Contents

The most typical, and most robust, deployment of the *EntityServer* is with **Internet Information Services** (IIS).

Hosting the *EntityServer* WCF services in IIS has several benefits:

- IIS provides process activation, health management, and recycling capabilities to increase the reliability of hosted applications.
- Services hosted in IIS use a dynamic compilation model, which simplifies development and deployment.
- Services are deployed and managed like any other type of IIS application.
- When the *EntityServer* is running in ASP.NET Compatibility Mode it can participate fully in the ASP.NET request lifecycle and use ASP.NET Security features.

IIS Setup

The *EntityServer* can be hosted on any version of IIS from 7 forward. You'll generally use the IIS Manager (available from **Control Panel | Administrative Tools** to manage your application.

Setup follows the same steps you would take for any IIS application. The only DevForce-specific requirements are:

- .NET Framework 4.5
- On some older operating systems or versions of IIS you may need to ensure that the WCF HTTP activation component is correctly installed and registered. See troubleshooting for more information.

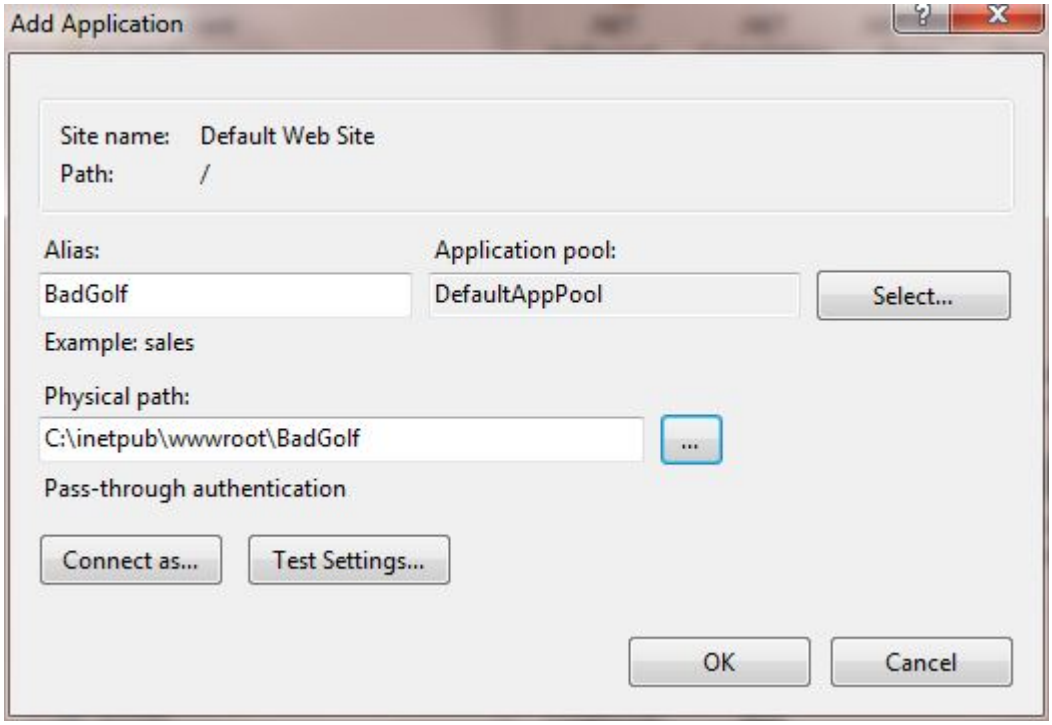
It's usually easiest to use the Visual Studio **Package/Publish** features to create a deployment package and deploy your web application to an IIS server supporting web deployment. We have a walk through with more information and resources.

If you want to do a manual setup, here are the steps (shown in Internet Information Services Manager for IIS 7.5):

1. On the web server, create a physical directory for your application under the web site root folder (typically c:\inetpub \wwwroot).
2. Create bin and log folders under the above directory. Note that the log folder is optional; see below for more information.
3. Create a new IIS application or reuse an existing application.

Be sure to use an application pool which supports ASP.NET 4.5 and the integrated pipeline mode. Note that the "v4.0" framework version will equate to .NET 4.5 if it is installed.

To add a new application, right click the *Default Web Site* and choose **Add Application**. Here we're creating the "BadGolf" application using the *DefaultAppPool*.



1. The **log** folder requires special setup as you'll need to both grant appropriate write permissions to create the DebugLog.xml, but also secure the file and folder from unauthorized access by removing read/browsing permissions. You can disable generation of the *DebugLog* altogether, but the diagnostics it produces are generally helpful, especially in a new deployment.
    1. **Grant** write permissions to the application pool account - In order to allow the *DebugLog* to be created and updated as your application is running, you need to ensure the application account has the appropriate write permissions. The application runs under the identity of the application pool, which will generally be *DefaultAppPool* unless you've created a custom application pool. You'll need to grant this account write permissions to the log folder. In IIS Manager, browse to the folder, right click and select *Edit Permissions* and the *Security* tab, and then edit the permissions for the account.
    2. **Secure** the *DebugLog* file from unauthorized browsing - If the log folder is located under the web application folder the DevForceDebugLog.xml contents will be browsable by anyone who can access your site. To see if your debug log is browsable, just point your internet browser to it - for example http://localhost/badgolf/log/devforcedebuglog.xml. There are several different ways to address this, here are a few options:
        - Change the *logFile* path to a secure folder outside of the IIS application folder, for example: <logging logFile="c:\logs\MyApplication\DevForceDebugLog.xml" />
        - Disable all NTFS permissions to the log folder except as needed by the application pool account.
        - Remove IIS anonymous authentication. In IIS Manager, browse to and select the application's log folder. Click the *Authentication* icon under the IIS section, then select *Anonymous Authentication* and right click to *disable*.

Silveright considerations

If your IIS application will also host the Silverlight application you need to take a few additional steps.

1. Create a ClientBin folder under the application folder. Copy all XAP and ZIP files used by your application into this folder.
2. Copy your *.aspx, *.html and Silverlight.js files to the application folder.
3. Ensure that the XAP mime type is registered in IIS. See here for more information on registering the mime type for different IIS versions.

Additional considerations

- To use SSL to encrypt communications you'll need to obtain a certificate and configure the https binding for your site. Here's more information.
- The *EntityServer* can also use the net.tcp protocol instead of either http or https. You'll need to setup the Windows Process Activation Service to bind a non-HTTP port to your site. Here's more information.
- If you're deploying to an Internet Service Provider you should know that the *EntityServer* requires "full trust". This trust level is not always supported for shared hosting providers.

- If using SQL Server Express databases (such as the aspnetdb database used with ASP.NET security), the advice from Microsoft is: "When deploying applications into production on any version of IIS, SQL Server Express user instances should not be used." The reason for this is that SQL Server Express versions 2005 and 2008 both run under the NETWORK SERVICE account. You will need to change the account identity of the application pool to NETWORK SERVICE in order to use your express database, and this is not recommended. Alternately, you can modify the connection string to use SQL authentication. (See Problems with SQL Server Express user instancing and ASP.net Web Application Projects for more information.)
- See the samples for additional steps needed to use Windows authentication.
- If deploying to production, make sure to turn off any diagnostics, WCF tracing or debugging settings. For example, debug compilation will be on by default until disabled:

```
<compilation debug="false">
```

- To ensure you are targeting .NET 4.5, set the *targetFramework* on the *httpRuntime* element in your web.config. If the targetFramework is not set .NET assumes a default of "4.0" and runs the application in quirks mode, which may cause your application to fail.

```
<system.web>
   <httpRuntime targetFramework="4.5" />
</system.web>
```

Files and assemblies

Now it's time to discuss what to put in the folders you've created for the IIS application.

The following assemblies are always required:

IdeaBlade.Core.dll
IdeaBlade.EntityModel.dll
IdeaBlade.EntityModel.Edm.dll (version 7.2.2 and later: IdeaBlade.EntityModel.Edm.EF5.dll or
IdeaBlade.EntityModel.Edm.EF6.dll)
IdeaBlade.EntityModel.Server.dll
IdeaBlade.EntityModel.Web.dll
IdeaBlade.Linq.dll
IdeaBlade.Validation.dll

You'll also need the .config file:
web.config

Optional but strongly encouraged:
global.asax

Optional (and deprecated):

EntityService.svc
EntityServer.svc

If you're deploying a Code First model, you'll also need the following:

EntityFramework.dll
IdeaBlade.Aop.dll
PostSharp.dll

If you've developed your application using one of the DevForce supplied templates, then the required files and references were added to your web project.

You'll of course also need all of your own "server-side" assemblies. These are the assemblies holding your entity model(s), and any custom extensions you've implemented.

All assemblies should be placed in the bin folder. You may place the DevForce assemblies in the GAC, although there's generally no compelling reason to do so, unless you have a large number of DevForce applications installed and they are all using the same DevForce version.

All other files listed above should be placed in the application root folder.

You do not need to, and in most cases should not, install DevForce to the target machine. Runtime license information is found in your model assembly and not the registry.

Configuration

All configuration information will be in the web.config.

Database connection information

You'll usually need the <*connectionStrings*> for any databases your application uses. (You can also use <*edmKeys*> instead of or in addition to *connectionStrings*, but they may be deprecated in the future.)

Remember the credential information in the connection string: if you include a userid and password you may want to think about encrypting the connection strings section if the server cannot be secured. If you use Windows integrated security for your database connections, the account used to access the database will be the application pool identity, which if you aren't using a custom application pool will be *DefaultAppPool*. In this case, using a custom application pool and identity with only the system and database privileges is a good option. See the security topic for additional information on securing the database.

If you are using a SQL Server Express database (for ASP.NET security for example), consider instead deploying the database to a standard SQL Server instance for improved security.

If you are using a custom *DataSourceKeyResolver* to dynamically provide connection information, you will not need to define connection information in the config file.

Logging

You can use the <logging> element to set the file name and location of the debug log, or to turn logging off altogether. You also might want to archive log files. Generally other logging attributes are not needed, or used only for debugging purposes.

Logging on the server is usually a good idea, since the diagnostics provided will help during testing and in resolving deployment problems. In IIS you must take extra precautions to secure the log file.

A typical config file might contain the following logging information:

```
<logging logFile="log\MyAppDebugLog.xml" archiveLogs="true" />
```

... to create a file named MyAppDebugLog.xml in the log folder and archive files automatically.

Remember in the IIS Setup above we discussed the special access requirements of the log. In IIS, unless you restrict access, anyone can browse to your debuglog file and discover much more about your application than you probably want to share. To see if your debug log is browsable, just point your internet browser to it - for example http://localhost/badgolf/log/myappdebuglog.xml.

One final note: do not put the log file in the bin folder. Doing so will cause your assemblies to be repeatedly recompiled.

Also see the Log and Secure topics for more information.

ObjectServer (custom configuration)

Unlike other *EntityServer* deployments, the service, port and URL information on the <objectServer> element are not used when hosting in IIS. This is because your IIS application configuration has already determined the base address(es) to use.

If you've modified any of the defaults for the <serverSettings> element, such as the *allowAnonymousLogin* or *loginManagerRequired* settings, you'll also need to include that information here. If you're using ASP.NET Security you'll need to include the appropriate configuration information too.

As an example, your <*objectServer*> element might have something like the following when using ASP.NET security and load balancing is required:

```
<objectServer>
  <serverSettings
    allowAnonymousLogin="false"
    loginManagerRequired="true"
    sessionEncryptionKey="mysecretkeyhere"
    useAspNetSecurityServices="true"
    />
</objectServer>
```

A note on **load balancing**: load balancing is only supported with the Data Center Enterprise license. You must use the **same non-blank value** for the *sessionEncryptionKey* on every load-balanced *EntityServer* hosting the application. The key you use is up to you, but remember that it functions like a password. If you're planning an Azure deployment with multiple instances you'll need to set the *sessionEncryptionKey*.

The <*clientSettings*> apply only to the client application.

As with any DevForce server or client, you can use a WCF <*system.serviceModel*> section to configure communications. See the samples for more information.

Other configuration files

Global.asax

You'll usually include a Global.asax in your application folder. The default generated for you with the Visual Studio templates registers a "virtual path provider" in the application start logic. What this does is remove the need for you to provide *.svc files for the various *EntityServer* services. The provider instead creates in-memory files on demand.

You can also use the start logic for other actions:

- Publish trace messages with a call to IdeaBlade.Core.TracePublisher.LocalInstance.MakeRemotable(). See the trace viewer topic for more information.
- Control discovery of your entity model and custom components by setting the IdeaBlade.Core.Composition.CompositionHost.SearchPatterns.

.svc files

You'll notice in the file list above that we called the various *.svc files, such as EntityService.svc and EntityServer.svc optional. These files, when present, control the activation of the WCF services comprising the *EntityServer*. If you've registered the *virtual path provider* in the Global.asax as discussed above, then there's no need to have physical files present, as the provider will create them virtually on demand.

If you don't wish to use the provider or you have customized service activation, then you can still use the .svc files. You will need a single EntityService.svc file, and one file for each EntityServer instance your application will use. If you are using either a data source extension or custom composition context when you create the *EntityManager* in your client application, then it will communicate with a specific EntityServer service having those same characteristics. For example, if you're using a tenant extension of "ABC", you'll need a file named EntityServer_ABC.svc. The file contents will also need to be customized: see the samples for more information.

Troubleshooting

IIS troubleshooting information can be found in the Troubleshooting IIS topic.

Additional Resources

- ASP.NET and IIS Configuration
- IIS Learning Center