Castle Windsor

##### Uvod

* Castle pomaha v programovani tim, ze nemusime vytvaret instance nekterych trid. Dobre vysvetleni byla stavba drevostavby na youtube. Stavi se zed ktera ma stejne prvky. Kdyz prijde na radu okno, musime zmenit postup a to pokazde kdyz chceme pridat okno. Toto okno zaregistrujeme do castlu a uz ho nemusime znovu a znovu vyrabet. Castle ho zna, a jeho instanci nam da pokazde kdyz ji potrebujeme. Instance umi vyrobit I z xml souboru ContainerConfig, kdyz mu v bootstrapperu rekneme, kde ma tohle xml najit.

m\_globalContainer = new WindsorContainer(new XmlInterpreter(Path.Combine(m\_applicationDirectory, "MIR.PrintStorage.Scanning2.Shell.Container.config")));

nebo:

m\_windsorContainer = new WindsorContainer();

m\_windsorContainer.Install(Castle.Windsor.Installer.Configuration.FromXmlFile(@"MIR.Media.Coding.Container.config"));

* Pak muze resolvovat instance trid ktere jsou definovane v xml. Viz: [ContainerConfig](https://d.docs.live.net/b22fb0fb09218bf0/Nielsen%20%20prace/Moje%20poznamky%20Nielsen/ContainerConfig%20aneb%20Jak%20vymenovat%20obrazek%20na%20splash%20screenu%20podle%20zeme%20a%20verze.docx)
* Tridy ktere chce resolvovat a nejsou v xml, musi byt zaregistrovane. Instance tridy se da vytvorit rucne a pak zaregistrovat. Tim rikame, ze ma castle pouzivat tuto instanci kdyz si rekneme o instanci takoveto tridy.

IWindowManager windowManager = new WindowManager();

m\_globalContainer.Register(Component.For<IWindowManager>().Instance(windowManager));

* Aby bylo mozne vytvaret instance naseho conteineru, musi si container zaregistrovat sam sebe.

m\_globalContainer.Register(Component.For<IWindsorContainer>().Instance(m\_globalContainer).LifestyleSingleton());

* Pro pouziti Installeru (trida ve ktere registrujeme jine tridy do castlu) potrebujeme jednu prazdnou tridu, kterou si pridame do assembly ze ktere chceme instalovat. A zavolame metodu Instal na containeru. V tomto pripade rikame ze se instaluje z Core a z Shellu.

m\_globalContainer.Install(FromAssembly.This(), FromAssembly.Containing<ScanningCoreAssemblyIdentificator>());

**Installery**

**Zavolanim metody** m\_globalContainer.Install se pravdepodobne zavolaji všechny Install metody z těchto assembly. Nejspise je to tim ze instance windsorContaineru je singleton. Metody instal jsou ve tridach :

Shell:

1. ScanningInstaller : IwindsorInstaller
2. PrintStorageDbAccessInstaller : IwindsorInstaller
3. MediaDataDbAccessInstaller : IwindsorInstaller

Core:

1. MappingInstaller : IWindsorInstaller

V těchto Install metodach jsou zaregistrovane všechny objekty které ma Castle znat, aby nam mohl poskytovat jejich instance.

##### Installers

##### Component registration using conventions

[Convention over configuration](https://en.wikipedia.org/wiki/Convention_over_configuration) is popular subject these days so why not to apply it to the component registration. Instead of writing boring:

Component.For<IFooRepository>()

.ImplementedBy<FooRepository>()

.Lifestyle.PerWebRequest,

...

Component.For<IBarRepository>()

.ImplementedBy<BarRepository>()

.Lifestyle.PerWebRequest

We may write just once:

Classes.**FromThisAssembly**()

.**BasedOn**(**typeof**(IRepository<>))

.WithService.**AllInterfaces**()

.Lifestyle.PerWebRequest

Castle Windsor is very flexible when it comes to registering components by convention, we may scan selected assemblies and/or namespaces, we may even select components to register by testing component Type.

*PITFALL: Avoid creating conventions based on type name (e.g. register all classes that have names ending with Repository) as much as possible. It is always better to create empty marker interface e.g. IApplicationService and use it to register all necessary components.*

##### Interceptors

Interceptors are most powerful Castle Windsor feature that brings power of [aspect oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) to .NET. Interceptors can be used to implement transaction management, logging, security checks, we may use them to gather performance related statistics and for many other purposes.

Here is a simple interceptor that log the invocations of all component methods:

**public** **class** **EventTracingInterceptor** : **IInterceptor** {

**public** **void** **Intercept**(IInvocation invocation) {

EventTracer.**AddEvent**("BEFORE " + invocation.Method.Name);

**try** {

*// call original method, we may inspect method arguments,*

*// generic parameters, return value and many others*

invocation.**Proceed**();

}

**finally** {

EventTracer.**AddEvent**("AFTER " + invocation.Method.Name);

}

}

}

[Interceptor(typeof(EventTracingInterceptor))]

**public** **class** **Service** : **IService** {

**public** **void** **Foo**() { ... }

**public** **void** **Bar**() { ... }

}

container.**Register**(

*// interceptors work only when you expose your*

*// components via interfaces.*

*// here I registered interceptors by using*

*// attributes on Service class but you may also*

*// use fluent api.*

Component.For<IService>().ImplementedBy<Service>(),

Component.For<EventTracingInterceptor>()

);

IService service = container.Resolve<IService>();

service.**Foo**();

When you start writing your own interceptors it is generally advisable to create custom attribute e.g. TransactionalAttribute to mark classes that should have interceptors attached. Then you should write your own facility that will scan all components registered in container and will attach interceptor for these marked with your custom attribute. [Here is a good example of this approach](http://blog.willbeattie.net/2010/09/implementing-custom-castle-windsor.html)used to implement caching.

##### AsFactory

Pokud chci pouzivat AsFactory musim to containeru rict:

container.AddFacility<TypedFactoryFacility>();

###### castle windsor most popular features

<https://marcin-chwedczuk.github.io/castle-windsor-most-popular-features>

**Castle Windsor most popular features**

**Typed factory**

When I follow good software development practices like [SOLID](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)) I find myself writing plenty of [factory](https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)) classes. These factory classes often fall in one of the two categories:

* I need to create instance of generic service for specific type e.g. I want to get service that implements ICommandHandler<TCommand> for TCommand type
* I must pass parameters and/or configuration to the service before I can use it e.g. HeuristicSearch service has quality constructor parameter to decide what solutions are good enough for the user

In cases like these we can use typed factory feature to generate factory implementations:

*// to enable typed factory we must add TypedFactoryFacility*

*// to the container*

container.AddFacility<TypedFactoryFacility>();

*// case I: get generic service instance for specific type*

**public** **interface** **ICommandHandler**<**TCommand**> {

**void** **Handle**(TCommand command);

}

**public** **interface** **ICommandHandlerFactory** {

ICommandHandler<T> Create<T>();

**void** Release<T>(ICommandHandler<T> instance);

}

**public** **class** **AddUserCommandHandler** : **ICommandHandler**<**AddUserCommand**> {

**public** **AddUserCommandHandler**(*/\* dependencies \*/*) { ... }

**public** **void** **Handle**(AddUserCommand command) { ... }

}

container.**Register**(

Component.For<ICommandHandler<AddUserCommand>>()

.ImplementedBy<AddUserCommandHandler>()

.LifeStyle.Transient,

*// tell Windsor that it should generate factory for me*

Component.For<ICommandHandlerFactory>()

.**AsFactory**()

);

*// usage*

ICommandHandlerFactory factory =

container.Resolve<ICommandHandlerFactory>();

ICommandHandler<AddUserCommand> handler =

factory.Create<AddUserCommand>();

handler.**Handle**(**new** **AddUserCommand**());

factory.**Release**(handler);

*// case II: pass configuration to the service*

**public** **interface** **IGreeter** {

**void** **Greet**();

}

**public** **interface** **IGreeterFactory** {

IGreeter **Create**(**string** greeting);

**void** **Release**(IGreeter instance);

}

**public** **class** **ConsoleGreeter** : **IGreeter** {

...

**public** **ConsoleGreeter**(**string** greeting

*/\* you may add other dependencies here,*

*\* e.g. ITextWrapper wrapper \*/*) { ... }

}

container.**Register**(

Component.For<IGreeter>()

.ImplementedBy<ConsoleGreeter>()

.**LifestyleTransient**(),

*// tell Windsor that it should generate factory for me*

Component.For<IGreeterFactory>()

.**AsFactory**()

);

*// usage*

IGreeterFactory greeterFactory =

container.Resolve<IGreeterFactory>();

IGreeter helloWorldGreeter = greeterFactory.**Create**("hello, world!");

IGreeter goodbyeGreeter = greeterFactory.**Create**("goodbye cruel world!");

helloWorldGreeter.**Greet**();

goodbyeGreeter.**Greet**();

greeterFactory.**Release**(helloWorldGreeter);

greeterFactory.**Release**(goodbyeGreeter);

Things to remember when using typed factory:

* Release method in factory interface is optional.  
  It is a good practice to always include Release method in factory interface and to release all instances created using factory when they are no longer needed
* In case of transient or per-web-request components that are disposable not releasing component will result in a memory leak
* Remember that some factories should be implemented manually especially these that contain domain knowledge e.g. factory that selects discount[strategy](https://en.wikipedia.org/wiki/Strategy_pattern) based on user profile

###### why would you use windsor asfactory

<https://stackoverflow.com/questions/25062673/why-would-you-use-windsor-asfactory>

[Why would you use Windsor AsFactory?](https://stackoverflow.com/questions/25062673/why-would-you-use-windsor-asfactory)

Why would you use Castle Windsor factory auto implementation feature: AsFactory() rather then asking for needed interface?

Example:

container.Register(Component.For<IEmailSender>().ImplementedBy<SmtpEmailSender>());

container.Register(Component.For<IEmailSenderFactory>().AsFactory().LifeStyle.Transient);

...

using (var factory = context.GetServiceFactory<IEmailSenderFactory>())

{

var emailSender = factory.CreateEmailSender();

emailSender.Send(message);

}

Why wouldn't you simply write:

var emailSender = context.GetServiceFactory<IEmailSender>();

emailSender.Send(message);

Effect is the same. Windsor will resolve IEmailSender as default registered implementation anyway so what is the point?

**Sometimes you'll write a class that requires a specific value when you need to resolve it. For instance:**

public class NumberWriter : INumberWriter

{

readonly int number;

readonly IStream stream;

public NumberWriter(int number, IStream stream)

{

this.number = number;

this.stream = stream;

}

public Write()

{

stream.Write(number);

}

}

You can't resolve an instance of this class without a number, and maybe you'd also like to specify the stream (console, file, printer, whatever). So, you define a factory:

public interface INumberWriterFactory

{

INumberWriter Create(int number);

INumberWriter Create(int number, IStream stream);

}

Now the following code will work:

public class RandomNumberGenerator

{

readonly INumberWriterFactory numberWriterFactory;

public RandomNumberGenerator(INumberWriterFactory numberWriterFactory)

{

this.numberWriterFactory = numberWriterFactory;

}

public void Generate()

{

Random random = new Random();

for (int i = 0; i < 10; i++)

{

// Writes to first IStream that Castle can resolve

var numberWriter = numberWriterFactory.Create(random.Next());

numberWriter.Write();

}

}

public void Generate(IStream stream)

{

Random random = new Random();

for (int i = 0; i < 10; i++)

{

// Writes to the given IStream

var numberWriter = numberWriterFactory.Create(random.Next(), stream);

numberWriter.Write();

}

}

}

2. To introduce a level of abstraction

Using a factory *can* insulate you from changes to how you'll need to create an object. For instance, if you'll need to create instances of objects and always use the same constructor parameter each time, you can create a concrete factory and then just use that everywhere instead of the one generated via AsFactory().

In other words, we could modify the behavior some of the code by burying the stream parameter in the factory so that a specific default stream is always used (for instance if an IStream cannot just be resolved from the container). Doing it this way means we wouldn't need to change the RandomNumberGenerator at all:

public class NumberWriterFactory : INumberWriterFactory

{

readonly IStream stream;

readonly IContainer container;

public NumberWriterFactory(IStream stream, IContainer container)

{

this.stream = stream;

this.container = container;

}

public INumberWriter Create(int number)

{

return container.Resolve<INumberWriter>(number, this.stream);

}

public INumberWriter Create(int number, IStream stream)

{

return container.Resolve<INumberWriter>(number, stream);

}

}

And no change in RandomNumberGenerator, but the behavior is changed:

public class RandomNumberGenerator

{

readonly INumberWriterFactory numberWriterFactory;

public RandomNumberGenerator(INumberWriterFactory numberWriterFactory)

{

this.numberWriterFactory = numberWriterFactory;

}

public void Generate()

{

Random random = new Random();

for (int i = 0; i < 10; i++)

{

// Writes to the IStream instance that the factory contains

var numberWriter = numberWriterFactory.Create(random.Next());

numberWriter.Write();

}

}

// the rest as before

}

Again, this is useful in the sense that if you were already using a factory interface, such as one implemented by using AsFactory(), you could easily swap it out for a new implementation. Doing this if you were already using a container instead is more difficult; it's harder to find the places you need to change and it's harder to swap out usage of the container to use a new type (i.e. a factory).

Note: you would have to create an INumberWriterFactoryFactory to inject the IStream into the concrete factory.

3. To keep usage of the IOC container in the composition root

There's a lot of folk who subscribe to the idea that there is only one [composition root](http://blog.ploeh.dk/2011/07/28/CompositionRoot/), and it is the *only* time a reference to the IOC container is permitted. Doing so can help you to avoid several anti-patterns, such as the [service locator](http://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/) pattern.

##### Installers

Castle Windsor installers allow us to group component registrations into reusable pieces of code. The real power of installers comes from the fact that we may pass them arguments or in other words we may configure them. For example installer may take a single argument that tells what lifestyle should be applied to all registrations contained in the installer. Such installer can be used in both ASP.NET MVC app when most of the components will be registered as PerWebRequest and in Windows service app where components will be registered as either Transient or Singleton.

Here is example of very simple installer:

**public** **class** **DummyModuleInstaller** : **IWindsorInstaller** {

**public** **void** **Install**(IWindsorContainer container, IConfigurationStore store) {

container.AddFacility<TypedFactoryFacility>();

*// add other installers, facilities etc.*

container.**Register**(

Component.For<DummyService>().LifeStyle.Transient

*// other components*

);

}

}

container.**Install**(**new** [] {

**new** **DummyModuleInstaller**()

});

##### IsFallback and IsDefault components

When we start grouping registrations into installers often we will find ourselves in situation that we want to register given service only when user of the installer didn’t provide she’s own implementation. We may achieve this by passing parameters to the installer but a fallback components are a better choice here. Components registered as fallbacks will be used by the container only when there is no other component that provides given service:

*// fallback is used when no other component*

*// for service is registered*

container.**Register**(

Component.For<IFooService>()

.ImplementedBy<FallbackFooService>()

.LifeStyle.Transient

.**IsFallback**()

);

Assert.**That**(container.Resolve<IFooService>(),

Is.InstanceOf<FallbackFooService>());

*// we may register our own component for FooService*

container.**Register**(

Component.For<IFooService>()

.ImplementedBy<FooService>()

);

Assert.**That**(container.Resolve<IFooService>(),

Is.InstanceOf<FooService>());

Since word isn’t perfect it happens from time to time that we want to overwrite component registration for some particular service. This usually happens because author of the installer doesn’t use fallback components. But don’t panic Castle Windsor allow us to overwrite service registrations using default components:

container.**Register**(

Component.For<IFooService>()

.ImplementedBy<FooService>());

Assert.**That**(container.Resolve<IFooService>(),

Is.InstanceOf<FooService>());

*// Without IsDefault() we*

*// would get an exception telling us that*

*// there is already component registered for IFooService*

*// interface.*

container.**Register**(

Component.For<IFooService>()

.ImplementedBy<DefaultFooService>()

.**IsDefault**()

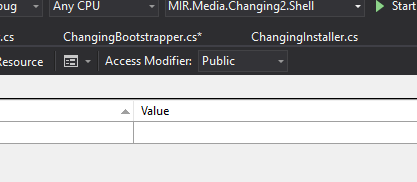
);

Assert.**That**(container.Resolve<IFooService>(),

Is.InstanceOf<DefaultFooService>());

##### Castle poprve vyzkouseno na projektu MediaPlayer2. Caliburn uz tam byl drive.

* Nejprve se postarame o Caliburn podle poznamek. (Start from bootstrapper using caliburn)
* Dale v Bootsrapperu vytvorime slozku Installers.
* Pridame projekty Common, Core a do nich tridy CoreAssemblyIdentificator a CommonAssemblyIdentifikator.
* V metode Bootstrapper.OnStartup si zaregistrujeme container, windowManagera a urcime z jakych assembly se bude instalovat. Postupovat podle kodu ve Skenovadle:
* Do Gui si pridame ResourcesFiles podle MojePoznamky -> JAZYK A CULTURE (nemusi se spustit generator staci zmenit AccesModifier na Public



Bootstrapper

##### Kod bootstrapperu

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Globalization;

using System.IO;

using System.Reflection;

using System.Security.AccessControl;

using System.Security.Principal;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Markup;

using Caliburn.Micro;

using Castle.DynamicProxy;

using Castle.MicroKernel.Registration;

using Castle.Windsor;

using Castle.Windsor.Configuration.Interpreters;

using Castle.Windsor.Installer;

using log4net.Config;

using Mediaresearch.Framework.Communication.Common;

using Mediaresearch.Framework.DataAccess.BLToolkit.Dao;

using Mediaresearch.Framework.DataAccess.Core.Auditable;

using Mediaresearch.Framework.Mapping;

using MIR.Common2.Gui;

using MIR.PrintStorage.Scanning2.Common;

using MIR.PrintStorage.Scanning2.Common.Auditable;

using MIR.PrintStorage.Scanning2.Common.Scan;

using MIR.PrintStorage.Scanning2.Core;

using MIR.PrintStorage.Scanning2.GUI.ViewModels;

using MIR.PrintStorage.Scanning2.Shell.Installers;

using MIR.PrintStorage.Scanning2.Shell.Properties;

using Action = System.Action;

using GuiResources = MIR.PrintStorage.Scanning2.GUI.GuiResources;

using ILog = log4net.ILog;

using LogManager = log4net.LogManager;

namespace MIR.PrintStorage.Scanning2.Shell.Bootstrapper

{

public class ScanningBootstrapper : Bootstrapper<MainViewModel>

{

private static readonly string m\_applicationDirectory = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

private static readonly ILog m\_log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);

private WindsorContainer m\_globalContainer;

protected override void Configure() // metoda caliburnu viz <https://d.docs.live.net/b22fb0fb09218bf0/Nielsen%20%20prace/Moje%20poznamky%20Nielsen/Caliburn/Start%20From%20Bootstrapper%20using%20Caliburn.docx>

{

ViewLocator.LocateForModel = (model, displayLocation, context) =>

{

var unproxiedModelType = ProxyUtil.GetUnproxiedType(model);

return ViewLocator.LocateForModelType(unproxiedModelType ?? model.GetType(), displayLocation, context);

};

}

protected override void OnStartup(object sender, StartupEventArgs e)

{

ConfigureLog4Net();

ConfigureLanguage();

m\_globalContainer = new WindsorContainer(new XmlInterpreter(Path.Combine(m\_applicationDirectory, "MIR.PrintStorage.Scanning2.Shell.Container.config"))); **// definuje xml soubor z ktereho se nacitaji property pro scannigConfiguration viz mojePoznamky => ContainerConfig**

var configuration = m\_globalContainer.Resolve<PrintStorageDbConfiguration>(); // resolvne z xml-ka tridu PrintStorageDbConfiguration

IScannigConfiguration scanningConfiguration = m\_globalContainer.Resolve<IScannigConfiguration>(); // resolvne z xml tridu scanningConfiguration a vlozi ji do promenne jako rozhrani

string path = scanningConfiguration.IconPath;

IWindowManager windowManager = new WindowManager();

m\_globalContainer.Register(Component.For<IWindowManager>().Instance(windowManager));

MainViewModel rootViewModel = null;

var assembly = Assembly.GetExecutingAssembly();

FileVersionInfo fvi = FileVersionInfo.GetVersionInfo(assembly.Location);

string version = $"{GuiResources.Scanning} {fvi.FileVersion}";

string appVersionInfo = $"{version} ({configuration.DefaultDataSource})";

Action initAction = () => // Tato cast se pusti asynchronne, mezitim se zobrazuje SplashScreen. Ten dostane tuto akci jako parametr konstruktoru. Tim vi kdy akce skonci a zavre se az po skonceni provadeni

{

m\_globalContainer.Register(Component.For<IWindsorContainer>().Instance(m\_globalContainer).LifestyleSingleton());

m\_globalContainer.Install(FromAssembly.This(), FromAssembly.Containing<ScanningCoreAssemblyIdentificator>()); //Spusti metody Install ze vsech Instalatoru

m\_globalContainer.Resolve<IMediaDataDaoSource>();

m\_globalContainer.Resolve<IPrintStorageDaoSource>();

var mappingConfiguratior = m\_globalContainer.Resolve<DependencyMappingConfigurator>();

mappingConfiguratior.Configure();

if (!Directory.Exists(ScanningConfiguration.TempFilesDirectory))

{

Directory.CreateDirectory(ScanningConfiguration.TempFilesDirectory); // Ve tride ScanningConfiguration (kterou si castle bere rovnou z xml souboru) je natvrdo definovano: public static string TempFilesDirectory { get; } = Path.Combine(Path.GetTempPath(), "Scanning");

}

ConfigureSubscriber();

ConfigureUser();

rootViewModel = m\_globalContainer.Resolve<MainViewModel>();

rootViewModel.AppVersionDescription = appVersionInfo;

};

SplashScreenViewModel splashScreen = new SplashScreenViewModel(initAction, version, path);

windowManager.ShowDialog(splashScreen); // Zobrazi se splash, ktery si vnitrne spusti initAction

windowManager.ShowDialog(rootViewModel); // Po skonceni splashe se spusti aplikace coz je taky dialog

Application.Shutdown();

m\_globalContainer.Dispose();

}

private void ConfigureUser() // Prihlasovani uzivatele

{

var userSource = m\_globalContainer.Resolve<IUserSource>(); // Kdyz Castlu rekneme, ze chceme rozhrani tak vytvori instanci tridy ktera toto rozhrani implementuje (vi podle registrace)

var auditableIdentityProvider = m\_globalContainer.Resolve<IAuditableIdentityProvider>();

var login = WindowsIdentity.GetCurrent().Name;

if (userSource.Login(login)) // UserSource je jednoducha trida, ktera se prez UserDao podiva do databaze, jestli tam existuje uzivatel login, coz je WindowsIdentity.GetCurrent().Name

{

var userId = userSource.GetUserId();

auditableIdentityProvider.SetAuditableIdentity(new AuditableIdentity(userId, login));

return true;

}

var message = new MessageBoxViewModel(GuiResources.UserNotFoundError, false, 3000); // Pokud uzivatele nenajde v databazi, tak nema opravneni spustit aplikaci, zobrazi se dialogove okno a ukonci se aplikace

windowManager.ShowDialog(message);

return false;

}

private static void ConfigureLog4Net() // Zapisovani do Logu viz <https://d.docs.live.net/b22fb0fb09218bf0/Nielsen%20%20prace/Moje%20poznamky%20Nielsen/Log.docx>

{

var logConfigFile = new FileInfo(Path.Combine(m\_applicationDirectory, "MIR.PrintStorage.Scanning2.Shell.log4net"));

XmlConfigurator.Configure(logConfigFile);

AppDomain.CurrentDomain.UnhandledException += CurrentDomainUnhandledException;

}

private void ConfigureLanguage()

{

var cultureInfo = CultureInfo.GetCultureInfo(Settings.Default.Language);

FrameworkElement.LanguageProperty.OverrideMetadata(typeof(FrameworkElement), new FrameworkPropertyMetadata(XmlLanguage.GetLanguage(cultureInfo.Name)));

Thread.CurrentThread.CurrentCulture = cultureInfo;

Thread.CurrentThread.CurrentUICulture = cultureInfo;

GuiResources.Culture = cultureInfo;

}

// kde se maji hledat requesty a responsy

private void ConfigureSubscriber()

{

var coreAssembly = typeof(ScanningCommonAssemblyIdentificator).Assembly;

var notificationProvider = m\_globalContainer.Resolve<INotificationsReceiversAssemblyProvider>();

notificationProvider.RegisterAssemblies(coreAssembly, coreAssembly);

var notificationsSubcriber = m\_globalContainer.Resolve<INotificationReceiverSubscriber>();

notificationsSubcriber.SubscribeAll();

var provider = m\_globalContainer.Resolve<IRequestsServiceActionsAssemblyProvider>();

provider.RegisterAssemblies(coreAssembly, typeof(ScanningCoreAssemblyIdentificator).Assembly);

var subscriber = m\_globalContainer.Resolve<IServiceActionSubscriber>();

subscriber.SubscribeAll();

}

// kde se maji hledat view

protected override IEnumerable<Assembly> SelectAssemblies()

{

return new[] {Assembly.GetAssembly(typeof(MainViewModel)), Assembly.GetAssembly(typeof(Common2GuiAssembyIdentificator))};

}

private static void CurrentDomainUnhandledException(object sender, UnhandledExceptionEventArgs e)

{

Log(e.IsTerminating, e.ExceptionObject, m\_log);

}

public static void Log(bool isTerminating, object exceptionObject, ILog logger)

{

var message = $"Unhandled exception in application (IsTerminating = {isTerminating})";

/\*

\* Why is UnhandledExceptionEventArgs.ExceptionObject of type Object and not Exception?

\* While not all languages support throwing non-Exception type exceptions, the CLR and IL allow for throwing any Object.

\* In general, throwing non-Exception types is discouraged because most developers do not expect this to occur,

\* and are not likely to catch the object.

\* On the other hand, developers who are overriding the unhandled exception logic may need to catch non-Exception objects, also.

\*/

if (exceptionObject is Exception ex)

{

logger.Fatal(message, ex);

}

else

{

logger.Fatal($"{message} : {exceptionObject}");

}

}

}

}

Vytvorime tridu Installer : IWindsorContainer ve slozce Shell.Installers a implementujeme metodu Install

##### Do slozky Bootstrapper si pridame dve tridy : InternalServicePublisher a NotificationProcessor

using System;

using Castle.Windsor;

using Mediaresearch.Framework.Communication.Common;

namespace Shell.Bootstrapper

{

public class InternalServicePublisher : ClientToServicePublisherBase

{

private readonly IWindsorContainer m\_kernel;

public InternalServicePublisher(IWindsorContainer kernel, IServiceActionSubscriber serviceActionSubscriber) : base(serviceActionSubscriber)

{

m\_kernel = kernel;

}

public override IServiceAction GetAction(Type actionType)

{

var action = m\_kernel.Resolve(actionType);

IServiceActionCallback actionCallback = (IServiceActionCallback)action;

actionCallback.ExecutionFinished += ActionCallbackOnExecutionFinished;

return (IServiceAction)action;

}

private void ActionCallbackOnExecutionFinished(object sender, EventArgs eventArgs)

{

IServiceActionCallback action = (IServiceActionCallback)sender;

action.ExecutionFinished -= ActionCallbackOnExecutionFinished;

m\_kernel.Release(action);

}

}

}

using System;

using Castle.MicroKernel.Registration;

using Castle.Windsor;

using Mediaresearch.Framework.Communication.Common;

namespace MIR.PrintStorage.Scanning2.Shell.Bootstrapper

{

public class ScanningNotificationProcesor : NotificationReceiverProcessor

{

public ScanningNotificationProcesor(INotificationsReceiversAssemblyProvider notificationsReceiversAssemblyProvider, IWindsorContainer container) : base(notificationsReceiversAssemblyProvider, container)

{

}

protected override void DoAfterSubscribe<TNotification>(Type receiverType)

{

if (!m\_container.Kernel.HasComponent(receiverType))

{

m\_container.Register(Component.For(receiverType).ImplementedBy(receiverType).LifestyleSingleton());

}

}

}

}

##### Vytvoreni WindsorConteineru a jeho konfigurace (Container.config) (Je tu I konfigurace pripojeni k databazi)

**Tim, ze containeru vyrobime konfiguracni xml soubor mu muzeme nastavit ruzne parametry**

Kod v Bootstrapperu: (Timto bude container vedet ze si ma brat property z xml souboru) :

m\_globalContainer = new WindsorContainer(new XmlInterpreter(Path.Combine(m\_applicationDirectory, "MIR.Media.Changing2.Shell.Container.config")));

Do Shellu se prida NewItem -> ApplicationConfigurationFile.

Bude se jmenovat MIR.Media.Changing2.Shell.Container.config

Muze byt na zacatku klidne prazdny ale v properties musi mit nastaveno **CopyAlways** jinak vyhazuje vyjimku : $exception {"Error processing node resource FileResource: [] []"}

Kdyz aplikaci rozchodime aby zobrazovala okno, muzeme si pridat do Commonu rozhrani. Ve skenovadle to bylo IscanningConfiguration. Bylo ve slozce :

*C:\Pool\Admosphere\src\MIR.PrintStorage\MIR.PrintStorage.Scanning2\MIR.PrintStorage.Scanning2.Common\Scan\IScannigConfiguration.cs*

Rozhrani jen definuje property ktere musi mit trida, implementujici toto rozhrani. Trida ktera ho implementuje je: ScanningConfiguration : IScannigConfiguration

*C:\Pool\Admosphere\src\MIR.PrintStorage\MIR.PrintStorage.Scanning2\MIR.PrintStorage.Scanning2.Shell\Installers\ScanningConfiguration.cs*

Tato trida musi mit vsechny property ktere definuje rozhrani a zaroven ma konstruktor ve kterem tyto property inicializuje. Tady probehne magie protože hodnoty se nikde nezadavaji. Tim, ze ma trida zavislosti na tyto property, si je Castle resolvne prave z configu.

napr.: public string IconPath { get; }

public ScanningConfiguration(int pdfConversionDpi, int pdfConversionQuality, string primaryScannerName,

bool primaryScannerDuplex, string secondScannerName, int scannerDpi, int pdfSourceOrder,

int scannerSourceOrder, int filesSourceOrder, bool useTwain2, bool useDoubleFeedDetection, string iconPath)

{

PdfConversionDpi = pdfConversionDpi;

PdfConversionQuality = pdfConversionQuality;

PrimaryScannerName = primaryScannerName;

PrimaryScannerDuplex = primaryScannerDuplex;

SecondScannerName = secondScannerName;

ScannerDpi = scannerDpi;

PdfSourceOrder = pdfSourceOrder;

ScannerSourceOrder = scannerSourceOrder;

FilesSourceOrder = filesSourceOrder;

UseTwain2 = useTwain2;

UseDoubleFeedDetection = useDoubleFeedDetection;

IconPath = iconPath;

}

Konfiguracni soubor ze skenovadla:

*C:\Pool\Admosphere\src\MIR.PrintStorage\MIR.PrintStorage.Scanning2\MIR.PrintStorage.Scanning2.Shell\MIR.PrintStorage.Scanning2.Shell.Container.config*

<?xml version="1.0" encoding="utf-8" ?>

<castle>

**Properties -> sem zadame hodnoty pro property z rozhrani**

<properties>

<mediaDataDbAlias>mediaDataDB</mediaDataDbAlias>

<mediaDataConnectionString>Data Source=stoupa;Initial Catalog=Mediadata3Auto;User ID=PrintStorageScanningUser; Pwd=5bTDwG-tJC;</mediaDataConnectionString>

<printStorageDbAlias>printStorageDB</printStorageDbAlias>

<printStorageConnectionString>Data Source=stoupa;Initial Catalog=PrintStorageAuto;User ID=PrintStorageScanningUser; Pwd=5bTDwG-tJC;</printStorageConnectionString>

<serverTimeZone>Central Europe Standard Time</serverTimeZone>

<pdfConversionDpi>120</pdfConversionDpi>

<pdfConversionQuality>50</pdfConversionQuality>

<primaryScannerName>Panasonic KV-S4085C KV-S4065C</primaryScannerName>

<primaryScannerDuplex>true</primaryScannerDuplex>

<secondScannerName>unknown</secondScannerName>

<scannerDpi>150</scannerDpi>

<useTwain2>true</useTwain2>

<useDoubleFeedDetection>false</useDoubleFeedDetection>

<pdfSourceOrder>2</pdfSourceOrder>

<scannerSourceOrder>1</scannerSourceOrder>

<filesSourceOrder>3</filesSourceOrder>

<iconPath>/MIR.PrintStorage.Scanning2.GUI;component/ScanningIcons/Scanning.png</iconPath>

</properties>

<components>

<component id="MediaDataDbConfiguration" type="MIR.PrintStorage.Scanning2.Shell.Installers.MediaDataDbConfiguration, MIR.PrintStorage.Scanning2.Shell">

<parameters>

<mediaDataDbAlias>#{mediaDataDbAlias}</mediaDataDbAlias>

<mediaDataConnectionString>#{mediaDataConnectionString}</mediaDataConnectionString>

<serverTimeZone>#{serverTimeZone}</serverTimeZone>

</parameters>

</component>

<component id="PrintStorageDbConfiguration" type="MIR.PrintStorage.Scanning2.Shell.Installers.PrintStorageDbConfiguration, MIR.PrintStorage.Scanning2.Shell">

<parameters>

<printStorageDbAlias>#{printStorageDbAlias}</printStorageDbAlias>

<printStorageConnectionString>#{printStorageConnectionString}</printStorageConnectionString>

<serverTimeZone>#{serverTimeZone}</serverTimeZone>

</parameters>

</component>

**Definice rozhrani IscanningConfiguration. Parametry uvnitr rikaji, ze napr. hodnotu pro promennou IconPath definovanou v rozhrani IscanningConfiguration vezmeme z properties castlu**

<component id="ScanningConfiguration" type="MIR.PrintStorage.Scanning2.Shell.Installers.ScanningConfiguration, MIR.PrintStorage.Scanning2.Shell"

service="MIR.PrintStorage.Scanning2.Common.Scan.IScannigConfiguration">

<parameters>

<pdfConversionDpi>#{pdfConversionDpi}</pdfConversionDpi>

<pdfConversionQuality>#{pdfConversionQuality}</pdfConversionQuality>

<primaryScannerName>#{primaryScannerName}</primaryScannerName>

<primaryScannerDuplex>#{primaryScannerDuplex}</primaryScannerDuplex>

<secondScannerName>#{secondScannerName}</secondScannerName>

<scannerDpi>#{scannerDpi}</scannerDpi>

<pdfSourceOrder>#{pdfSourceOrder}</pdfSourceOrder>

<scannerSourceOrder>#{scannerSourceOrder}</scannerSourceOrder>

<filesSourceOrder>#{filesSourceOrder}</filesSourceOrder>

<useTwain2>#{useTwain2}</useTwain2>

<useDoubleFeedDetection>#{useDoubleFeedDetection}</useDoubleFeedDetection>

<iconPath>#{iconPath}</iconPath>

</parameters>

</component>

</components>

</castle>

**Shrnuti:**

Kdyz chci aby byl container trochu chytry pridame mu rozhrani InecoConfig, Tridu ktera ho bude implementovat NecoConfig a xml konfiguracni soubor Container.config;

Kdyz chceme aby castle znal nejakou novou propertu at uz boolean, string nebo cokoli jineho, pridame si propertu do rozhrani, do tridy ktera ho implementuje, do properties v configu a do definice rozhrani v configu.

V installeru si musime rozhrani **Resolvnout** . : **IScannigConfiguration scanningConfiguration = container.Resolve<IScannigConfiguration>();**

**Mozna chyba:**

* Zkopiroval jsem cestu k souborum z Properties tridy ChangingConfiguration (pri zakladani zmenovadla) . Do cesty se mi ale misto tecek pridaly lomitka a Castlu se to nelibilo. Napsal ze nemuze vyrobit type from string .
* I kdyz jsem provedl uvedene kroky, string ktery jsem potreboval v Bootstrapperu byl porad null:

IChangingConfiguration changingConfiguration = m\_globalContainer.Resolve<IChangingConfiguration>();

var somethingUsefull = changingConfiguration.SomethingUsefull; **(porad null)**

Console.WriteLine(somethingUsefull);

Bylo to proto, ze :

Chybel konstruktor ve tride ChangingConfiguration se zavislosti na somethingUsefull. Ale je to stale null

Trida public class ScanningConfiguration : IscannigConfiguration musi mit propertu: public static string TempFilesDirectory { get; } = Path.Combine(Path.GetTempPath(), "Scanning"); d

V Bootstrapperu musi byt :

if (!Directory.Exists(ScanningConfiguration.TempFilesDirectory))

{

Directory.CreateDirectory(ScanningConfiguration.TempFilesDirectory);

}

Ted uz to funguje :D

##### Language

Dale jsem sel po cervene sviticich radcich a dopnoval to co aplikace potrebuje. Svitilo mi language var cultureInfo = CultureInfo.GetCultureInfo(Settings.Default.Language);

Ve slozce Shell je potreba rozklikat Settings.settings az dolu a doplnit :

public string Language

{

get

{

return ((string)(this["Language"]));

}

set

{

this["Language"] = value;

}

}

##### Pripojeni k databazi

Zacneme v uz znamem Configu. Na pripojeni k databazi potrebujeme mit v ContainerConfigu ConnectionString a komponentu PrintStorageDbConfiguration:

Definice ConnectionStringu

<properties>

<printStorageDbAlias>printStorageDB</printStorageDbAlias>

<printStorageConnectionString>Data Source=stoupa;Initial Catalog=PrintStorageAuto;User ID=PrintStorageScanningUser; Pwd=5bTDwG-tJC;</printStorageConnectionString>

<serverTimeZone>Central Europe Standard Time</serverTimeZone>

</properties>

Definice tridy PrintStorageDbConfiguration

<components>

<component id="PrintStorageDbConfiguration" type="MIR.PrintStorage.Scanning2.Shell.Installers.PrintStorageDbConfiguration, MIR.PrintStorage.Scanning2.Shell">

<parameters>

<printStorageDbAlias>#{printStorageDbAlias}</printStorageDbAlias>

<printStorageConnectionString>#{printStorageConnectionString}</printStorageConnectionString>

<serverTimeZone>#{serverTimeZone}</serverTimeZone>

</parameters>

</component>

</components>

Musime si pridat tridu :

*C:\Pool\Admosphere\src\MIR.PrintStorage\MIR.PrintStorage.Scanning2\MIR.PrintStorage.Scanning2.Shell\Installers\PrintStorageDbConfiguration.cs*

Ktera potrebuje*:*

*C:\Pool\Admosphere\src\MIR.PrintStorage\MIR.PrintStorage.Scanning2\MIR.PrintStorage.Scanning2.Shell\Installers\DbAccessConfigurationBase.cs*

##### Tridy PrintStorageDbConfiguration a DbAccessConfigurationBase

namespace MIR.PrintStorage.Scanning2.Shell.Installers

{

public class PrintStorageDbConfiguration : DbAccessConfigurationBase

{

protected override string ConnectionString => PrintStorageConnectionString;

public string PrintStorageDbAlias { get; }

public string PrintStorageConnectionString { get; }

public string ServerTimeZone { get; }

public PrintStorageDbConfiguration(string printStorageDbAlias, string printStorageConnectionString, string serverTimeZone)

{

PrintStorageDbAlias = printStorageDbAlias;

PrintStorageConnectionString = printStorageConnectionString;

ServerTimeZone = serverTimeZone;

}

}

}

using System.Text.RegularExpressions;

namespace MIR.PrintStorage.Scanning2.Shell.Installers

{

public abstract class DbAccessConfigurationBase

{

protected abstract string ConnectionString { get; }

public string DefaultDatabaseName

{

get

{

Match match = GetConnectionStringMatch();

if (match == null || match.Groups.Count < 3)

return null;

return match.Groups[2].Value;

}

}

public string DefaultDataSource

{

get

{

Match match = GetConnectionStringMatch();

if (match == null || match.Groups.Count < 3)

return null;

return $"{match.Groups[1].Value}/{match.Groups[2].Value}";

}

}

private Match GetConnectionStringMatch()

{

if (string.IsNullOrEmpty(ConnectionString))

return null;

Regex regex = new Regex(@".\*Data Source=([a-z0-9\w\.]\*);.\*Initial Catalog=([a-z0-9\w\.]\*);.\*", RegexOptions.IgnoreCase);

Match match = regex.Match(ConnectionString);

if (!match.Success)

return null;

return match;

}

}

}

##### Pokracovani pripojeni k Db:

* Ted uz v Installeru muzeme rict ze:

PrintStorageDbConfiguration configuration = container.Resolve<PrintStorageDbConfiguration>();

* Abychom mohli v installeru rict ze:

container.Register(Component.For<IParamsSource>().ImplementedBy<DatabaseParamsSource>().LifestyleSingleton());

* Budeme potrebovat rozhrani a tridu IparamsSource a DatabaseParamsSource

**Tato cast uz s Castlem nema moc spolecneho ale budu tady pokracovat at to neni na deseti mistech**

Vyjimka kterou mi vyhazovalo protoze jsem nemel vyresenou zavislost na Usera vyresil Mitroz : no bylo potreba trocu uklidit, user nesel kvuli tomu ze tam chybelo m\_globalContainer.Resolve<IMediaDataDaoSource>();

* Otestovat spojeni s databazi se da pomoci Unit testu -> MojePoznamky -> Unit Testy -> ConnectionTest
* Projekt jsem ulozil na plochu do slozky ChangingProjekty

##### Zavislosti v installerech => .DependsOn

**Pricing:**

container.Register(

Component.For<IPriceListProvider>().ImplementedBy<PricingServicePriceListProviderAdapter>().Named("pricingServicePriceListProvider")

.DependsOn(

Property.ForKey("container").Eq(container),

Property.ForKey("Name").Eq(Localisation.OfficialADMSPricing),

Property.ForKey("pricingServiceKey").Eq(PricingServiceClientInstaller.PricingServiceKey),

Property.ForKey("pricingEvaluatorKey").Eq("pricingServiceEvaluatorAdapter")

)

);

**Nebo treba:**

Property.ForKey("pricingService").Is(PricingServiceClientInstaller.PricingServiceKey),

Property.ForKey("batchSize").Eq(container.Resolve<Configuration.Properties>().ItemsCalculationBatch)

**Kdyz chceme rict kteraze konkretne komponenta se ma resolvnout :**

Component.For<IMessageLoader>().ImplementedBy<RadioMessageLoaderViewModel>().Named("radioMessageLoaderViewModel")

.DependsOn(

Property.ForKey("loader").Is("radioMessageLoader")

)

);

public class RadioMessageLoaderViewModel : MessageLoaderViewModelBase<RadioMessageLoader>

{

public RadioMessageLoaderViewModel(RadioMessageLoader loader, ITaskQueue taskQueue) : base(loader, taskQueue)

{

DisplayName = Localisation.Radio;

}

}

}

##### Registrace z xml

Pricing:

container.Install(Castle.Windsor.Installer.Configuration.FromXmlFile(@"MIR.Pricing.Container.config"));

Scanning:

private WindsorContainer m\_globalContainer;

m\_globalContainer = new WindsorContainer(new XmlInterpreter(Path.Combine(m\_applicationDirectory, "MIR.PrintStorage.Scanning2.Shell.Container.config")));

m\_globalContainer.Register(Component.For<IWindsorContainer>().Instance(m\_globalContainer).LifestyleSingleton());

##### .Named()

Pricing:

container.Register(

Component.For<ITaskQueue>().ImplementedBy<TaskQueue>().Named("mainTaskQueue").LifeStyle.Singleton

);

##### Jak rovnou setnout propertu z xml ka pri registraci v installeru

Propertu musime mit na nejakem jinem objektu . Typicky v properties (Pricing). Je normalne definovana v xml ku a pak ji pri registraci resolvneme a vlozime do componenty.

##### Container pouzitelny I mimo Shell

namespace MIR.Pricing.Container

{

public sealed class Container : WindsorContainer

{

static Container m\_current;

public static Container Current

{

get

{

if (m\_current == null)

{

Configure();

}

return m\_current;

}

}

private static void Configure()

{

lock (typeof(Container))

{

m\_current = new Container();

}

}

}

}

V pricingu je container v Mir.Pricing pak se da v teto assembly pouzit :

Container.Container.Current.Install(new CommonInstaller());

##### Forward

<http://mikehadlow.blogspot.com/2010/02/10-advanced-windsor-tricks-11.html>

**Mejme tri rozhrani**

public interface IThing

{

string SayHello(string name);

}

public interface IWidget

{

double Calculate(double a, double b);

}

public interface IWonder

{

void DoesNothing();

}

**Mame tridu, ktera implementuje vice rozhrani.**

public class SrpViolator : IThing, IWidget, IWonder

{

public string SayHello(string name)

{

return string.Format("Hello {0} from SrpViolator", name);

}

public double Calculate(double a, double b)

{

return Math.Pow(a, b);

}

public void DoesNothing()

{

Console.WriteLine("Doing nothing");

}

}

**Pri registraci do containeru muzeme zaregistrovat vsechny rozhrani ktere jsou implementovany jednou tridou**

var container = new WindsorContainer()

.Register(

Component

.For<IThing>()

.Forward<IWidget>()

.Forward<IWonder>()

.ImplementedBy<SrpViolator>()

);

***Nebo takto:***

var container = new WindsorContainer()

.Register(

Component

.For<IThing, IWidget, IWonder>()

.ImplementedBy<SrpViolator>()

);

**Now we can resolve each interface independently, but behind the scenes they are all share the same implementation:**

var thing = container.Resolve<IThing>();

Console.WriteLine(thing.SayHello("Krzysztof"));

var widget = container.Resolve<IWidget>();

Console.WriteLine("The answer is {0}", widget.Calculate(2, 3));

var wonder = container.Resolve<IWonder>();

wonder.DoesNothing();

Which outputs:

Hello Krzysztof from SrpViolator

The answer is 8

Doing nothing

Since SrpViolator is registered without specifying a lifestyle, it will have the default lifestyle: **singleton**. That means that ‘thing’, ‘widget’ and ‘wonder’ from the code snippet above **are all** **the same instance of SrpViolator**.

Type forwarding can be very useful, but note that the [Single Responsibility Principle](http://en.wikipedia.org/wiki/Single_responsibility_principle) means that it’s generally considered bad practice to have a single class play many different roles.

##### Resolvnuti pojmenovane (named komponenty)

container.Resolve<ICreativeItemLoader>("imageCreativeLoader")

##### .Eq( new [])

Tady pouzity List<>

container.Register(Component.For<FileSystemAccessManagerComposite>().Named("prahafileSystemAccessManagerComposite")

.DependsOn(

Property.ForKey("internalManagers").Eq(new List<IFileSystemAccessManager>

{

container.Resolve<FileSystemAccessManager>("localFileSystemAccessManager"),

container.Resolve<FileSystemAccessManager>("prahaFileSystemAccessManager")

})

));

##### Registrace v container configu vs registrace v installeru

<components>

<component id="hradecLocation"

service="Mediaresearch.Framework.Utilities.Net.ILocation, Mediaresearch.Framework.Utilities"

type="MIR.Media.Coding.Location.Location, MIR.Media.Coding">

<parameters>

<ipAddressInSubnetString>#{hradecSubnetIpAddress}</ipAddressInSubnetString>

<identificator>JH</identificator>

</parameters>

</component>

</components>

**=**

container.Register(Component.For<ILocation>().ImplementedBy<Location.Location>().Named("hradecLocation")

.DependsOn(

Property.ForKey("ipAddressInSubnetString").Eq(fileSystemonfiguration.HradecSubnetIpAddress),

Property.ForKey("identificator").Eq("JH")

));

##### Registrace generickeho typu v containerConfigu vs installer

<components>

<component id="locationResolver"

type="Mediaresearch.Framework.Utilities.Net.ByInterNetworkIpAddressLocationResolver`1[[MIR.Media.Coding.Location.Location, MIR.Media.Coding]], Mediaresearch.Framework.Utilities">

<parameters>

<locations>

<array>

<item>${hradecLocation}</item>

</array>

</locations>

</parameters>

</component>

</components>

**Vs:**

container.Register(Component.For<ByInterNetworkIpAddressLocationResolver<Location.Location>>().Named("locationResolver")

.DependsOn(

Property.ForKey("locations").Eq(new [] {container.Resolve<ILocation>("hradecLocation")})

));

##### Property.ForKey() vs OnComponent()

[‎19.‎09.‎2018 15:11]  Filip Čálek:

no kazdopadne je rozdil mezi Property.ForKey a Dependency.OnComponent<>()

vsude se cpe Property.ForKey, coz funguje, ale je to vlastne spatne

[‎19.‎09.‎2018 15:12]  Filip Čálek:

Dependency.OnComponent<>()  rika, ze se jedna o zavislost v ctoru a castle ji pak nikde jinde nehleda. Property.ForKey hleda primarne propertu s public setterem a teprve pak ctor

a pak mam jeste pocit, ze Dependency.OnComponent<>() vyhazuje vyjimku, pokud tu zavislost nenajde, ale nejsem si tim ted uplne jistej

**Priklady:**

1)

.DependsOn(

Dependency.OnComponent(typeof(PriceCalculationDialogViewModel), "priceCalculationDialogViewModel"),

2)

Simbios:

container.Register(Component.For<IDaoSource>().Forward<IShodanDaoSource>().ImplementedBy<DependencyDaoSource>().DependsOn(Dependency.OnComponent("entityDaoFactory", typeof (EntityDaoFactory))));

##### Dependency.OnValue

Chci predat jen string, coz neni komponenta:

<component id="focusUsePriceOnPricingServiceCalculationMethodAspect"

service="Castle.DynamicProxy.IInterceptor, Castle.Core"

type="MIR.Media.Coding.Core.WorkFlow.Aspects.Focus.SetFocusAspect, MIR.Media.Coding.Core">

<parameters>

<elementPath>UsePricingService</elementPath>

</parameters>

</component>

container.Register(Component.For<IInterceptor>().ImplementedBy<SetFocusAspect>().Named("focusUsePriceOnPricingServiceCalculationMethodAspect")

.DependsOn(

Dependency.OnValue(typeof(string), "UsePricingService")

));

##### Registrace Interceptoru

1. Pricing.config zaregistrovat interceptory v PricingInstalleru:

<component id="pricingServiceCalculationMethod"

service="MIR.Media.Coding.Core.Utilities.Dialogs.PriceCalculationDialog.IPriceCalculationMethod, MIR.Media.Coding.Core"

type="MIR.Media.Coding.Core.Utilities.Dialogs.PriceCalculationDialog.PricingService.PricingServiceMethodViewModel, MIR.Media.Coding.Core"

lifestyle="transient">

<parameters>

<priceListsIdHolder>${priceListsIdHolder}</priceListsIdHolder>

</parameters>

<interceptors selector="${pricingServiceCalculationMethodInterceptorSelector}">

<interceptor>${focusUsePriceOnPricingServiceCalculationMethodAspect}</interceptor>

</interceptors>

</component>

2) container.Register(Component.For<IPriceCalculationMethod>().ImplementedBy<PricingServiceMethodViewModel>().Named("pricingServiceCalculationMethod").LifestyleTransient()

.DependsOn(

Dependency.OnComponent(typeof(PriceListsIdHolder), "priceListsIdHolder")

).Interceptors("focusUsePriceOnPricingServiceCalculationMethodAspect"));

**interceptorSelector reseny v mailu s Filipem**

V kodovadle je registrace interceptoru takto:

Component.For<IMediaMessageControl>().ImplementedBy<PressMediaMessageControlViewModel>().Named("pressMediaMessageControlViewModel")

.DependsOn(

Dependency.OnComponent(typeof(PriceCalculationDialogViewModel), PricingComponents.PriceCalculationDialogViewModel),

Property.ForKey("entityDaoFactory").Is("entityDaoFactory"),

Property.ForKey("taskQueue").Is("mainTaskQueue"),

Property.ForKey("MessagingService").Is("messagingService"),

Property.ForKey("MediaMessageSpecificVisualComponents").Eq(new List<IMediaMessageSpecificComponent>

{

container.Resolve<IMediaMessageSpecificComponent>("sheetComponent")

}),

Property.ForKey("SheetViewModel").Is("sheetComponent"),

Property.ForKey("pressPriceableItemLoader").Is("pressPriceableItemLoader"),

Property.ForKey("MediaMessageSpecificComponents").Eq(new List<IMediaMessageSpecificComponent>())

)

.DynamicParameters((k, p) => p["placementDefaultId"] = paramSource.NespecPlacementId)

.Interceptors(

InterceptorReference.ForKey("codingPlausibilitySetToSureActivateContentPageInterceptor")

)

.SelectedWith(pressMediaMessageControlInterceptorSelector)

.Anywhere,

##### Dependency.OnComponentCollection

V installerech by nemely byt resolvy. Kolekci pridame takto:

container.Register(Component.For<MethodNameRegexToInterceptorsMapping>().Named("calculatePricepricingServiceCalculationMethodInterceptingMapping")

.DependsOn(Dependency.OnComponentCollection(typeof(List<IInterceptor>) ,"focusUsePriceOnPricingServiceCalculationMethodAspect")));

##### Named()

ahoj,

ano rikas to spravne a taky mi na prvni pohled prijde, ze je na vetsine mist named rozkopirovano zbytecne. Jen doplnim, pokud se jedna o jednu jedinou implementaci, pak neni ani potreba zavislost definovat a container ji doda automaticky.

F.

Jen pro jistotu se zeptam:

Mam komponentu PriceListHolder.  Protože není schovana za zadne rozhrani, nepotrebuje zadne Named() pojmenovani a když ji budu chtit v jiné komponente pouzit tak reknu jen :

Dependency.OnComponent(typeof(PriceListsIdHolder), typeof(PriceListsIdHolder))

Container jinou komponentu typu PriceListIdHolder nemá, takze dostanu tu spravnou.  Je to v poradku?  V podstate to znamena, ze zadna komponenta, která není schovana za rozhranim nepotrebuje mit Named().  A ze jich je..

##### CollectionResolver

<https://malvinly.com/2012/02/27/castle-windsor-resolving-collections/>

**m\_globalContainer.Kernel.Resolver.AddSubResolver(new CollectionResolver(m\_globalContainer.Kernel));**

kdyz pak chci resolvnout typ ktery ma jen kolekci<T> tak mi ho Castle da

[‎26.‎09.‎2018 14:58]  Peter Hlavenka:

prosim Te, da se napsat jinak toto : Dependency.OnComponentCollection(typeof(IEnumerable<IMessageEntityLoader>), container.ResolveAll<IMessageEntityLoader>())      - chci se zbavit ResolveAll

[‎26.‎09.‎2018 15:02]  Filip Čálek:

pokud nepotrebujes specifikovat o jake se jedna a chces vsechny, tak ti staci mit v ctoru jen **IEnumerable<IMessageEntityLoader> zavislost a container to da**

jen jeste musis mit aktivovanej collectionresolver

[‎26.‎09.‎2018 15:03]  Peter Hlavenka:

to jeste neznam. Zkusim to najit v SimBiosu

[‎26.‎09.‎2018 15:03]  Filip Čálek:

otevri si SimAdmin a vyhledej si pouziti CollectionResolver

##### Collection resolver

Sometime we want to get all components that provide given service. For example we may try to implement message filtering component and we want to get all components that implement IFilter interface. We may achieve this easily by using Castle Windsor CollectionResolver:

*// register CollectionResolver in the container:*

container.Kernel.Resolver.**AddSubResolver**(

**new** **CollectionResolver**(container.Kernel));

*// demo:*

**public** **interface** **IFilter** {

**bool** **IsAllowed**(**string** message);

}

**public** **class** **MessageFilterService** {

**private** ICollection<IFilter> \_filters;

**public** **MessageFilterService**(ICollection<IFilter> filters) {

**this**.\_filters = filters;

}

...

}

container.**Register**(

Component.For<MessageFilterService>().LifeStyle.Transient,

Component.For<IFilter>().ImplementedBy<RejectBazWordFilter>(),

Component.For<IFilter>().ImplementedBy<FooOrBazFilter>()

);

MessageFilterService service = container.Resolve<MessageFilterService>();

service.**IsAllowed**("foo");

Since registering CollectionResolver requires a bit of interaction with a container it is advisable to wrap that logic into custom facility:

**public** **class** **ResolveCollectionsFacility** : **AbstractFacility** {

**protected** **override** **void** **Init**() {

Kernel.Resolver.**AddSubResolver**(**new** **CollectionResolver**(Kernel));

}

}

*// then use:*

// container.AddFacility<ResolveCollectionsFacility>();

##### UsingFactoryMethod()

**Resolvuje se az kdyz si o tu zavislost nekdo rekne**

Mam v pringu v registraci toto:

Container.Register(Component.For<ITvMediumDao>().UsingFactoryMethod((k,c) => k.Resolve<ITvMediumDao>()));

Kdyz se podivam na rozhrani ITvMedium vidim ze je to factory:

namespace MIR.Entities.MediaData.Media

{

[DaoFactory(DaoType = typeof(TvMediumDao), ImplementorType = typeof(TvMediumSkeleton), InheritedTypes = new[] { typeof(Medium), typeof(TvMedium) })]

public interface ITvMedium : IMedium

{

short TVStorageChannelId { get; set; }

}

[TableName(Owner = "Media", Name = "TvMedium")]

public abstract class TvMedium : DatabaseEntityShortKey<TvMedium>

{

public abstract short TVStorageChannelId { get; set; }

}

}

##### AsFactory

**V Kodovadle mame rozhrani:**

public interface IHistoryDialogFactory

{

MessageHistoryDialogViewModel **Create**(int messageId, MediaType.Values mediaTypeValue);

void **Release**(MessageHistoryDialogViewModel model);

}

**Toto rozhrani se registruje v Castlu jako factory:**

Component.For<IHistoryDialogFactory>().AsFactory(),

**A zaroven registrujeme tridu, kterou bude Factory vracet jako Transient:**

Component.For<MessageHistoryDialogViewModel>().LifeStyle.Transient,

**Kdyz chci dostat MessageHistoryDialogViewModel ktery vraci metoda Create, musim si nekde resolvnout factory, (konstruktor nebo property injection) a na instanci factory zavolat Create() :**

MessageHistoryDialogViewModel dialog = m\_historyDialogFactory.Create(MediaMessage.Id, (MediaType.Values)MediaMessage.MediaTypeId); //messageId a mediaType jsou zavislosti ktere si zada konstruktor VM a zaroven parametry ktere chce metoda Create();

**V com je vyhoda resolvnuti instance pomoci Factory netusim :D**

##### Property Injection

Castle umi resolvnout public property, ktere maji public setter. Trida se ale musi resolvovat v kontejneru. Pokud je trida vytvorena pomoci new(), zadne zavislosti ji Castle nedoda.

##### Resolvnuti podle Name nebo podle typu

var mediaMessageControlConductor = m\_container.Resolve<IMediaMessageControlConductor>() as MediaMessageControlConductorViewModel;

var mediaMessageControlConductor = m\_container.Resolve<IMediaMessageControlConductor>(“name”);

##### Resolvnuti EnumTableValues

private readonly IEnumTableValues<FastenType, byte> m\_fastenTypes;

public GetFastenTypesAction(IAuditableIdentityProvider auditableIdentityProvider, IEnumTableValues<FastenType, byte> fastenTypes)

: base(auditableIdentityProvider)

{

m\_fastenTypes = fastenTypes;

}

**Pricing:**

var mediaTypes = m\_daoSource.GetEnumTableByEntityType<Entities.MediaData.Media.MediaType, short>()

**Pricing: se da predelat tak, ze do konstruktoru pridam zavislost na** **:**

private readonly IEnumTableValues<Entities.MediaData.Media.MediaType, short> m\_mediaTypes;

**Konstruktor:**

, IEnumTableValues<Entities.MediaData.Media.MediaType, short> mediaTypes,