Gateway adapter class. The Business Service Settings for the .NET Gateway section lists the configurable settings.

2.1.1 Business Service Settings for the .NET Gateway

Dot Net Server

IP address or name of the machine where the .NET Gateway server executable is located.

Port

TCP port number for communication between the .NET Gateway server and the proxy classes in Ensemble. The default is `55000`.

File Path

Location of the .NET Gateway server executable. It is used to find the target executable and assemble the command to start the .NET Gateway on a local server. If you do not specify this setting, the service uses the default directory ...\Dev\dotnet\bin\ under the Ensemble installation directory.

Allowed IP Addresses

IP addresses allowed to connect to the .NET Gateway server. If this setting is `0.0.0.0` (default) or `""`, any system (local or remote) may connect; otherwise any listed IP addresses are allowed to connect.

Exec64

If you select this check box, the business service uses the 64–bit version of the .NET Gateway executable. Otherwise, it uses the 32–bit version of the executable. This setting is available only on Windows 64-bit platforms.

.NET Version

Select the .NET version to use: 2.0 or 4.0.

Log File

Full pathname of the file where the .NET Gateway logs messages. These messages include acknowledgment of opening and closing connections to the server, and any difficulties encountered in mapping .NET classes to Ensemble proxy classes.

Heartbeat Interval

Number of seconds between each communication with the .NET Gateway to check whether it is active. When enabled, the minimum value is 5 seconds and the maximum value is 3600 seconds (1 hour). The default is 10 seconds. A value of 0 disables this feature.

Heartbeat Failure Timeout

Number of seconds without responding to the heartbeat, to consider that the .NET Gateway is in failure state. If this value is smaller than the HeartbeatInterval property, the gateway is in failure state every time the .NET Gateway communication check fails. The maximum value is 86400 seconds (1 day). The default is 30 seconds.

Heartbeat Failure Action

Action to take if the .NET Gateway goes into a failure state. Setting it to Restart (default) causes the .NET Gateway to restart. Setting it to Alert generates an alert entry in the Event Log. This is independent of the **Alert on Error** setting.

Heartbeat Failure Retry

Time to wait before retrying the HeartbeatFailureAction if the .NET Gateway server goes into failure state, and stays in failure state. The default is 300 seconds (5 minutes). A value of 0 disables this feature, meaning that once there is a failure that cannot be immediately recovered, there are no attempts at automatic recovery.

See "Settings in All Business Services" in *Configuring Ensemble Productions*.

Once you have added and configured the .NET Gateway business service, it automatically manages the .NET Gateway as follows:

- When the production starts, the .NET Gateway business service starts an instance of the .NET Gateway server, using the settings that you specify on the configuration page.
- When the production receives a signal to stop, the .NET Gateway business service attaches to the .NET Gateway server and instructs it to stop, as well.

For more information, see the EnsLib.DotNetGateway.Service entry in the *Class Reference*.

2.2 Calling Business Service Methods

The .NET Gateway business service provides methods that you can use to start, connect to, and stop the .NET Gateway engine. You can call the following methods from Ensemble code after you have configured the .NET Gateway business service as a member of the production:

- StartGateway()
- ConnectGateway()
- StopGateway()

See the EnsLib.DotNetGateway.Service entry in the *Class Reference* for details on these methods.

2.2.1 StartGateway() Method

```
EnsLib.DotNetGateway.Service:StartGateway(pFilePath As %String,
     pPort As %String,
     pAllowedIPAddresses As %String,
     pLogfile As %String = "",
     ByRef pDevice As %String = "",
     pServer As %String = "127.0.0.1",
     pCmdLine As %String = "")
```

This class method starts the .NET Gateway server using the specified arguments. If *pLogFile* specifies a valid file name, then messages regarding gateway activities are written to this file. These messages include acknowledgment of opening and closing connections to the server, and difficulties encountered (if any) in mapping .NET classes to Ensemble proxy classes.

2.2.2 ConnectGateway() Method

```
EnsLib.DotNetGateway.Service:ConnectGateway(pEndpoint As %String,
     ByRef pGateway As %Net.Remote.Gateway,
     pTimeout As %Integer = 5,
     pAdditionalPaths As %String = "")
```

This class method connects to the .NET Gateway server at the specified *pEndpoint* (hostname:port:namespace).

2.2.3 StopGateway() Method

```
EnsLib.DotNetGateway.Service:StopGateway(pPort As %String,
     pServer As %String = "127.0.0.1",
     pTimeout As %Integer = 5)
```

This class method connects to the .NET Gateway server and shuts it down.

2.3 Creating a Business Operation

An abstract business operation is available as a base for building .NET Gateway oriented business operations for Ensemble productions. You can simply subclass the abstract class EnsLib.DotNetGateway.AbstractOperation and implement the appropriate message handlers.

Call the **GetConnection()** method to verify there is a valid .NET Gateway connection. For example:

ObjectScript

```
 Set tSC = ..GetConnection(.tGateway)
     If $$$ISOK(tSC) {
        // Start using the .NET Gateway connection object tGateway
      ...
     }
```

This method returns a private gateway connection object to be used with the proxy classes.

You can configure the .NET Gateway IP address and port in the business operation settings when you add the business operation to the production. Note that the connection to the .NET Gateway instance is made during **OnInit()** and closed in **OnTearDown()**. You must override these methods in the business operation class to implement your own setup and tear down procedures.

See the EnsLib.DotNetGateway.AbstractOperation entry in the *Class Reference* for details on these methods and also the AdditionalPaths, ConnectTimeout, DotNetServer, and Port properties.

2.4 Calling API Methods

In addition to using connect, disconnect, and stop from the business service, the following methods are also available in the %Net.Remote.Gateway class. You can use them when the business service model is not appropriate for your situation:

The %Net.Remote.Gateway class provides the following types of methods:

- API methods that let you %Connect to the .NET Gateway server, %Disconnect from it, and %Shutdown the .NET Gateway server.
- The %Import method, which imports .NET classes or assemblies from the .NET and generates all the necessary proxy classes for the Ensemble side.
- The %ExpressImport method, which combines calls to **%Connect**, **%Import**, and **%Disconnect**.
- The utility method %GetAllClasses.

2.4.1 %Connect() Method

```
Method %Connect(host As %String,
                port As %Integer,
                namespace As %String,
                timeout As %Numeric = 5,
                additionalClassPaths As %ListOfDataTypes = "")
    As %Status [ Final ]
```

The **%Connect()** method establishes a connection with the .NET Gateway engine. It accepts the following arguments:

Argument	Description
host	Identifies the machine on which the .NET Gateway server is running.
port	Port number over which the proxy classes communicate with the .NET classes.
namespace	Ensemble namespace.
timeout	Number of seconds to wait before timing out, the default is 5.
additionalClassPaths	Optional — use this argument to supply additional class paths, such as the names of additional assembly DLLs that contain the classes you are importing via the .NET Gateway. See the Import Arguments section for details using this argument.

2.4.2 %Disconnect() Method

```
Method %Disconnect() As %Status [ Final ]
```

The **%Disconnect()** method closes a connection to the .NET Gateway engine.

2.4.3 %Shutdown() Method

```
Method %Shutdown() As %Status [ Final ]
```

The **%Shutdown()** method shuts down the .NET Gateway engine.

2.4.4 %Import() Method

```
Method %Import(class As %String,
               ByRef imported As %ListOfDataTypes,
               additionalClassPaths As %ListOfDataTypes = "",
               exclusions As %ListOfDataTypes = "")
    As %Status [ Final ]
```

The **%Import()** method imports the given *class* and all its dependencies by creating and compiling all the necessary proxy classes. The **%Import()** method returns, by reference, a list (in *imported*) of generated Ensemble proxy classes. For details of how .NET class definitions are mapped to Ensemble proxy classes, see the "Mapping Specification" chapter in *Using the Caché Gateway for .NET* guide.

%Import() is a onetime, startup operation. It only needs to be called the first time you wish to generate the Ensemble proxy classes. It is necessary again only if you recompile your .NET code and wish to regenerate the proxies. The following sections provide further details about the **%Import()** method:

- Import Arguments
- Import Dependencies and Exclusions

2.4.4.1 Import Arguments

Before you invoke **%Import()**, prepare the *additionalClassPaths* and *exclusions* arguments. That is, for each argument, create a new %ListOfDataTypes object and call its **Insert()** method to fill the list. The optional *additionalClassPaths* argument can be used to supply additional path arguments, such as the names of additional assembly DLLs that contain the classes you are importing via the .NET Gateway. List elements should correspond to individual additional assembly DLL entries, which require the following format:

```
" rootdir\...\mydll.dll "
```

You can try to load an assembly from a directory outside of where DotNetGatewaySS.exe is running, but you might experience a load error for your assembly when you try to use a class in the assembly. InterSystems recommends that you put all local assemblies in the same directory as DotNetGatewaySS.exe. You can also specify assemblies in the GAC by using partial names for them, `System.Data`, for example.

2.4.4.2 Import Dependencies and Exclusions

While mapping a .NET class into an Ensemble proxy class and importing it into Ensemble, the .NET Gateway loops over all class dependencies discovered in the given .NET class, including all classes referenced as properties and in argument lists. In other words, the .NET Gateway collects a list of all class dependencies needed for a successful import of the given class, then walks that dependency list and generates all necessary proxy classes.

You can control this process by specifying a list of assembly and class name prefixes to exclude from this process. While this situation would be rare, it does give you some flexibility to control what classes get imported. The .NET Gateway automatically excludes a small subset of assemblies such as Microsoft.* assemblies.

2.4.5 %ExpressImport() Method

```
ClassMethod %ExpressImport(name As %String,
                           port As %Integer,
                           host As %String = "127.0.0.1",
                           silent As %Boolean = 0,
                           additionalClassPaths As %ListOfDataTypes = "",
                           exclusions As %ListOfDataTypes = "")
    As %Status [ Final ]
```

%ExpressImport() is a one-step convenience class method that combines calls to **%Connect()**, **%Import()**, and **%Disconnect()**. It returns a list of generated proxies. It also logs that list, if the *silent* argument is set to 0. The *name* argument is a semicolon-delimited list of classes or assembly DLLs.

2.4.6 %GetAllClasses()Method

```
 %GetAllClasses(jarFileOrDirectoryName As %String,
                ByRef allClasses As %ListOfDataTypes)
     As %Status
```

This method returns, in the ByRef argument *allClasses*, a list of all public classes available in the assembly DLL specified by the first argument, *jarFileOrDirectoryName*.

2.5 Using the Command Prompt

Usually you start and stop the .NET Gateway server automatically, by configuring the EnsLib.DotNetGateway.Service business service as a member of the production. Once this is done, the .NET Gateway server starts and stops automatically with the production. The **StartGateway()** class method is also available to manually start the .NET Gateway server.

However, during development or debugging, or when Ensemble and the .NET Gateway server run on different machines, you may find it useful to start the gateway server from a command prompt. Copy the file DotNetGatewaySS.exe to the directory where you load an assembly. By default, DotNetGatewaySS.exe is shipped in the directory *install-dir*\dev\dotnet\bin. Run DotNetGatewaySS.exe from *install-dir*\dev\dotnet\bin as follows:

`DotNetGatewaySS` *port listener logfile*

Argument	Description
port	Port number on which to listen for the incoming requests.
listener	*Optional* — Contains the local IP address on the local machine where the gateway listens. Specify null, `""`, or `0.0.0.0` (the default) to listen on *all* IP addresses local to the machine (127.0.0.1, VPN address, etc.) You can restrict the listener to one existing local IP address or listen on all of them; you cannot enter a list of acceptable addresses. You must provide a value for this argument if you are specifying a *logfile*.
logfile	*Optional* — If specified, the command procedure creates a log file of this name; you must specify the full pathname in the string. The *listener* argument is required if you enter a value for *logfile*.

For example:

```
DotNetGatewaySS 55000 "" ./gatewaySS.log
```

Note: When using classes in local side-by-side assemblies (assemblies are not installed into the GAC), run DotNetGatewaySS.exe from the same directory as those assemblies to resolve their dependencies.

2.6 Using the .NET Gateway Wizard

You can import a DLL assembly file from .NET and create a set of corresponding classes using the .NET Gateway wizard built into Studio. To start the wizard:

1. Start Studio.
2. From the **Tools** menu, point to and click **Add-Ins**.
3. Click **.NET Gateway Wizard** to start the .NET Gateway Wizard dialog.
4. Enter the path and name of a DLL assembly file; or click **Browse** to help navigate to one.
5. Enter the **.NET Gateway server name / IP address** and **.NET Gateway server port** for the .NET Gateway server.
6. You can also enter **Additional paths\assemblies to be used in finding dependent classes** and **Exclude dependent classes matching the following prefixes** as instructed in the dialog.
7. Click **Next** to generate Ensemble proxy classes. The wizard displays the class name as it generates each proxy class.
8. When the import operation is complete, click **Finish** to exit the wizard.

2.7 Error Checking

The .NET Gateway provides error checking as follows:

- When an error occurs while executing Ensemble proxy methods, the error is, in most cases, a .NET exception, coming either from the original .NET method itself, or from the .NET Gateway engine. When this happens, an error is trapped.
- The .NET Gateway API methods like **%Import()** or **%Connect()** return a typical Ensemble %Status variable.

In both cases, Ensemble records the last error value returned from a .NET class (which in many cases is the actual .NET exception thrown) in the local variable *%objlasterror*.

You can retrieve the complete text of the error message by calling **$system.OBJ.DisplayError()**, as follows:

ObjectScript

```
Do $system.OBJ.DisplayError(%objlasterror)
```

2.8 Troubleshooting

Should you encounter problems while using the .NET Gateway it is always a good idea to turn logging on. That might be necessary for InterSystems staff to help you troubleshoot problems. To activate logging, simply identify a log file when you start the .NET Gateway. You can do this whether you start from the command line or use the **StartGateway()** API method.

Sometimes, while using the .NET Gateway in a debugging or test situation, you may encounter problems with a Terminal session becoming unusable, or with write errors in the Terminal window. It is possible that a .NET Gateway connection terminated without properly disconnecting. In this case, the port used for that connection may be left open.

If you suspect this is the case, to close the port, type the following command at the Terminal prompt:

ObjectScript

```
Close "|TCP|port"
```

Where *port* is the port number to close.