Contents

An n-tier client application may **connect to more than one application server**.

In most n-tier applications, the client application needs to communicate with only a single "host" of the *EntityServer* services. By "host" we mean the machine name, port, protocol and application name forming the URL information commonly found in the <objectServer> element in your configuration file. We can consider each "host" to be a separate application server.

Communicating with a single application server, to login, and perform queries and saves, is sufficient for many applications. But DevForce also allows a client application to work with multiple application servers, both at the same time and as needed based on application requirements. For example, your client application may need to work with HR services at site A while also working with payroll services at site B. Or maybe your application will sometimes connect to a local application server but at other times needs to work with the server at headquarters. (In all cases the application server is a DevForce *EntityServer*, not custom or third party web services.)

When your application needs to work with multiple application servers, you do so through something called a *serviceKey*.

Using ServiceKeys

ServiceKeys provide address information for the application servers your client application may work with. These keys are defined in your configuration file and allow you to specify a name and address for each application server. The *serviceKey* provides additional flexibility in defining the application server over the fixed information in the <objectServer> element.

You choose the *serviceKey* to use, and thus the application server, when you construct an EntityManager.

```
var entityManager = new NorthwindIBEntityManager(new EntityManagerContext(serviceKey: "foo"));
```

```
Dim entityManager = New NorthwindIBEntityManager(New EntityManagerContext(serviceKey := "foo"))
```

That *serviceKey* name, here we've used the silly name of "foo", tells DevForce to look for a *serviceKey* with this name in the configuration file to obtain the address of the application server. (You can still use the <*system.serviceModel*> section to define the service endpoints in advanced configurations, we describe that below.)

Here's the configuration information for the "foo" key:

```
<objectServer remoteBaseURL="http://www.contoso.com" serviceName="BigApp\EntityService.svc" serverPort="80" >
  <serviceKeys>
    <serviceKey name="foo" remoteBaseURL="http://foo.contoso.com" serviceName="SmallApp\EntityService.svc" serverPort="8080" />
  </serviceKeys>
</objectServer>
```

An *EntityManager* constructed with this *serviceKey* will query and save to the application server located at the http://foo.contoso.com address. Other *EntityManagers* in the application can continue to communicate with the *default* application server at http://www.contoso.com.

Difference from data source extension and composition context

In other topics we've discussed the use of multiple EntityServers. An *EntityServer* is created to match the specifics of the requesting *EntityManager*. If you are using either a data source extension or custom composition context when you create your *EntityManager*, then it will communicate with an *EntityServer* having those same characteristics. How does this differ from service keys?

When using either a data source extension or custom composition context your application will still communicate with the same application server. The address of the *EntityService* will be the same, and that *EntityService* will determine the specific *EntityServer* to be used.

With a *serviceKey*, an entirely different application server, and thus *EntityService*, will be used. That application server might be on a different machine altogether, or use a different port or protocol. You can still use data source extensions and/or custom composition contexts when using a *serviceKey*.

The default key

When constructing an EntityManager you can provide a *serviceKey* in the constructor arguments. By default, if the *serviceKey* is not provided then DevForce will use the *default* key to determine the application server it will use.

The determination of this *default* is worth noting. Generally the *remoteBaseURL* and other <objectServer> attributes will be used to determine the *default key*. (In Silverlight applications this information is set for you based on the address the XAP was loaded from.) If this information isn't defined then the *serviceKeys* are searched. If a *serviceKey* named "default" is found (the search is case-insensitive) then that *serviceKey* is used as the *default*, otherwise the first *serviceKey* defined is used as the *default*.

The following are different ways to specify your keys. In each, the *default key* found by DevForce will be the same - the one resolving to an address of http://www.contoso.com/Samples/EntityService.svc.

```
<objectServer remoteBaseURL="http://www.contoso.com" serviceName="Samples\EntityService.svc" serverPort="80" >
  <serviceKeys>
    <serviceKey name="Backup" remoteBaseURL="http://test.contoso.com" serviceName="OldSamples\EntityService.svc"
serverPort="8080" />
  </serviceKeys>
 </objectServer>
```

```
<objectServer>
  <serviceKeys>
    <serviceKey name="Default" remoteBaseURL="http://www.contoso.com" serviceName="Samples\EntityService.svc" serverPort="80" /
>
    <serviceKey name="Backup" remoteBaseURL="http://test.contoso.com" serviceName="OldSamples\EntityService.svc"
serverPort="8080" />
  </serviceKeys>
 </objectServer>
```

```
<objectServer>
  <serviceKeys>
    <serviceKey name="BOS1" remoteBaseURL="http://www.contoso.com" serviceName="Samples\EntityService.svc" serverPort="80" />
    <serviceKey name="Backup" remoteBaseURL="http://test.contoso.com" serviceName="OldSamples\EntityService.svc"
serverPort="8080" />
  </serviceKeys>
 </objectServer>
```

On the server

The purpose of *ServiceKeys* is to allow you to specify the various application servers your client application may communicate with. But, since it's often easier to copy configuration information between client and server config files, the server can also use the *serviceKeys* in some configurations.

If an *EntityServer* is deployed as either a console application or Windows service then DevForce must determine a "base address" for its services. This address is determined based on the default key logic described above.

Using a system.serviceModel

If you've found you need the full control the <*system.serviceModel*> configuration section offers, you can still use it to define endpoints when using *serviceKeys*. DevForce will look for endpoint names with the {ServiceKey}_{ServiceName} format. In other words, if your *serviceKey* name is "foo", then DevForce will look for endpoints named "foo_EntityService" and "foo_EntityServer". Note that the bindings do not need to be the same across *serviceKeys*, so for instance one application server might use https while another uses http.

```
<system.serviceModel>
  <client>
<!-- Endpoints for the "default" server -->
    <endpoint name="EntityService"
      address="http://localhost:9009/EntityService.svc/sl"
      binding="customBinding" bindingConfiguration="CustomBinding"
      contract="IdeaBlade.EntityModel.IEntityServiceContractAsync"
      />
    <endpoint name="EntityServer"
```

```
        address="http://localhost:9009/EntityServer.svc/sl"
        binding="customBinding" bindingConfiguration="CustomBinding"
        contract="IdeaBlade.EntityModel.IEntityServerContractAsync"
        />
<!-- Endpoints using the "foo" serviceKey -->
    <endpoint name="foo_EntityService"
        address="http://fooserver:9009/EntityService.svc/sl"
        binding="customBinding" bindingConfiguration="CustomBinding"
        contract="IdeaBlade.EntityModel.IEntityServiceContractAsync"
        />
    <endpoint name="foo_EntityServer"
        address="http://fooserver:9009/EntityServer.svc/sl"
        binding="customBinding" bindingConfiguration="CustomBinding"
        contract="IdeaBlade.EntityModel.IEntityServerContractAsync"
        />
  </client>
  <bindings>
    <customBinding>
      <binding name="CustomBinding">
        <binaryMessageEncoding />
        <httpTransport maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
      </binding>
    </customBinding>
  </bindings>
</system.serviceModel>
```