Contents

You can **provide DevForce configuration information programmatically** by setting properties of the ***IdeaBladeConfig.Instance*** singleton, either on the client or on the server.

Many DevForce components require configuration. A remote client, for example, must be configured with the address of the server. The DevForce *EntityServer* needs to know the connection string of the database.

The "Configure and Deploy" topic describes many ways to provide configuration information such as adding Ideablade sections to *Web.config* and *App.config* files and creating custom DataSourceKeyResolvers.

This topic describes the *IdeaBladeConfig* in-memory representation of DevForce configuration.

IdeaBladeConfig.Instance

At runtime, DevForce creates an in-memory representation of configuration information in a static "singleton" object called the *IdeaBladeConfig.Instance* . DevForce components reference this in-memory singleton when they need configuration data such as

- *ObjectServer.ClientSettings.IsDistributed*: Whether the client is running 2-tier (false) or n-tier (true)
- The URL to the remote server
- The Entity Data Model (EDM) keys (*EDM Keys*) that hold the database connection strings
- ... much more ...

You can inspect configuration values in your code via the static *IdeaBladeConfig.Instance* property. You can update them as we will see.

Both the client application and the server have their own instances of *IdeaBladeConfig*. They may share some configuration in common such as the assembly names to probe for custom classes. Some configuration is meaningful only on the client such as the URL of the server. Some configuration is meaningful only on the server such as database connection strings. We use one object type, the *IdeaBladeConfig*, for both environments.

Client and server functionality execute in the same process in a 2-tier deployment and their respective configuration data are held in a single *IdeaBladeConfig* instance.

Where IdeaBladeConfig gets its values

DevForce populates the *IdeaBladeConfig* from values in the XML configuration file that it discovers.

It doesn't actually try to populate an *IdeaBladeConfig* until you or a DevForce component asks for configuration information.

This just-in-time behavior gives you the ability to specify where to get external configuration data by setting certain static properties on the *IdeaBladeConfig* class:

Property	Description
ConfigFileLocation	Directory to search for configuration files
ConfigFileName	Local path and filename of the configuration file to use
ConfigFileAssembly	Assembly to search for an embedded

These setting come into play the moment you retrieve the *IdeaBladeConfig.Instance*. DevForce discovers the configuration information (if any) and populates the in-memory *IdeaBladeConfig* object.

You will notice in the examples below that many of the *IdeaBladeConfig* object values are not set at all. That's because DevForce didn't find any pertinent IdeaBlade XML sections in any of the configuration files it searched.

That's quite normal. When a component requests a configuration value and there is none, DevForce resorts to default behaviors to discover and construct the unspecified values. We'll see an example of that below with regard to determining a database connection string.

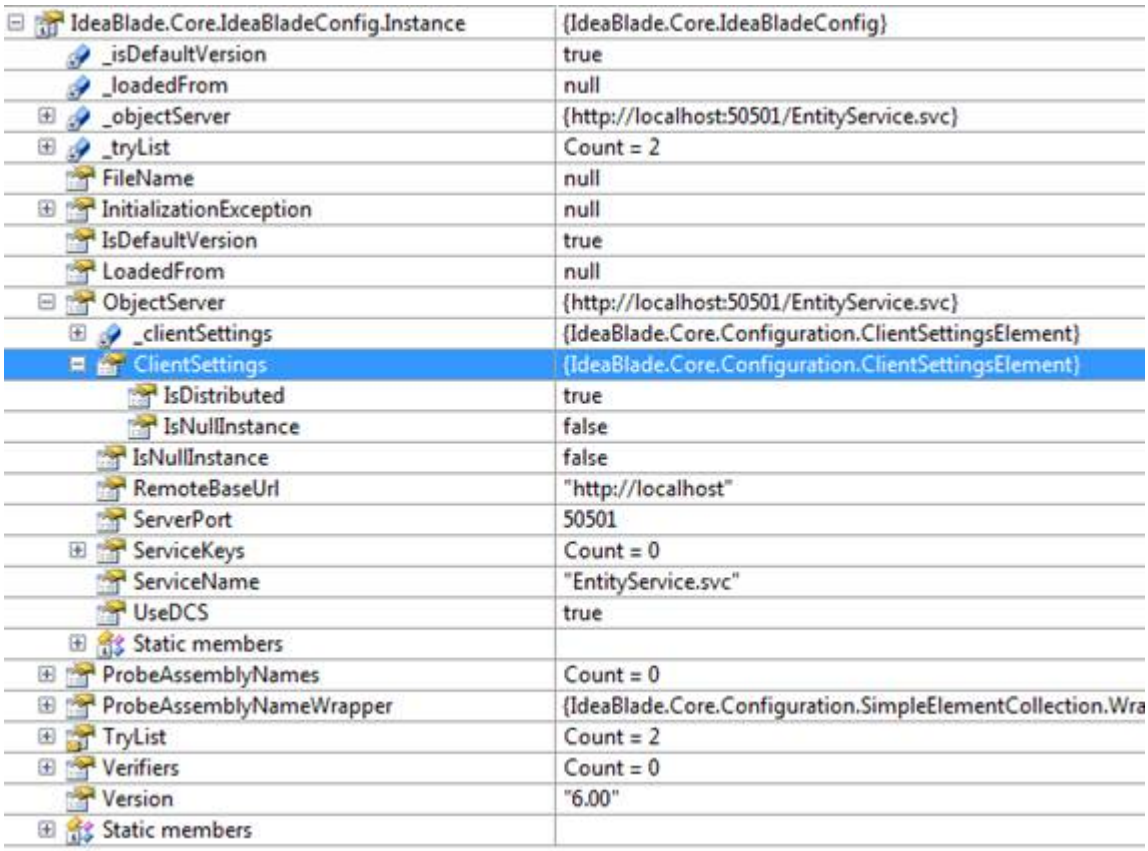
DevForce prefers configuration values to its own calculated defaults. You can set configuration values as well as read them. If you set configuration values in the *IdeaBladeConfig* object *before* a component requests them, your custom configuration values will prevail.

The timing is critical. DevForce components only read IdeaBladeConfig once at the moment they need a piece of configuration. They stash config away in a place of their convenience (typically a static field) and don't look back. You must update *IdeaBladeConfig* before a DevForce component reads it; otherwise, the horse has left the barn and your updates are without effect.

If you intend to change the IdeaBladeConfig programmatically, do so very early in the client application bootstrapping. The *Application_Startup* method of your *Application* class is a good candidate in a client app. The *Global.asax* is a good place for this logic on the server.

Setting the EntityServer address dynamically

Here's a debugger screenshot of an IdeaBladeConfig in a Silverlight client.



DevForce Silverlight applications, like Windows Store applications, are necessarily n-tier, and thus always require the *EntityServer* URL. You can programmatically set the server URL information, which has 3 parts as you see below:

Property	Example	Description
RemoteBaseUrl	*http://localhost*	Base URL of the server
ServerPort	*50501*	The port, typically 80 (HTTP) or 443 (HTTPS) in production (prefer HTTPS!)
ServiceName	*EntityService.svc*	The application name, followed by the service name. The service name of 'EntityService.svc' may not be changed. Prepend the ASP.NET application name.

With the sample values above, DevForce combines the parts to form this: *http://localhost:5051/EntityService.svc*.

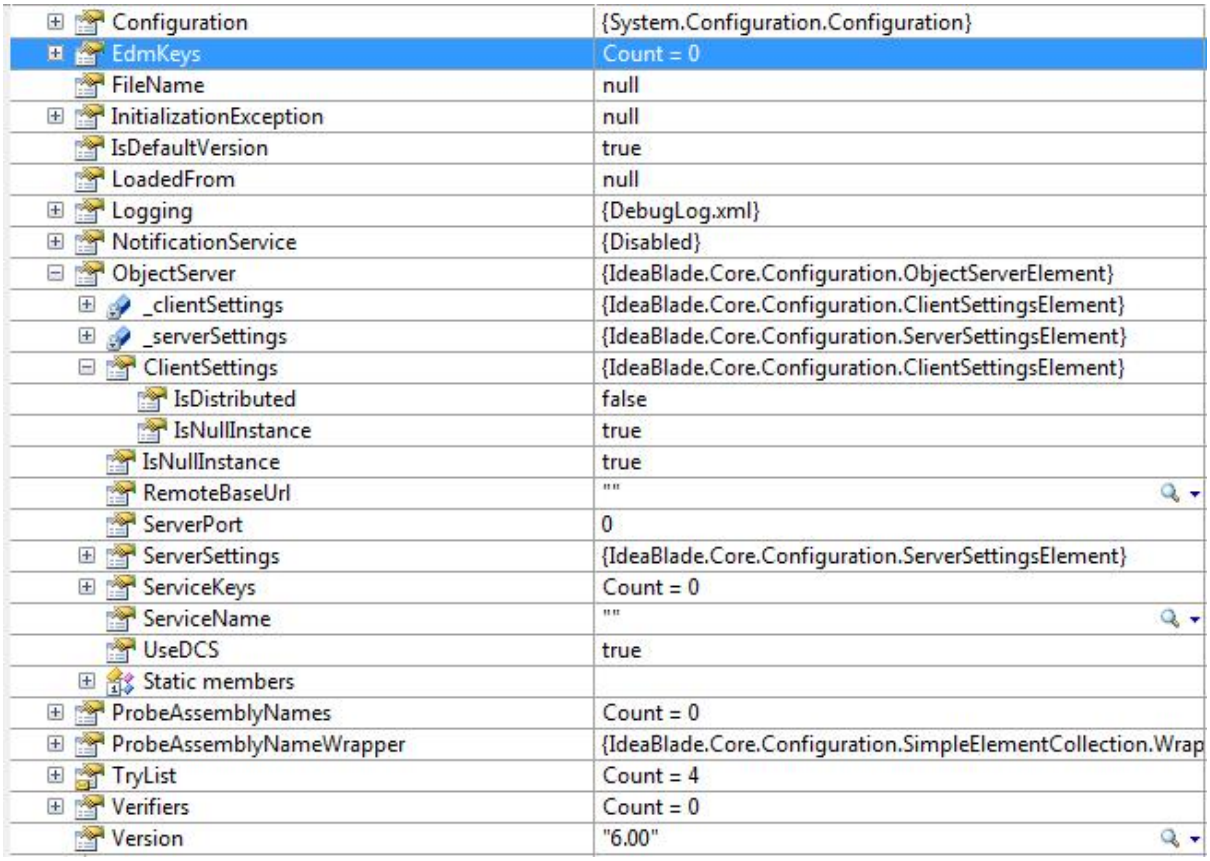
Here's how you'd set the URL values in code:

```
// URL is http://someserver:80/SomeApp/EntityService.svc
var server = IdeaBladeConfig.Instance.ObjectServer;
```

```
server.RemoteBaseUrl = "http://someserver";
server.ServerPort = 80;
server.ServiceName = "SomeApp/EntityService.svc";
```

Two-tier client IdeaBladeConfig example

Here's a debugger screenshot of an IdeaBladeConfig in a 2-tier application.



In a 2-tier application, the *ObjectServer.ClientSettings.**IsDistributed*** flag is false and remote URL information is irrelevant (the URL information is null in this example).

Specifying database connections with configuration

Database connection string information is an example of a server side concern. If we want to specify a connection string in code we can do that in one of the *IdeaBladeConfig* server-oriented configuration settings.

The example is taken from a 2-tier *IdeaBladeConfig*. Because we're focusing on server-side configuration, the following discussion applies as well to a server-side *IdeaBladeConfig* in an n-tier application.

Before explaining how, it's worth noting that database connection strings aren't usually specified in code. They are typically acquired from XML configuration files.

In the screenshot, the count of the *EdmKeys* property is zero. Database connection information is held in *EDM Keys*. An *EDM Key* represents connection information about an Entity Data Model data source (e.g., the database).

There are no *EDM Key* specifications in the *IdeaBladeConfig*. Yet the application works; it retrieves data from the database. Evidently the connection string is coming from somewhere else. It works because DevForce looks for the connection information in a .NET configuration file if you don't specify an *EDM Key* explicitly. Here's the procedure.

When the application queries for an entity, say a *Customer* entity, DevForce looks up that entity type's *DataSourceName*. That name is inscribed in the entity class itself as seen in this extract from the generated class file:

```
[IbEm.DataSourceKeyName(@"NorthwindEntities")]
[IbEm.DefaultEntitySetName(@"NorthwindEntities.Customers")]
public partial class Customer : IbEm.Entity { ...
```

```
<IbEm.DataSourceKeyName("NorthwindEntities")> _
```

```
<IbEm.DefaultEntitySetName("NorthwindEntities.Customers")> _
Partial Public Class Customer Inherits IbEm.Entity ...
```

Notice that the data source key name is “NorthwindEntities”.

DevForce sees that you did not specify an *EDM Key* named “NorthwindEntities”. Therefore it looks for a correspondingly-named *<connectionStrings/>* element inside the appropriate .NET configuration file: *Web.config* if the application server runs in IIS; the *App.config* file in the application assembly if the application is running 2-tier (as in this example). In either place, the *<connectionStrings/>* element looks something like this:

```
<connectionStrings>
  <add name="NorthwindEntities" connectionString="metadata=res://*/Northwind.csdl|res://*/Northwind.ssdl|res://
*/Northwind.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=localhost;Initial
Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True&quot;" providerName="System.Data.EntityClient" />
</connectionStrings>
```

The unusual looking *connectionString* value is an Entity Framework connection string. The first sections identify the 3 file parts of the EDM; the last section is the database connection string.

<connectionStrings/> is a .NET configuration tag which is why it doesn't appear in the *IdeaBladeConfig* object; *IdeaBladeConfig* only holds DevForce-specific configuration data.

DevForce now has all it needs to dynamically create an *EDM Key* with the “NorthwindEntities” name. You didn’t have to specify it explicitly in your configuration.

You COULD have done so … in which case your explicit definition would take precedence. You can specify the *EDM Key* in the XML configuration file if you wish. But specifying it in code ... as we do next ... is of greater interest in this *IdeaBladeConfig* topic.

Specifying database connections in code

If you're not using a configuration file, perhaps you're getting the connection information from some other external source. Some authors of multi-tenant applications, for example maintain a central, shared database table with different connection strings for each tenant. They can add and remove tenants without modifying configuration files and construct the corresponding *EDM Keys* on the fly. Most developers would construct such dynamic keys by means of a custom DataSourceKeyResolver.

Perhaps you have only one or a few connection string and you truly want to bake those strings directly into your code. There are two simple ways to do it that don't involve a *DataSourceKeyResolver*.

The first and preferable way is to add an *EDM Key* to the *IdeaBladeConfig* object. Here is an example that has the same effect as what DevForce did by default, assuming the model and connections shown above:

```
var config = IdeaBlade.Core.IdeaBladeConfig.Instance;
var keys = config.EdmKeys;
keys.Add(
 new EdmKeyElement
  {
   Name          = "NorthwindEntities",
   LogTraceString = false, // whether to log EF generated SQL
  // Same connection string as above, tweaked for C#
   Connection =
@"metadata=
res://DomainModel.Desktop/Northwind.csdl|
res://DomainModel.Desktop/Northwind.ssdl|
res://DomainModel.Desktop/Northwind.msl;
provider=System.Data.SqlClient;
provider connection string=
 'Data Source=localhost;Initial Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True'"
  }
);
```

```
Dim config = IdeaBlade.Core.IdeaBladeConfig.Instance
Dim keys = config.EdmKeys
keys.Add(New EdmKeyElement With {.Name = "NorthwindEntities", _
.LogTraceString = False, _
.Connection = "metadata=" & ControlChars.CrLf & "res://DomainModel.Desktop/Northwind.csdl|" _
& ControlChars.CrLf & "res://DomainModel.Desktop/Northwind.ssdl|" & ControlChars.CrLf & _
"res://DomainModel.Desktop/Northwind.msl;" & ControlChars.CrLf & _
"provider=System.Data.SqlClient;" & ControlChars.CrLf & "provider connection string=" _
& ControlChars.CrLf & "  'Data Source=localhost;Initial _
Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True'"})
```

A second way to add the connection string is to add it to the parent configuration's collection of connection strings as in this example:

```
var config = IdeaBlade.Core.IdeaBladeConfig.Instance;
// Climb up the configuration tree to get to .NET ConnectionStrings
var connections = config.Configuration.ConnectionStrings.ConnectionStrings;
connections .Add(
 new ConnectionStringSettings   {
     Name          = "NorthwindEntities",
    // Same connection string as above, tweaked for C#
    Connection =
@"metadata=
res://DomainModel.Desktop/Northwind.csdl|
res://DomainModel.Desktop/Northwind.ssdl|
res://DomainModel.Desktop/Northwind.msl;
provider=System.Data.SqlClient;
provider connection string=
  'Data Source=localhost;Initial Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True'"
    }
);
```

```
Dim config = IdeaBlade.Core.IdeaBladeConfig.Instance
' Climb up the configuration tree to get to .NET ConnectionStrings
Dim connections = config.Configuration.ConnectionStrings.ConnectionStrings
' Same connection string as above, tweaked for C#
connections.Add(New ConnectionStringSettings With {.Name = "NorthwindEntities", _
  .Connection = "metadata=" & ControlChars.CrLf & "res://DomainModel.Desktop/Northwind.csdl|" _
  & ControlChars.CrLf & "res://DomainModel.Desktop/Northwind.ssdl|" & ControlChars.CrLf _
  & "res://DomainModel.Desktop/Northwind.msl;" & ControlChars.CrLf & _
 "provider=System.Data.SqlClient;" & ControlChars.CrLf & "provider connection string=" _
  & ControlChars.CrLf & "  'Data Source=localhost;Initial _
  Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True'"})
```

This approach is generally less useful as you cannot set the *LogTraceString* which tells DevForce to log the generated SQL.

Adapting Entity Framework connection strings

The programmatic string is not an exact duplicate of the one that Entity Framework put in the model project's *App.config* (or in the *Web.config* if your model co-habitates with the web application project). You"ll have to make some adjustments:

- make sure to replace the two """ tokens either with two escaped double quotes (\") or with single quotes (')
- if your model is in a separate project, replace the (*) in *res://*/* with the model project assembly's name as in *res://**DomainModel.Desktop**/*; if the model project is strongly-named, you'll have to use the strong name of the assembly.