Contents

- Overview
- Discovery
  - Silverlight
  - Windows Store
  - Windows Phone
  - Windows 10 Universal
  - Desktop / ASP.NET
- Configuration contents and defaults
  - Root element
  - ObjectServer Element
    - ClientSettings element
    - ServiceKeys element
    - ServerSettings element
  - Logging Element
  - ProbeAssemblyNames Element
  - EdmKeys Element
  - Verifiers Element
- Configuration Editor

DevForce applications are usually configured using an external configuration file - an **app.config** for desktop (and sometimes other) applications, and a **web.config** for ASP.NET applications and the *EntityServer* when hosted by IIS.

With so many .config files in your projects; differences between design-time configuration and run-time configuration; differences between client and server; and further differences between development, test and deployed configurations, it's easy to be confused. Here we'll try to make some sense of these files - what they should contain, and when you should use them.

Overview

Your first question may be "When do I need to use a .config file?" For now, you should assume you'll need a .config file whenever you need to override DevForce's default assumptions. "But what are the DevForce defaults?" you wonder. We'll describe these defaults below in detail, but generally if you need to tell DevForce something about your application: "Where is the database? Is there a remote *EntityServer*? Where? Where is the log file?" then you will likely need to supply this information via a .config file. You can also programmatically supply configuration information, and in some cases supply configuration information via probed interface implementations (e.g., the *IDataSourceKeyResolver* for dynamic database connection information is the most common).

If you create your application using one of the DevForce-supplied Visual Studio project templates, we've included simple .config files where appropriate to get you started. You'll see a web.config in the web projects of your n-tier solutions, and an app.config in the desktop projects of both 2-tier and n-tier solutions; these files will contain the necessary information to get your application running. You'll notice that the Silverlight application does not by default contain an app.config since it's usually not required, but we'll discuss that further below.

As you add projects to your solution - for example a class library to hold your Entity Data Model - you may find these projects also contain an app.config file. In the case of the EDM, the app.config is used only at design time by the EDM designer. The only configuration files used at run-time - the only ones DevForce uses - will be the .config for your application executable and the web.config.

As you develop your application and move from development to test to production environments your configuration will usually need to change too. For example, log file locations, database connection strings, the URL of the *EntityServer* services - all might differ in the different environments. You'll need to be aware of these changing needs and how to set configuration appropriate to each environment.

Discovery

Let's jump in and begin with the issue of discovery. DevForce does have some DevForce-specific features regarding .config discovery, and it's good to understand these features early to avoid confusion later on.

Silverlight

Unlike standard Silverlight applications in which the app.config (or System.Configuration) is not supported, DevForce Silverlight applications do support use of an app.config file. Only the **ideablade.configuration** section can be specified in the file, since standard Microsoft-supplied configuration is not supported. (Note the *ServiceReferences.ClientConfig* file is also supported in DevForce Silverlight applications for customization of WCF communications.)

DevForce will find the file if placed in the application project, named "app.config", and given one of the following build actions:

- **Embedded Resource –** to embed the file in the assembly
- **Content –** to leave the file as a loose file in the XAP, accessible for modification without recompilation

In most Silverlight applications the default configuration assumed by DevForce is sufficient and you will not need to supply an app.config at all.

The default configuration assumes that the *EntityServer* is located in the same location from which the XAP was downloaded. For example, if the XAP for your application was downloaded from http://myhost/myapp/default.aspx, DevForce will default the service URL to http://myhost/myapp/EntityService.svc. If your *EntityServer* is located at this address, then you won't need an app.config in your Silverlight application. If your *EntityServer* is at a different location, then you will need to override the DevForce default; you can do this by including an app.config with the appropriate <objectServer> information. Changing the default with an app.config is optional -- you can also do this programmatically, or via a *ServiceReferences.ClientConfig* file.

A sample Silverlight app.config is provided in the deployment snippets available with the downloaded code.

Windows Store

Similarly to Silverlight applications, DevForce does support an app.config in Windows Store applications even though the environment itself does not support them. Only the **ideablade.configuration** section can be specified in the file, since standard Microsoft-supplied configuration is not supported.

DevForce will find the file if placed in the application project, named "app.config", and given a build action of **Embedded Resource**.

You must use either an app.config or specify *ObjectServer* information programmatically in a Window Store application.

Windows Phone

Similarly to Silverlight applications, DevForce does support an app.config in Windows Phone applications even though the environment itself does not support them. Only the **ideablade.configuration** section can be specified in the file, since standard Microsoft-supplied configuration is not supported. (Note the *ServiceReferences.ClientConfig* file is also supported in DevForce Windows Phone applications for customization of WCF communications.)

DevForce will find the file if placed in the application project, named "app.config", and given one of the following build actions:

- **Embedded Resource –** to embed the file in the assembly
- **Content –** to leave the file as a loose file in the XAP, accessible for modification without recompilation

You must use either an app.config, a ServiceReferences.ClientConfig, or specify *ObjectServer* information programmatically in a Window Phone application.

Windows 10 Universal

Similarly to Silverlight applications, DevForce does support an app.config in UWP applications even though the environment itself does not support them. Only the **ideablade.configuration** section can be specified in the file, since standard Microsoft-supplied configuration is not supported.

DevForce will find the file if placed in the application project, named "app.config", and given one of the following build actions:

- **Embedded Resource –** to embed the file in the assembly
- **Content –** to leave the file as a loose file in the package, accessible for modification without recompilation

You must use either an app.config or specify *ObjectServer* information programmatically in a UWP application.

Desktop / ASP.NET

In desktop applications, DevForce follows a slightly different “probing” path to find the app.config than standard .NET applications. Standard .NET applications use only the application’s config file – MyApp.exe.config, or web.config for a web application. In DevForce, the search for configuration information is as follows, and continues until a valid configuration is found:

1. If the ConfigFileLocation property of the IdeaBladeConfig object has been set in the executing code, DevForce will search the indicated location for a file named or matching “*.exe.config” (or “web.config” if a web project). If not found, other *.config files in the folder are searched for a valid ideablade.configuration section.
   IdeaBladeConfig.ConfigFileLocation = @"c:\myapp";

2. If the IdeaBladeConfig.ConfigFileAssembly property has been set, DevForce will look for an embedded resource named “app.config” in the specified assembly.
   IdeaBladeConfig.ConfigFileAssembly = Assembly.GetExecutingAssembly();
3. Next, the current executable/bin folder is searched for a file named or matching “*.exe.config” (or “web.config” if a web project). If not found, other *.config files in the folder are searched for a valid ideablade.configuration section.
4. DevForce next searches for an embedded resource named “app.config” in the entry assembly.
5. If a valid Ideablade.configuration section was not found in any of the above locations then DevForce will create a default IdeaBladeConfig instance. In some applications this default instance may be sufficient, but check your debuglog.xml if you find that your configuration is not being used.

Caution: If you rely on DevForce-specific discovery - using an embedded resource or a loose config file not identified by .NET as the config file for the AppDomain - non-IdeaBlade sections of the config file will not be found by .NET.

Note that in a test project, such as one created with MSTest, if you have enabled deployment you should also ensure that a loose config file is deployed, or set either the IdeaBlade.ConfigFileLocation or IdeaBlade.ConfigFileAssembly properties, since standard DevForce probing may not work as expected.

Configuration contents and defaults

Now that we know how to find a .config file, what should we put in it? DevForce defines the *ideablade.configuration* section to hold all IdeaBlade-specific configuration information. At run-time, this information is used to load the single in-memory instance of the *IdeabladeConfig* class, available via the *IdeaBladeConfig.Instance* static property.

You may modify many properties of the *IdeaBladeConfig* at run-time, although this should be done before your application begins using other DevForce features. Programmatic run-time configuration can be accomplished by modifying *IdeaBladeConfig.Instance*.

The ideablade.configuration section includes the following child elements.

Element	Description
(Root)	
objectServer	Governs access to the *EntityServer* service in an n-tier deployment. Contains *clientSettings* and *serverSettings* child elements to specify configuration specific to either client or server.
logging	Identify where and how to write the DebugLog.
probeAssemblyNames	Deprecated. Allows the developer to specify assemblies to be used in the discovery of custom implementations of DevForce interfaces. Any assembly names specified here supplement the default DevForce discovery options.
edmKeys	Optional configuration data applicable to one or more Entity Data Model data sources. There may be several named *<edmKey/>* tags if the application uses more than one Entity Data Model. DevForce will use information in the *<connectionStrings>* element to discover data source information, but you may specify an *EdmKey* to override that discovery.
verifiers	Used to define verifiers external to your application code. See the Validation topic for more information.

Root element

In order to include the *ideablade.configuration* section in your config file you must "register" the section. You do this by placing a section element in the *configSections*, like so:

```
<configuration>
  <configSections>
    <section name="ideablade.configuration" type="IdeaBlade.Core.Configuration.IdeaBladeSection, IdeaBlade.Core" />
  </configSections>
</configuration>
```

The *configSections* should be at the top of the configuration, so that the section registrations take place before the actual section.

The *ideablade.configuration* section looks like the following. It can go anywhere in the config file after the *configSections* definitions. Providing the namespace ensures that you can use Intellisense while editing the config in Visual Studio.

```
<ideablade.configuration version="6.00" xmlns="http://schemas.ideablade.com/2010/IdeaBladeConfig">
</ideablade.configuration>
```

ObjectServer Element

Settings controlling data service features, security, and communications to the *EntityServer*.

```
<objectServer remoteBaseURL="http://localhost"
              serverPort="9009"
              serviceName="EntityService"
              >
 <clientSettings isDistributed="false" />
 <serviceKeys>
   <serviceKey name="BOS2" remoteBaseURL="http://somehost" serverPort="80" serviceName="myapp/EntityService.svc"  />
 </serviceKeys>
 <serverSettings allowAnonymousLogin="true"
                 loginManagerRequired="false"
                 sessionEncryptionKey=""
                 supportedClientApplicationType="UseLicense"
                 useAspNetSecurityServices="false"
                 userSessionTimeout="30"
                  />
</objectServer>
```

Attribute	Description
remoteBaseURL	The protocol and machine name (or IP address) used to form the full URL.
serverPort	The port the *EntityServer* is listening on.
serviceName	The name of the entry point service. This is generally "EntityService" when not hosted by IIS; when hosted in IIS the name consists of both the ASP.NET application name and the service file "EntityService.svc", for example, "myapp/EntityService.svc".

The *remoteBaseURL*, *serverPort* and *serviceName* attributes are used to determine the "endpoint" address of the server when running in an n-tier configuration. These settings are common to both client and server. When running in IIS, or if using a *system.serviceModel* section to configure WCF services, you do not need to provide these fields. A full URL, when built from these fields, might look something like "http://localhost:9009/EntityService" or "http://localhost/MyApp/EntityService.svc". (Of course "localhost" is used only during development when both client and server are on the same machine.)

ClientSettings element

The *clientSettings* apply only to the client. Child element of the *objectServer* element.

Attribute	Description
isDistributed	Determines whether the client will use a remote *EntityServer*. When enabled, the client will communicate with an application server tier; when disabled, the client performs its own data service operations.

ServiceKeys element

The *serviceKeys* apply to the client, but may be used on the server in some situations as defined below. A *serviceKey* provides the address information for a single application server; a client application may communicate with multiple application servers. Child element of the *objectServer* element.

ServiceKey attributes

Attribute	Description
name	Identifies the service key.
remoteBaseURL	The protocol and machine name (or IP address) used to form the full URL.
serverPort	The port the *EntityServer* is listening on.
serviceName	The name of the entry point service. This is generally "EntityService" when not hosted by IIS; when hosted in IIS the name consists of both the ASP.NET application name and the service file "EntityService.svc", for example, "myapp/EntityService.svc".

The *remoteBaseURL*, *serverPort* and *serviceName* attributes are used to determine the "endpoint" address of a server when running in an n-tier configuration. These attributes serve the same purpose on the *serviceKey* as they do when defined on the <*objectServer*> element itself. You use one or more <*serviceKey*> definitions when your client application may communicate with multiple application servers.

The <*serviceKeys*> are not usually defined on the server, since their intention is to provide for multiple named servers, but they can be used there. If the *EntityServer* is deployed as either a console application or Windows service then DevForce must determine the "base address" of the *EntityServer*. It will use the attributes on the <*objectServer*> if present, otherwise it will look for a key named "default", and if not present use the first *serviceKey* defined.

ServerSettings element

These settings concern server configuration. In a 2-tier application without an application server tier you would include any *serverSettings* needed in the application config file. The descriptions below indicate if a setting is specific to n-tier. Child element of the *objectServer* element.

Attribute	Description
allowAnonymousLogin	Determines whether "guest" users are allowed by the application. These users can use the application without supplying login credentials. Default is true.
loginManagerRequired	Determines whether the application will function if a "login manager" cannot be found. When not using ASP.NET security features you must provide an IEntityLoginManager implementation if this flag is true. Default is false.
sessionEncryptionKey	In an n-tier application with load balancing, you should set this key to the same non-empty value on all servers in the cluster. The key is used to encrypt information in the token passed between client and server and allows the session to be transferred to other servers when a machine in the cluster fails. Data Center license only.
supportedClientApplicationType	This is used in an n-tier application to allow the application server tier to correctly initialize WCF communications. By default DevForce will use your license to determine which application types to support. You may override this setting, as long as the chosen value is consistent with your license. For example, you might have a Universal license but a given server might support only Silverlight clients, or only WinClient clients. By default, if you have a Universal license communications for Silverlight, Windows Store and WinClient types of clients will be initialized.
useAspNetSecurityServices	Determines whether ASP.NET security is used for user authentication. By default this flag is off.
userSessionTimeout	Sets the number of minutes of inactivity after which a client session is removed from the server's session map. Default is 30 minutes.

Logging Element

Controls the logging of run-time debug and trace information applicable to any configuration.

```
<logging
   logFile="DevForceDebugLog.xml"
   archiveLogs="false"
   shouldLogSqlQueries="false"
   port="9922"
   serviceName="TracePublisher"
   usesSeparateAppDomain="false" />
```

Attribute	Description
logFile	The name of the file containing tracing and debugging information. You can set the field to an empty string to turn off default logging. You can also supply a relative or full path name. By default the log file is written to the exe folder in Desktop applications and to a subfolder in ASP.NET applications. Silverlight and Windows Store applications do not create a physical log file.
archiveLogs	Defaults to false. Turn this on to archive previous log files.
shouldLogSqlQueries	Whether to log the generated SQL for queries. This is a global setting which applies to all data sources. If an EdmKey *logTraceString* attribute is false (the default), logging will still occur when *shouldLogSqlQueries* is true.
port	This is the port used by the TracePublisher service when "remote" publishing is enabled. By default port 9922 is used, but if you have multiple publishers on the same machine you need to provide either a unique port or serviceName for each.

serviceName	The name of the TracePublisher service when "remote" publishing is enabled. By default this name is "TracePublisher".
usesSeparateAppDomain	This is an advanced feature which allows the logger to be run in a separate application domain. By default this value is false and logging takes place within the same application domain as the publisher.

ProbeAssemblyNames Element

By default DevForce uses MEF (the Managed Extensibility Framework) to discover exported implementations of custom features. In Silverlight applications, discovery uses the assemblies in the main XAP. In all other applications, discovery uses the files in the exe/bin folder. If the default discovery is insufficient you can specify additional assembly names here to be included when "probing".

Unknown macro: IBNote

```
The "IBNote" macro is not in the list of registered macros. Verify the spelling or contact your administrator.
```

```
<probeAssemblyNames>
    <probeAssemblyName name="MyAssembly" />
</probeAssemblyNames>
```

Sub-element	Description
probeAssemblyName	Name of an assembly to be probed. In Silverlight, this should be a fully-qualified assembly name.

If you do find you need or want to specify the probe assembly names here, there are some naming rules to be aware of:

In Desktop and ASP.NET applications, you can use the assembly *simple* name; i.e., the assembly file name less the ".DLL" or ".EXE" extension. For example, for the *DomainModel.dll* assembly, use "DomainModel".

In Silverlight applications, the assembly display name should be used. The display name includes the assembly simple name (defined just above), version, culture and public key token. For example, the assembly display name for the *DomainModel.dll* assembly might look like the following:

DomainModel, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

The public key will be non-null if you have signed the assembly. The version number is the assembly version defined in your code.

EdmKeys Element

An *EdmKey* is used to uniquely identify a datasource in DevForce. By default DevForce will build in-memory *EdmKeys* based on information in the <*connectionStrings*> element in the **server's** config file. You can specify one or more *EdmKeys* here to override this behavior. If the same key name is defined in both the *EdmKeys* and *connectionStrings*, the *EdmKey* specification will be used.

```
<edmKeys>
  <edmKey
     name="Default"
     connection="metadata= ....."
     tag="some info here"
     logTraceString="false"
      />
</edmKeys>
```

EdmKey attributes

Attribute	Description
name	Identifies the data source. If you are using data source extensions the name should contain the extension. For example a key named "Default_ext1" would be used for the "ext1" data source extension with the "Default" data source.
connection	The complete EF connection string. This is the same as the *connectionString* attribute in the <*connectionStrings*>.
tag	Can be used to provide additional information.
logTraceString	Can be used to enable logging of generated SQL queries. This defaults to false, but can be turned on when debugging is needed.

The *shouldLogSqlQueries* flag on the logging element overrides the setting on the *EdmKey*.

Verifiers Element

Verifiers defined in the .config file are discovered by calling *VerifierEngine.DiscoverVerifiersFromConfig* .

```
<verifiers>
  <verifier
    name="SampleDateTimeRangeVerifier"
    verifierType="IdeaBlade.Validation.DateTimeRangeVerifier, IdeaBlade.Validation"
    applicableType="DomainModel.Employee, DomainModel"
    executionModes="InstanceAndOnBeforeSetTriggers"
    errorContinuationMode="Stop">
    <verifierArgs>
      <verifierArg name="propertyName" value="BirthDate" />
      <verifierArg name="minValue" value="1/1/1965" />
    </verifierArgs>
  </verifier>
</verifiers>
```

Verifier attributes

Attribute	Description
name	Name of the verifier.
description	Optional description used in status messages.
verifierType	The assembly-qualified name of the verifier
applicableType	The assembly-qualified name of the object type on which the verifier is defined.
executionModes	The conditions under which the verifier will execute.
errorContinuationMode	Determines the action to take after a verification error.
sortValue	The order in which the verifier will be executed within a verifier batch.
tag	Can be used to provide additional information.
verifierArgs	Sub-element defining any arguments for the verifier.

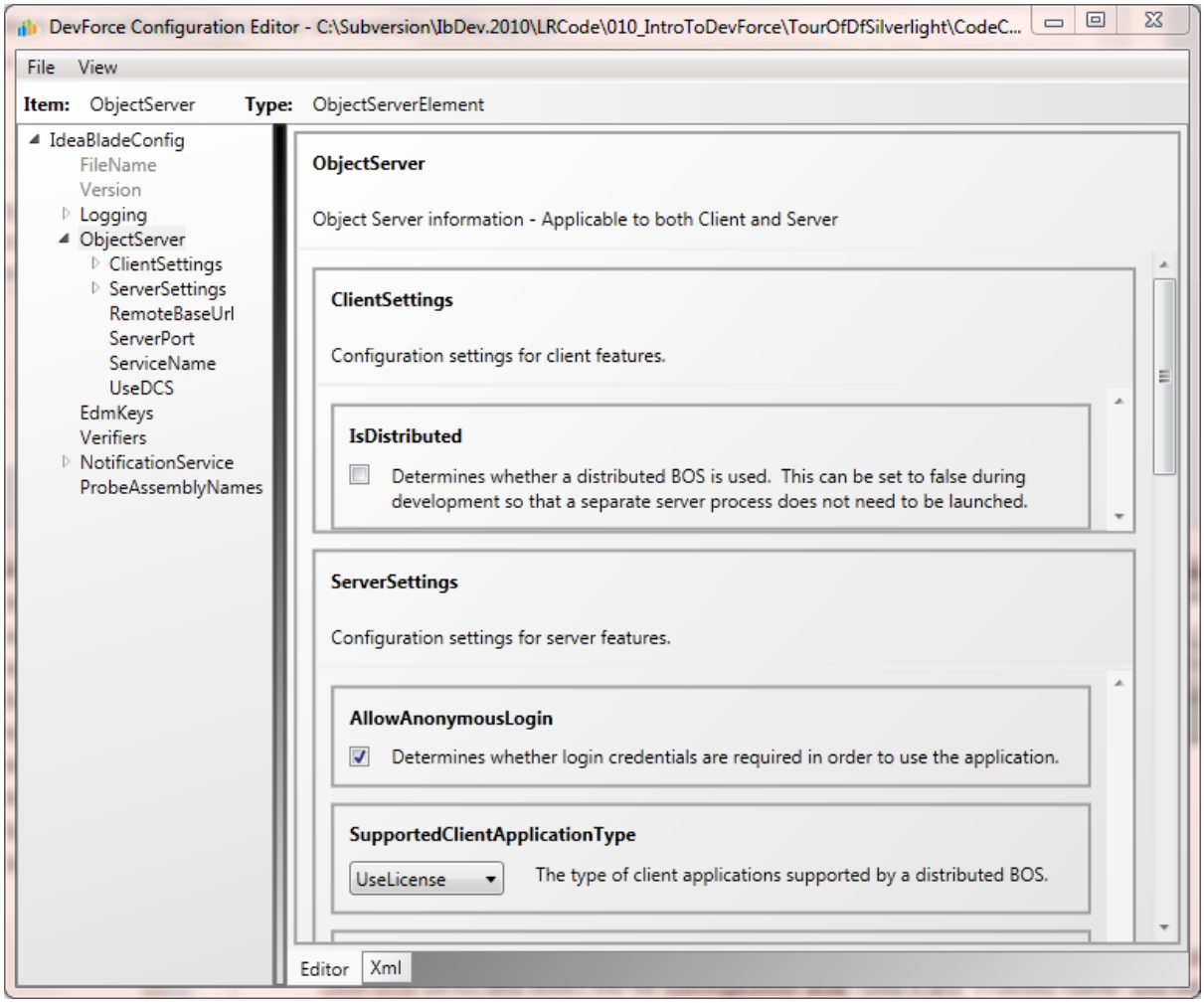
VerifierArg attributes

The verifier arguments are specific to the type of verifier.

Attribute	Description
name	Name of the argument.
value	Value of the argument.

Configuration Editor

So how to edit your app.config or web.config? You can edit directly in Visual Studio, and Intellisense will help guide you in the elements and attributes available, but if you'd like a more structured editor we've also provided the DevForce Configuration Editor to help in editing the *ideablade.configuration* section. The editor provides descriptive help, and can eliminate many simple errors such as misspellings. You can use the editor to modify an existing config file or create a new one.



You can find ConfigEditor.exe in the Tools sub-folder of the DevForce installation.

You can use the *Configuration Editor* in two different ways:

1. Launch it directly.
2. Configure it to work within Visual Studio. To do this:
    1. Select the app.config file, right-click, and select **Open With…** option.
    2. If you do not see **ConfigEditor.exe** in the list of programs, click the <Add…> button.
    3. On the Add Program dialog, click the ellipsis button to browse to a file. Navigate to the DevForce installation directory (typically C:\Program Files\DevForce 2012) and select the file **ConfigEditor.exe** from the Tools subfolder. Give it any “Friendly Name” you wish; e.g., “DevForce Configuration Editor”.
    4. Once **DevForce Configuration Editor** is in the list, double-click it to open the configuration file in that editor.