Contents

- Dependency Injection
- Inversion of Control (IoC)
- Managed Extensibility Framework (MEF)
- Additional resources

Dependency injection is an architectural design pattern applied throughout Punch and something we highly encourage you use in your own applications.

Dependency Injection

The basic idea behind dependency injection is to externalize dependencies between classes so they can be controlled and resolved at a higher level. Let's look at a common dependency scenario to illustrate the pattern.

```
public class A
{
  private B _objectB;
  public A()
   {
     _objectB = new B();
   }
}
public class B
{
}
```

In this illustration, class A uses class B. There's nothing technically wrong with this approach. Let's imagine for a second, though, that we want an A that works with a subclass of B, because we want to extend the out-of-the-box B implementation. Currently, we have no way of replacing B with a subclass. Even subclassing A isn't going to help us here, because the dependency is private. The way out of this bind is to externalize the dependency.

```
public class A
{
  private B _objectB;
  public A(B objectB)
   {
     _objectB = objectB;
   }
}
public class B
{
}
```

With just a minor change, we can now supply B from the outside, whenever we create an A. The constructor takes an instance of B and injects it into the instance of A during creation. This is what dependency injection is all about. A simple yet powerful approach.

Inversion of Control (IoC)

Hand in hand with dependency injection goes the concept of Inversion of Control (IoC). Without mentioning it, inversion of control is what we achieved in the above illustration. In the initial illustration, the control lies with A. A creates and controls B. In the second illustration, we inverted that control and now the control lies with the part of the application that creates an instance of A. It decides and controls which specific B is injected into A.

In a modern application, this control is typically delegated to an IoC container for ultimate flexibility and maintainability. An IoC container manages instances and resolves dependencies between them.

```
A objectA = container.CreateA();
```

The above snippet illustrates what we mean by that. Instead of A objectA = new A(new B()), we request an instance from our fictitious container. The idea is that the container knows based on the current scope and configuration of the application, which specific B to supply to the instance of A the container is creating for us.

Managed Extensibility Framework (MEF)

There are several IoC implementations that one can use to build their applications. Microsoft Unity and Autofac are two popular ones. In Punch we are using the Managed Extensibility Framework. MEF is part of Silverlight and .NET, so no additional

assemblies need to be downloaded or included with your application. Moreover, DevForce already uses MEF for the same purpose and by using MEF in Punch, we are naturally extending what DevForce already does for us.

MEF also adds one powerful aspect to the mix: the ability to auto-discover part implementations dynamically at runtime without configuration of the container. This is very powerful when it comes to making your own applications extensible.

We are not going to explain MEF in detail here. To learn all about MEF, please visit the MEF documentation on CodePlex.

So, to continue with our two classes, the following snippet shows how to implement them with MEF.

```
[Export]
public class A
{
  private B _objectB;
   [ImportingConstructor]
  public A(B objectB)
   {
      _objectB = objectB;
   }
}
[Export]
public class B
{
}
```

The basic concepts behind MEF are exports and imports. The various parts are exported using a form of the MEF ExportAttribute and then imported using a form of the MEF ImportAttribute, ImportManyAttribute or ImportingConstructorAttribute. We can then use the static Composition class in Punch to obtain an instance of A with all its dependencies resolved.

```
A objectA = Composition.GetInstance<A>();
```

Additional resources

For another perspective on Dependency Injection and Inversion of Control, visit the Punch blog and read Rob Eisenberg's blog post on the subject.