

Castle Windsor

Tuna Toksoz

March 23, 2010

Who am I?

Introduction

- Dependency Injection

- Inversion of Control Container

Castle Windsor

- Why Castle Windsor?

- Configuration

- Extensibility points

 - Facilities

 - Events

 - Dependency resolution control mechanisms

 - Lifestyle control mechanisms

 - Component initialization control mechanisms

Result

Who am I?

- ▶ Senior student at Bogazici University
- ▶ (Passive) committer at Castle and NHibernate
- ▶ Blogger at his own blog and also on devlicio.us
- ▶ Has an interest in Robotics and its applications

What is DI all about?

- ▶ It is a pattern in Martin Fowler's book
- ▶ Depends on the principle of providing dependencies from the outside
- ▶ Made up of 3 components
 - ▶ Dependent
 - ▶ Dependency
 - ▶ Dependency provider

Why should we use DI?

- ▶ Loosely coupled components/services
- ▶ Increased testability
- ▶ Reduced cost of changes in later stages of development
- ▶ Ability to change implementations between testing and deployment

Why should not we use DI?



Dependency Injection Methods

- ▶ Constructor Injection
- ▶ Property Injection
- ▶ Method Injection

Dependency Injection Methods - Examples

► Constructor Injection

```
public CurrentBatteryLevelStatisticsCollector(IObjectSource objectSource, IEventAggregator eventAggregator)
    : base(objectSource)
{
    this.eventAggregator = eventAggregator;
    this.batteryLevels = new Dictionary<ObjectBase, float>();
}
```

► Property Injection

```
public class BasicEnvironment
{
    public IObjectSource ObjectSource { get; set; }
}
```

► Method Injection

Inversion of Control Container

- ▶ A point where all components are registered and being accessed
- ▶ A component which resolves dependencies of a requested component automatically
- ▶ Enables us to change implementations without much trouble

Why Castle Windsor?

- ▶ A popular framework
- ▶ Active development
 - ▶ 118 commits between October 2009 and February 2010.
 - ▶ 2nd version
- ▶ Extensibility points

Castle Windsor Configuration

- ▶ XML Configuration
- ▶ Fluent Configuration
- ▶ Binsor/Boo Configuration

XML Configuration

Cons

- ▶ Old school
- ▶ Error-prone

Pros

- ▶ Ability to change without compilation

```
<?xml version="1.0" ?>
<castle>
  <components>
    <component id="HtmlTitleRetriever" type="WindsorSample.HtmlTitleRetriever, WindsorSample"/>
    <component id="StringParsingTitleScraper" service="WindsorSample.ITitleScraper, WindsorSample"
      type="WindsorSample.StringParsingTitleScraper, WindsorSample"/>
    <component id="HttpFileDownloader" service="WindsorSample.IFileDownloader, WindsorSample"
      type="WindsorSample.HttpFileDownloader, WindsorSample"/>
  </components>
</castle>
```

Fluent/Programmatic Configuration

Cons

- ▶ Very hard, if not impossible, to change after compilation

Pros

- ▶ Compile time checking
- ▶ Intellisense
- ▶ AllTypes Of

Fluent/Programmatic Configuration - Cont'd

```
public void Install(IWindsorContainer container, Castle.MicroKernel.IConfigurationStore store)
{
    container
        .Register(Component.For<ICatalogService>()
            .ImplementedBy<MyCatalogService>().LifeStyle.Singleton)
        .Register(Component.For<IPriceService>()
            .ImplementedBy<PriceService>()
            .Named("priceService")
            .DependsOn(new {taxRate=0.18f})
            .OnCreate((kernel, service)=>service.Name="priceService"))
        .Register(AllTypes.Of<IConsoleCommandInterpreter>()
            .FromAssembly(typeof(IConsoleCommandInterpreter).Assembly)
            .WithService.FirstInterface());
}
```

Boo/Binsor Configuration

- ▶ Compile/Runtime checking
- ▶ Intellisense (MonoDevelop)
- ▶ Easy to change after compilation of application
- ▶ Easier configuration with the help of Boo extensibility(macros)

```
component mycompfactory, MyCompFactory  
component mycomp, MyComp:  
    createUsing @mycompfactory.Creat
```

Extensibility points

- ▶ Facilities
- ▶ Events
- ▶ Dependency resolution control mechanisms
 - ▶ Subdependency Resolver
 - ▶ Handler Selector
 - ▶ Interceptor Selector
- ▶ Lifestyle control mechanisms
- ▶ Object initialization control mechanisms

Facilities

- ▶ MK/Windsor's points of configurations
- ▶ A point where a group of related configuration (microkernel) tasks take place

Available Facilities

- ▶ Active Record Integration
- ▶ Automatic Transaction Management
- ▶ Batch Registration - Obsolete
- ▶ Event Wiring
- ▶ Factory Support
- ▶ Nhibernate Integration
- ▶ Synchronize
- ▶ WCF Facility

Eventler

- ▶ ComponentRegistered
- ▶ ComponentUnregistered
- ▶ ComponentModelCreated
- ▶ ComponentCreated
- ▶ ComponentDestroyed
- ▶ DependencyResolving
- ▶ and several others

Eventler - Code

```
public class EnrichWithFacility : AbstractFacility
{
    public delegate void ExtendComponentDelegate(IKernel kernel, object instance);
    public const string ExtendWithPropertyKey = "extendwith";
    protected override void Init()
    {
        Kernel.ComponentCreated += Kernel_ComponentCreated;
    }
    void Kernel_ComponentCreated(ComponentModel model, object instance)
    {
        if (model.ExtendedProperties.Contains(ExtendWithPropertyKey))
        {
            var action = model.ExtendedProperties[ExtendWithPropertyKey] as ExtendComponentDelegate;
            action(this.Kernel, instance);
        }
    }
}
```

Dependency resolution control mechanisms

- ▶ Subdependency Resolver
- ▶ Handler Selector
- ▶ Interceptor Selector

Subdependency Resolver

- ▶ Tells how a specific dependency of a component should be resolved
- ▶ We can either use an existing component or create a new one as the dependency
- ▶ Does not affect previously initialized components (MEF can do it)

Subdependency Resolver - Code

```
public class ServiceIdResolver : ISubDependencyResolver
{
    #region ISubDependencyResolver Members
    public bool CanResolve(CreationContext context, ISubDependencyResolver parentResolver,
        ComponentModel model, DependencyModel dependency)
    {
        return dependency.DependencyKey.ToLowerInvariant().Equals("serviceid") &&
            dependency.TargetType == typeof(string);
    }
    public object Resolve(CreationContext context, ISubDependencyResolver parentResolver,
        ComponentModel model, DependencyModel dependency)
    {
        return model.Name;
    }
    #endregion
}
```

Subdependency Resolver - Code 2

```
public class ArrayResolver : ISubDependencyResolver
{
    private readonly IKernel kernel;
    public ArrayResolver(IKernel kernel)
    {
        this.kernel = kernel;
    }

    public object Resolve(CreationContext context, ISubDependencyResolver contextHandlerResolver,
        ComponentModel model, DependencyModel dependency)
    {
        return kernel.ResolveAll(dependency.TargetType.GetElementType(), null);
    }

    public bool CanResolve(CreationContext context, ISubDependencyResolver contextHandlerResolver,
        ComponentModel model, DependencyModel dependency)
    {
        return dependency.TargetType != null && dependency.TargetType.IsArray &&
            kernel.HasComponent(dependency.TargetType.GetElementType());
    }
}
```

Spot the potential problem

Handler Selector

- ▶ Allows us to specify what to return as a result of `.Resolve<T>` calls depending on context
- ▶ Does not affect previously initialized components

Handler Selector - Code

```
public class DataAccessHandlerSelector : IHandlerSelector
{
    bool databaseIsDown = false;

    public DataAccessHandlerSelector()
    {
        DatabaseMonitor.OnChangedState +=
            state => databaseIsDown = state == DatabaseState.Down;
    }

    public bool HasOpinionAbout(string key, Type service)
    {
        return databaseIsDown && service == typeof(IRepository);
    }

    public IHandler SelectHandler(string key, Type service, IHandler[] handlers)
    {
        return handlers.Where(x => x.ComponentModel.Implementation == typeof(CacheOnlyRepository)).First();
    }
}
```

Interceptor Selector/Interceptor Model Selector/IProxyGeneration Hook

- ▶ Allows us to change cross-cutting concerns at runtime
- ▶ We can specify what interceptors should be attached
- ▶ Allows us to specify what methods to intercept

Lifestyle control mechanisms

Decides when to create a component

- ▶ Singleton
- ▶ PerThread
- ▶ PerWebRequest
- ▶ Transient
- ▶ Poolable
- ▶ Custom

Available Lifestyles - Singleton

```
public class SingletonLifestyleManager : AbstractLifestyleManager
{
    private volatile Object instance;

    public override void Dispose()
    {
        if (instance != null) base.Release( instance );
    }
    public override object Resolve(CreationContext context)
    {
        if (instance == null)
            lock (ComponentActivator)
                if (instance == null)
                    instance = base.Resolve(context);
        return instance;
    }
    public override bool Release(object instance)
    {
        return false;
    }
}
```

Component initialization control mechanisms

Contains the logic related to creation of components. They are called Activators in Castle terms.

- ▶ Default Activator (The place where dependency injection basically takes place)
- ▶ Accessor/Factory Activator (Used by Factory Support Facility)

Component initialization control mechanisms - Accessor Activator

```
public class AccessorActivator : DefaultComponentActivator
{
    public AccessorActivator(ComponentModel model, IKernel kernel,
        ComponentInstanceDelegate onCreation, ComponentInstanceDelegate onDestruction)
        : base(model, kernel, onCreation, onDestruction)
    {
    }

    protected override object Instantiate(CreationContext context)
    {
        String accessor = (String)Model.ExtendedProperties["instance.accessor"];

        PropertyInfo pi = Model.Implementation.GetProperty(accessor, BindingFlags.Public | BindingFlags.Static);

        return pi.GetValue(null, new object[0]);
    }
}
```

DI Advantages

- ▶ Reduced cost of change
- ▶ Increased testability
- ▶ Allows us to think in terms of component

Windsor

- ▶ A framework that is developed as a result of needs
- ▶ Easy integration with other frameworks
- ▶ Active development

Resources

- ▶ <http://castleproject.org>
- ▶ <http://groups.google.com/group/castle-project-users/>
- ▶ <http://ayende.com>