Contents

To use DevForce, you must first **create an EntityManager** from which you'll do most of your work.

Construct a bare EntityManager

Call "new" to construct a new instance of the *EntityManager* class. It can be as simple as

```
manager = new EntityManager();
```

```
manager = New EntityManager()
```

You'll generally work directly with the sub-typed domain-specific *EntityManager* class that DevForce generates for you, but you can always construct the base *EntityManager* class and work with that when wanted. The base *EntityManager* can work with any domain model.

This default constructor assumes that you want to contact the server as soon as possible and tries to connect to the EntityServer. This connection will be done asynchronously in Silverlight and Windows Store applications. Since a connect attempt might be slow in some installations, and can fail if the *EntityServer* doesn't respond, it's often a good idea to perform the connect explicitly, and asychronously if you don't want to block your UI.

You won't be able to do anything with the *manager* until you've both connected to an *EntityServer* and established a security context for the EntityManager. That isn't readily apparent out-of-the-box since DevForce enables anonymous authentication by default. But you are sure to require authentication constraints of some sort, and when you do, the EntityManager will refuse to send any messages to the server until those constraints are met.

Please establish your security plan early in the development process.

Construct the EntityManager for your entity model

The entity model you created using the Entity Data Model Designer also generated a custom EntityManager for you, one that derives from the DevForce *EntityManager* class and is enriched with additional members that make it easier to work with your entity model.

Peek inside the generated code file (e.g., NorthwindIBEntitiesIB.Designer.cs). Notice the custom EntityManager near the top of the file. It might looks a bit like this:

```
IbEm.DataSourceKeyName(@"NorthwindEntities")]
public partial class NorthwindEntities : IbEm.EntityManager {
// ..
}
```

```
<IbEm.DataSourceKeyName("NorthwindEntities")>
Partial Public Class NorthwindEntities
 Inherits IbEm.EntityManager
 ' ..
End Class
```

You can construct one of these as you did the bare DevForce *EntityManager*.

```
manager = new NorthwindEntities();
```

```
manager = New NorthwindEntities()
```

Look further in the generated code to find several constructors that enable you to determine initial characteristics of the EntityManager. The first is the constructor you called above; all of its parameters are optional.

```
public NorthwindManager(
  bool shouldConnect=true, // Whether to start connecting to the server immediately
  string dataSourceExtension=null, // optional target environment
  IbEm.EntityServiceOption entityServiceOption=IbEm.EntityServiceOption.UseDefaultService,
  string compositionContextName=null)
   : base(shouldConnect, dataSourceExtension, entityServiceOption, compositionContextName) { }
```

```
Public Sub New(Optional ByVal _
  shouldConnect As Boolean =True, _
 Optional ByVal dataSourceExtension As String =Nothing, _
 Optional ByVal entityServiceOption As IbEm.EntityServiceOption =IbEm.EntityServiceOption.UseDefaultService, _
 Optional ByVal compositionContextName As String =Nothing)
 MyBase.New(shouldConnect, dataSourceExtension, entityServiceOption, compositionContextName)
End Sub
```

Another generated constructor gathers all options together into an *EntityManagerContext*. The *EntityManagerContext* allows for additional settings not available in the other constructors, including options to control refetch behavior and connect to a non-default application server.

```
public NorthwindManager(
   IbEm.EntityManagerContext entityManagerContext) : base(entityManagerContext) { }
```

```
Public Sub New(ByVal entityManagerContext As IbEm.EntityManagerContext)
 MyBase.New(entityManagerContext)
End Sub
```

These constructors are available for the base *EntityManager* class too. You can read the DevForce *API documentation* for details about these parameters and the other constructors.

You might consider a brief digression now to learn about creating a disconnected EntityManager and creating EntityManagers that target different databases under different execution scenarios.

Custom *EntityQuery* properties

The custom *EntityQuery* properties are one reason you may prefer to create a new custom EntityManager such as *NorthwindEntities* rather than a bare-bones *EntityManager*.

Peeking again at the generated code for *NorthwindEntities* we find query properties generated for each of the entity types. Here is one for querying *Customers*.

```
public IbEm.EntityQuery<Customer> Customers {
   get { return new IbEm.EntityQuery<Customer>("Customers", this); } // IbEm == IdeaBlade.EntityModel
}
```

```
Public ReadOnly Property Customers() As IbEm.EntityQuery(Of Customer)
 Get ' IbEm == IdeaBlade.EntityModel
  Return New IbEm.EntityQuery(Of Customer)("Customers", Me)
 End Get
End Property
```

It's a nice touch - a bit of syntactic sugar - that can make a LINQ query a little easier for developers to read and write:

```
query1 = northwindManager.Customer.Where(...);
query2 = bareManager.GetQuery<Customer>().Where(...);
```

```
query1 = northwindManager.Customer.Where(...)
query2 = bareManager.GetQuery(Of Customer)().Where(...)
```

The two queries are functionally the same. Most developers prefer *query1*.