Contents

- How to use the EntityManagerProvider
- Providing connection details
    - Custom ConnectionOptions
- Sample data
- Initializing the Fake Backing Store
- Using the EntityManagerDelegate to perform central tasks
- Synchronizing changes

The core abstraction layer in Punch is the EntityManagerProvider. It shields the application from the details of obtaining a correctly configured and authenticated EntityManager. By switching the connection details of the EntityManagerProvider, the application can run against the database, the fake backing store, or provide sample data at design time without the need for changing the application itself.

How to use the EntityManagerProvider

An EntityManagerProvider is created for a specific EntityManager type and made available to the application through the MEF catalog via an appropriate export.

The following example shows how to create a single shared EntityManagerProvider for applications that wish to use a single shared EntityManager.

```
using System.ComponentModel.Composition;
using System.ComponentModel.Composition.Hosting;
using Cocktail;
using DomainModel;
namespace TempHire
{
  public class AppBootstrapper : CocktailMefBootstrapper<HarnessViewModel>
  {
    protected override void PrepareCompositionContainer(CompositionBatch batch)
    {
      base.PrepareCompositionContainer(batch);
      // Inject the shared EntityManagerProvider
      batch.AddExportedValue<IEntityManagerProvider<TempHireEntities>>(
            new EntityManagerProvider<TempHireEntities>());
    }
  }
}
```

The next example shows how to support an application that creates multiple EntityManagerProvider instances.

```
using System.ComponentModel.Composition;
using Cocktail;
using DomainModel;
namespace TempHire
{
  public class EntityManagerProviderFactory
  {
    [Export]
    public IEntityManagerProvider<TempHireEntities> TempHireEntityManagerProvider
    {
      get
      {
        return new EntityManagerProvider<TempHireEntities>();
      }
    }
  }
}
```

Providing connection details

An EntityManagerProvider must be given details about where its EntityManager should connect or whether it should connect at all. This is accomplished by an independent ConnectionOptions object. Punch provides three out of the box ConnectionOptions. See *ConnectionOptions.Default*, *ConnectionOptions.Fake* and *ConnectionOptions.DesignTime*.

ConnectionOptions.Default is used when nothing else is specified and connects to a server based on the information in the app.config or web.config.

ConnectionOptions.Fake is used when the EntityManager should connect to the Fake Backing Store.

ConnectionOptions.DesignTime is used when the EntityManager is used at design time. In this mode, it won't connect to a server at all and instead gets its cache populated with sample data.

The following example shows how to configure an EntityManagerProvider with the Fake Backing Store ConnectionOptions.

```
using System.ComponentModel.Composition;
using Cocktail;
using DomainModel;
namespace TempHire
{
  public class EntityManagerProviderFactory
  {
    [Export]
    public IEntityManagerProvider<TempHireEntities> TempHireEntityManagerProvider
    {
      get
      {
        return new EntityManagerProvider<TempHireEntities>()
            .Configure(provider => provider.WithConnectionOptions(ConnectionOptions.Fake.Name));
      }
    }
  }
}
```

Custom ConnectionOptions

In addition to the out of the box ConnectionOptions, one can create any number of custom ConnectionOptions or override the out of the box ConnectionOptions by implementing one or more IConnectionOptionsResolver.

The following snippet shows an example of an IConnectionOptionsResolver that overrides the default ConnectionOptions to connect to the Fake Backing Store instead.

```
using Cocktail;
using Security.Composition;
namespace TempHire
{
  public class ConnectionOptionsResolver : IConnectionOptionsResolver
  {
    public ConnectionOptions GetConnectionOptions(string name)
    {
      // Replace default connection info in order to connect to fake store
      if (name == ConnectionOptions.Default.Name)
        return ConnectionOptions.Fake
            .WithName(ConnectionOptions.Default.Name);
      return null;
    }
  }
}
```

Sample data

Punch supports the use of the DevForce Fake Backing Store and Design Time use of an EntityManager. In both cases, sample data must be provided by means of a SampleDataProvider. Multiple SampleDataProviders are supported.

Punch automatically discovers all SampleDataProviders through MEF and uses them to populate the EntityManager cache and/or Fake Backing Store. The following code snippet shows how to implement a SampleDataProvider.

```
using System;
using System.ComponentModel.Composition;
using Cocktail;
namespace SampleData
{
  [Export(typeof(ISampleDataProvider<NorthwindIBEntities>))]
  public class SampleDataProvider : ISampleDataProvider<NorthwindIBEntities>
  {
    #region ISampleDataProvider<NorthwindIBEntities> Members
    void ISampleDataProvider<NorthwindIBEntities>.AddSampleData(NorthwindIBEntities manager)
    {
```

```
            AddCustomers(manager);
        }
        #endregion
        private static void AddCustomers(NorthwindIBEntities manager)
        {
            var customer1 = new Customer
                            {
                                CustomerID = Guid.NewGuid(),
                                CompanyName = "Company1",
                                ContactName = "John Doe",
                                Address = "SomeAddress",
                                City = "SomeCity",
                                PostalCode = "11111"
                            };
            manager.AttachEntity(customer1);
            var customer2 = new Customer
                            {
                                CustomerID = Guid.NewGuid(),
                                CompanyName = "Company2",
                                ContactName = "Jane Doe",
                                Address = "SomeAddress",
                                City = "SomeCity",
                                PostalCode = "11111"
                            };
            manager.AttachEntity(customer2);
        }
    }
}
```

Initializing the Fake Backing Store

In order for the Fake Backing Store to return data when queried, it needs to be initialized with sample data first.

The following example shows how to initialize the Fake Backing Store with the provided sample data during bootstrapping.

```
public class AppBootstrapper : CocktailMefBootstrapper<ShellViewModel>
{
    [Import]
    public IEntityManagerProvider<TempHireEntities> EntityManagerProvider;
    protected override async Task StartRuntimeAsync()
    {
        await EntityManagerProvider.InitializeFakeBackingStoreAsync();
    }
}
```

Using the EntityManagerDelegate to perform central tasks

Applications are often required to perform tasks in response to events from the EntityManager. Punch provides the means to perform these tasks centrally and independent of the actual EntityManager provider used at runtime via the EntityManagerDelegate.

The EntityManagerDelegate provides optional methods that the developer can override and that are called in response to the corresponding event on the EntityManager. The following snippet shows an example of an EntityManagerDelegate.

```
using System.Collections.Generic;
using System.Linq;
using Cocktail;
using Common.Messages;
using DomainModel;
using IdeaBlade.EntityModel;
namespace TempHire
{
    public class EntityManagerDelegate : EntityManagerDelegate<TempHireEntities>
    {
        public override void OnEntityChanged(TempHireEntities source, EntityChangedEventArgs args)
        {
            EventFns.Publish(new EntityChangedMessage(args.Entity));
        }
        public override void OnSaving(TempHireEntities source, EntitySavingEventArgs args)
        {
```

```
        // Add necessary aggregate root object to the save list for validation and concurrency check
        List<EntityAspect> rootEas = args.Entities.OfType<IHasRoot>()
            .Select(e => EntityAspect.Wrap(e.Root))
            .Distinct()
            .Where(ea => ea != null && !ea.IsChanged && !ea.IsNullOrPendingEntity)
            .ToList();
        rootEas.ForEach(ea => ea.SetModified());
        rootEas.ForEach(ea => args.Entities.Add(ea.Entity));
      }
      public override void OnSaved(TempHireEntities source, EntitySavedEventArgs args)
      {
        if (args.CompletedSuccessfully)
            EventFns.Publish(new SavedMessage(args.Entities));
      }
    }
}
```

Synchronizing changes

In DevForce, each instance of an EntityManager manages a client-side cache of entities. If entities get changed and saved in an EntityManager, one might want to synchronize these changes to all other EntityManagers that keep a copy of the same entities in their respective cache. Punch has built-in support for change synchronization, but it is disabled by default.

To enable change synchronization, implement an EntityManagerSyncInterceptor to control which entities should be exported from the source EntityManager and which entities should be imported to the target EntityManagers. The following snippet shows how to implement an EntityManagerSyncInterceptor.

```
using Cocktail;
using IdeaBlade.EntityModel;
namespace Synchronization
{
  public class SyncInterceptor : EntityManagerSyncInterceptor
  {
    public override bool ShouldExportEntity(object entity)
    {
      return true;
    }
    public override bool ShouldImportEntity(object entity)
    {
      // Only import if the importing EntityManager holds a copy of the entity in its cache
      return EntityManager.FindEntity(EntityAspect.Wrap(entity).EntityKey) != null;
    }
  }
}
```

The synchronization preserves pending changes in the importing EntityManager, so if an EntityManager holds a copy of an entity with pending changes and another EntityManager saves the same entity, triggering change synchronization, the pending changes will be merged with the saved changes.