Contents

You can take control of the configuration of WCF communications in your n-tier client application by directly working with the WCF <system.serviceModel> configuration section.

Problem

The <*system.serviceModel*> section is defined in your application's config file. In Silverlight application's this will be the ServiceReferences.ClientConfig while in all other application types it will be the *.exe.config, or web.config for an ASP.NET application.

When you define the <*system.serviceModel*> in your configuration file, you're telling DevForce that you're taking control over the configuration of the WCF endpoints over which communication to the *EntityServer* will occur.

The configuration is usually the same in all environments, although the WCF Silverlight implementation is not as robust as in other .NET environments, so some configuration options are not available.

You can use the WCF Service Configuration Editor available from the Visual Studio **Tools** menu to edit the <*system.seviceModel*> information.

Solution

We can't possibly explain all the nearly endless complexities of WCF configuration, but here we'll show you a few samples.

Unknown macro: IBNoteThe "IBNote" macro is not in the list of registered macros. Verify the spelling or contact your administrator.

Silverlight

Note that the ServiceReferences.ClientConfig does not support binding extensions, which is how DevForce supports compression via the GZip binding element. In Silverlight you must programmatically add the GZip element via the *ServiceProxyEvents* if you use a custom ClientConfig. See the adding GZip sample for information.

Here's the configuration in Silverlight. Endpoints are defined for "EntityService" and "EntityServer", and a *CustomBinding* defines binary formatted messages over http.

```
<configuration>
 <system.serviceModel>
  <client>
   <endpoint name="EntityService"
          address="http://localhost:9009/EntityService.svc/sl"
          binding="customBinding" bindingConfiguration="customBinaryBinding"
          contract="IdeaBlade.EntityModel.IEntityServiceContract"
          />
   <endpoint name="EntityServer"
          address="http://localhost:9009/EntityServer.svc/sl"
          binding="customBinding" bindingConfiguration="customBinaryBinding"
          contract="IdeaBlade.EntityModel.IEntityServerContract"
          />
  </client>
  <bindings>
   <customBinding>
    <binding name="customBinaryBinding">
     <binaryMessageEncoding/>
     <httpTransport maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
    </binding>
   </customBinding>
```

```
    </bindings>
  </system.serviceModel>
</configuration>
```

Other clients

Here's the default configuration for other environments. Note that we were able to add GZip support to the *CustomBinding* and set *ReaderQuotas*. The address here assumes the *EntityServer* is hosted by a web application, as is standard in the DevForce n-tier templates.

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="EntityService"
                address="http://localhost:9009/EntityService.svc"
                binding="customBinding" bindingConfiguration="compressedBinaryBinding"
                contract="IdeaBlade.EntityModel.IEntityServiceContract"
                />
      <endpoint name="EntityServer"
                address="http://localhost:9009/EntityServer.svc"
                binding="customBinding" bindingConfiguration="compressedBinaryBinding"
                contract="IdeaBlade.EntityModel.IEntityServerContract"
                />
    </client>
    <bindings>
      <customBinding>
        <binding name="compressedBinaryBinding">
          <gzipMessageEncoding>
            <readerQuotas
                maxDepth="2147483647"
                maxStringContentLength="2147483647"
                maxArrayLength="2147483647" />
          </gzipMessageEncoding>
          <httpTransport maxReceivedMessageSize="2147483647" />
        </binding>
      </customBinding>
    </bindings>
    <extensions>
      <bindingElementExtensions>
        <add name="gzipMessageEncoding"
             type="IdeaBlade.Core.Wcf.Extensions.GZipMessageEncodingElement, IdeaBlade.Core"/>
      </bindingElementExtensions>
    </extensions>
  </system.serviceModel>
</configuration>
```

Example

Change timeout values

Changing communication timeouts is often needed. The *SendTimeout* on the client determines how long a client request will wait for a server response before timing out. This applies whether the request was synchronous or asynchronous. Although other WCF timeouts are available and can be set, generally only the *SendTimeout*, which defaults to 1 minute, is of importance.

Since only the binding definition is affected, we show only the binding below, in this case for Silverlight, but the attribute is the same for other application types too. Here we've set the *SendTimeout* to 2 minutes.

```
    <bindings>
      <customBinding>
        <binding name="customBinaryBinding" sendTimeout="00:02:00">
          <binaryMessageEncoding />
          <httpTransport maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
        </binding>
      </customBinding>
    </bindings>
```

Use Windows security in Silverlight

WCF in Silverlight does not support passing Windows credentials with the binding. The credentials will, however, be passed by the browser if the web server has enabled Windows authentication and the service's endpoint binding supports Windows credentials. The browser may prompt for credentials. See the resources below for more information. See the topic on customizing server configuration for how to configure the services.

Use HTTPS in Silverlight

Here's a sample showing how to use transport security in Silverlight. Your server should have been configured for SSL.

Unknown macro: IBNote

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

If your XAP was downloaded from an HTTP address you'll alos need a cross domain policy in place on the server, since Silverlight considers non-secure to secure service access to be "cross domain". See here for more information.

```
<system.serviceModel>
  <client>
    <endpoint name="EntityService"
              address="https://myserver/myapp/EntityService.svc/sl"
              binding="basicHttpBinding" bindingConfiguration="secureBinding"
              contract="IdeaBlade.EntityModel.IEntityServiceContract"
              />
    <endpoint name="EntityServer"
              address="https://myserver/myapp/EntityServer.svc/sl"
              binding="basicHttpBinding" bindingConfiguration="secureBinding"
              contract="IdeaBlade.EntityModel.IEntityServerContract"
              />
  </client>
  <bindings>
    <basicHttpBinding>
      <binding name="secureBinding" maxReceivedMessageSize="2147483647" maxBufferSize="2147483647">
        <security mode="Transport"/>
      </binding>
    </basicHttpBinding>
  </bindings>
</system.serviceModel>
```

Configure TCP bindings

This sample shows several different binding possibilities. You can define any number of binding definitions in your configuration, but it's the *binding* and *bindingConfiguration* attributes on the *endpoint* which determine the binding used. In the sample below the endpoints are using the binding named "tcpBindingDefault" and the other bindings shown are for information only. If you try this sample and wish to test the various bindings, make sure that you change both the *binding* and *bindingConfiguration* on both endpoints.

This sample also shows that you can use either standard or custom WCF bindings. DevForce uses a *CustomBinding*, but this is not a requirement for your application. The standard bindings are better documented by Microsoft and therefore usually easier to use. Standard bindings do not offer compression.

Note that the addresses used here do not use the *.svc extension because the services are not hosted in IIS. You can use net.tcp with an IIS-hosted *EntityServer* if you install and configure the Windows Process Activation Service. *net.tcp* is the name for the TCP scheme in WCF.

You can use net.tcp with Silverlight, but we don't yet have a sample.

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="EntityService"
                address="net.tcp://localhost:9009/EntityService"
                binding="netTcpBinding" bindingConfiguration="tcpBindingDefault"
                contract="IdeaBlade.EntityModel.IEntityServiceContract"
        />
      <endpoint name="EntityServer"
                address="net.tcp://localhost:9009/EntityServer"
                binding="netTcpBinding" bindingConfiguration="tcpBindingDefault"
                contract="IdeaBlade.EntityModel.IEntityServerContract"
        />
```

```
  </client>
  <bindings>

   <netTcpBinding>

    <!-- Standard tcp binding - uses Windows auth and transport security by default. -->
     <binding name="tcpBindingDefault" maxReceivedMessageSize="2147483647" />
    <!-- Tcp binding using message security. -->
     <binding name="tcpBindingMessage" maxReceivedMessageSize="2147483647">
      <security mode="Message" />
     </binding>
   </netTcpBinding>
   <customBinding>
    <!-- DevForce custom binding with TCP and compression -->
     <binding name="tcpBindingCustom">
      <gzipMessageEncoding>
       <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
        maxArrayLength="2147483647" />
      </gzipMessageEncoding>
      <tcpTransport maxReceivedMessageSize="2147483647" />
     </binding>
    <!-- Another custom binding with TCP, security and compression -->
     <binding name="tcpBindingCustomSecure">
      <gzipMessageEncoding>
       <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
        maxArrayLength="2147483647" />
      </gzipMessageEncoding>
      <windowsStreamSecurity />
      <tcpTransport maxReceivedMessageSize="2147483647" />
     </binding>
   </customBinding>
  </bindings>
  <extensions>
   <bindingElementExtensions>
    <add name="gzipMessageEncoding" type="IdeaBlade.Core.Wcf.Extensions.GZipMessageEncodingElement, IdeaBlade.Core"/>
   </bindingElementExtensions>
  </extensions>
 </system.serviceModel>
</configuration>
```

Specify an EntityServer endpoint

DevForce will by default automatically determine the *EntityServer* endpoint address, but you can instead specify the information in your client-side configuration file. This may be necessary when overriding DevForce defaults. The name of the configuration element is usually just "EntityServer", but will depend on whether you're using data source extensions and/or custom composition contexts.

If you're using a data source extension or custom composition context the *EntityServer* service will be named with these attributes. For DevForce to find the endpoint name in the configuration, it must follow a few rules.

The name should be in the format "EntityServer_dsext+ccname", where *dsext* is the data source extension and *ccname* is the composition context name. For example, if using both a data source extension named "ABC" and a composition context named "Special", then the endpoint name should be "EntityServer_ABC+Special". If the specific name isn't found DevForce will default to its standard programmatic configuration.

```
<configuration>
    <endpoint name="EntityServer_TenantA"
            address="http://localhost:9009/EntityServer_TenantA.svc/sl"
            binding="customBinding" bindingConfiguration="customBinaryBinding"
            contract="IdeaBlade.EntityModel.IEntityServerContract"
      />
```

Additional resources

Securing Access to Services for Silverlight
Custom Bindings