|  |  |
| --- | --- |
| Title: |  |
| Attendees |  |
| Absent |  |

# Index

Contents

[1 Index 1](#_Toc385313891)

[2 About the introspect Framework 2](#_Toc385313892)

[2.1 Why develop yet another application framework 2](#_Toc385313893)

[2.2 Introspect Core Values 3](#_Toc385313894)

[3 Introspect Licence 3](#_Toc385313895)

[4 Introspect Domain Concepts 3](#_Toc385313896)

[4.1 Domain Object 4](#_Toc385313897)

[4.1.1 Domain Object naming 4](#_Toc385313898)

[4.1.2 Domain Object Constructor 4](#_Toc385313899)

[4.1.3 Domain Object Presentation 4](#_Toc385313900)

[4.2 Domain Object Properties 4](#_Toc385313901)

[4.3 Domain Object Methods 4](#_Toc385313902)

[4.4 Service Objects 4](#_Toc385313903)

[4.5 Front End Service Object 5](#_Toc385313904)

[4.5.1 Front End Service Object Name 5](#_Toc385313905)

[4.5.2 Front End Service Object Constructor 5](#_Toc385313906)

[4.6 Back end Service Object 5](#_Toc385313907)

[5 Adding object behaviour 5](#_Toc385313908)

[5.1 Title 6](#_Toc385313909)

[5.1.1 Visible In Form 6](#_Toc385313910)

[5.1.2 Visible In Table 6](#_Toc385313911)

[5.1.3 Enabled 6](#_Toc385313912)

[5.1.4 Order 6](#_Toc385313913)

[5.1.5 Parameter Factory 6](#_Toc385313914)

[5.1.6 Icons 7](#_Toc385313915)

[5.1.7 FieldMode 7](#_Toc385313916)

[5.1.8 Validation 7](#_Toc385313917)

[5.1.9 MultiLanguage 7](#_Toc385313918)

[6 Introspect Framework Concepts 7](#_Toc385313919)

[6.1 Dependency injection 7](#_Toc385313920)

[6.2 Hexagon architecture 7](#_Toc385313921)

[6.3 Providers 7](#_Toc385313922)

[6.4 DomainProvider 7](#_Toc385313923)

[6.5 ValidationProvider 7](#_Toc385313924)

[6.6 UserInterfaceProvider 7](#_Toc385313925)

[6.7 AuthorizationProvider 7](#_Toc385313926)

[6.8 PathProvider 7](#_Toc385313927)

[6.9 LanguageProvider 7](#_Toc385313928)

[7 Getting Started 7](#_Toc385313929)

[7.1 Integrated Development Environment(IDE) 8](#_Toc385313930)

[7.2 Create your first domain object's (TODO youtube video) 8](#_Toc385313931)

[7.3 Create a command line application 8](#_Toc385313932)

[7.4 Create a command desktop application 8](#_Toc385313933)

[7.5 Create a command mobile application 8](#_Toc385313934)

[7.6 Create a command web application 8](#_Toc385313935)

[7.7 Create a command soap application 8](#_Toc385313936)

[7.8 Create a command restfull XML application 8](#_Toc385313937)

[7.9 Icons (TODO youtube video) 8](#_Toc385313938)

[7.10 Language (TODO youtube video) 8](#_Toc385313939)

[7.11 Validation (TODO youtube video) 8](#_Toc385313940)

[7.12 Authorization (TODO youtube video) 8](#_Toc385313941)

[7.13 Reports 8](#_Toc385313942)

# About the introspect Framework

Introspect is a light weight [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [Application Framework](http://en.wikipedia.org/wiki/Application_framework). It provides a framework with basic functionalities to develop applications for the desktop (based on [Swing](http://en.wikipedia.org/wiki/Swing_(Java))), mobile devices (based on [Android](http://en.wikipedia.org/wiki/Android_(operating_system))), the web (based on [Vaadin](http://en.wikipedia.org/wiki/Vaadin)) and [command line](http://en.wikipedia.org/wiki/Command-line_interface).

## Why develop yet another application framework

Almost everyday new libraries, frameworks and tools are being released by the developer community, many of which reinvent the wheel.

This is called the ‘Yet Another Framework Syndrome’ (YAFS), or in more general terms ‘Not Invented Here’ (NIH). While innovation is something we should all welcome, YAFS can lead to confusion and frustration for users because there’s a big risk of it introducing more noise than value.

So why develop a new framework while there are so many Java application frameworks out there?

Answers:

* Because its fun (see [Dutch article](http://www.sdn.nl/SDN/Artikelen/tabid/58/view/View/ArticleID/524/Just-because-its-fun.aspx))
* The journey of developing yet another framework has thought me more than I could have learned implementing an existing framework. Specifically on how other frameworks solved issues that I run into.
* I love the thoughts behind the [Naked Objects Framework](http://nakedobjects.codeplex.com/) (for [.net](http://en.wikipedia.org/wiki/.NET_Framework)) and the [Apache Isis Framework](http://isis.apache.org/) (for [Java](http://en.wikipedia.org/wiki/Java_(programming_language))). But looking at the Apache Isis Framework, there are several things I personally dislike which I tried to do better in the Introspect framework. Note that what I dislike does not mean I thinks is wrong in all cases!
  + The domain objects and service objects (often repository objects) usually extend a convenience class that contains methods to interact with the framework/ object container. Extending such a class is not mandatory, because you can implement these methods in your objects, but to me this ignores the principle of "Naked objects" or [POJO’s](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) .
  + Apache Isis’s uses (depends on) Maven. Maven has its pros (managing dependencies, and standardizing the build process), but is also famous for its cons (difficult to configure, steep learning curve, etc, etc).
  + I personally dislike the way Apache Isis manages the editing of objects. This is very tightly linked to its persistence framework. I however believe that objects do not necessarily need to be persisted after its been edited. I prefer to inverse the solution: domain objects can be passed to a method as a method parameter. This method parameter can be edited by a user before a method is executed (depending on how the method is [annotated](#_Method_Execution_Mode)) the domain object/ method parameter may then be handled by a [back end service](#_Back_end_Service) like a persistence service (or not at all).

## Introspect Core Values

Introspect is designed around the following core values:

* Based on the [Naked Objects Design Pattern](http://en.wikipedia.org/wiki/Naked_objects)
  + All [business logic](http://en.wikipedia.org/wiki/Business_logic) should be encapsulated onto the [domain objects](http://en.wikipedia.org/wiki/Business_object_(computer_science)). This principle is not unique to naked objects: it is just a strong commitment to [encapsulation](http://en.wikipedia.org/wiki/Information_hiding).
  + The [user interface](http://en.wikipedia.org/wiki/User_interface) should be a direct representation of the domain objects, with all user actions consisting, explicitly, of creating or retrieving domain objects and/or invoking [methods](http://en.wikipedia.org/wiki/Method_(computer_science)) on those objects. This principle is also not unique to naked objects: it is just a specific interpretation of an [object-oriented user interface (OOUI)](http://en.wikipedia.org/wiki/Object-oriented_user_interface).
  + The user interface should be created 100% automatically from the definition of the domain objects. Introspect uses [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science)) instead of code generation.
* Provide a good structure for applications
  + Enforce [separation of concerns](http://en.wikipedia.org/wiki/Separation_of_concerns)
  + The introspect framework exists of one static [Introspect](#_The_Introspect_class) class and a few basic interfaces ([providers](#_Providers)) that can have multiple different implementations.
* Lightweight
  + The Introspect framework should only be a few kilobytes in size.
* Simple to configure
* Configure by exception: Reasonable defaults wherever possible
  + No configuration files: There are 2 things that need to be configured. Both are configured in the code of the application:
    - How the framework is [initialized](#_Initializing_the_framework).
    - How the domain objects and service objects [behave](#_Adding_object_behaviour) (how the domain needs to handled by Introspect framework)
* No dependencies with an [Integrated Development Environment](http://en.wikipedia.org/wiki/Integrated_development_environment) or [build tool](http://en.wikipedia.org/wiki/Build_automation).

## Introspect Licence

Introspect is an [open source](http://en.wikipedia.org/wiki/Open_source) project under the [GNU Lesser General Public Licence](http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License).

The licence can be found [here](https://github.com/ntenhoeve/Introspect-Framework/blob/master/LICENSE).

# Introspect Domain Concepts

This section describes the key concepts in a Introspect application, and how they fit together.

## Domain Object

Domain objects represent entities, the nouns of the domain. If your application domain is a sales application it’s likely that your domain model contains domain objects such as: customers, products and orders.

You can specify certain things about both the behaviour and presentation of domain objects by adding specific attributes or methods (see chapter [Adding object behaviour](#_Adding_object_behaviour)).

### Domain Object Naming

Domain objects names are nouns, such as customer, product and order. They basically describe the things that are in your application domain.

### Domain Object Construction

The principle of “naked objects” is that any ['Plain Old Java Object'](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) (POJO) can function as a domain object. In other words: a domain class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

Extending domain classes is allowed.

The domain class must have a public [default constructor](http://en.wikipedia.org/wiki/Default_constructor) (with no arguments). This allows easy instantiation within the framework.

### Domain Object Presentation

An Introspect Interface can display domain objects in 3 ways:

(TODO picture) (TODO picture) (TODO picture)

As a form As a field in a form As a row in a table

## Domain Object Properties

Domain objects have [properties](http://en.wikipedia.org/wiki/Property_(programming)). Properties are a special type of class members and are an intermediate between a [field](http://en.wikipedia.org/wiki/Field_(computer_science)) and a method. Properties are read and sometimes written.

Introspect uses the same naming convention as [JavaBeans](http://en.wikipedia.org/wiki/JavaBeans)

TODO get

TODO is

TODO set

TODO note that the introspect framework only recognizes public set methods with one parameter with the same type as being used for the getter

TODO read only properties such as composit property

TODO note that the introspect framework only recognizes public properties

Here is an example of a domain class customer that has 2 properties

Public class Customer() {

Private String name;

Private Boolean male;

Public string getName() {

return name;

}

Public void setName(String newName) {

name=new name

}

Public string isMale() {

return male;

}

Public void setMale(boolean isMale) {

male=isMale;

}

## Property Types

## Domain Object Methods

TODO: explain that introspect only accepts methods with one parameter. You can use methods with multiple parameters, but these methods can only be used internally (calling from code) and won’t be visible for the outside world.

## Service Objects

Services perform three roles in a Naked Objects application:

* To provide methods for creating and retrieving domain objects where the user does not have an existing object to navigate from. Services that perform this role are often referred to as 'repositories'. (Some designers prefer to draw a distinction between a 'factory' service, which has responsibility for creating domain objects, and a 'repository' service for retrieving them. If you prefer to maintain this distinction, then they may be implemented in two separate services. However, there is merit in keeping the ideas together, and we tend to refer to the combination as a 'repository'. For more information see Section 2.2.5, “Repository”.
* To provide a bridge to external functionality. For more information see Section 2.2.6, “External or System service”.
* To provide functionality that is to be shared by multiple classes of domain objects which do not necessarily have any common superclass. This is achieved through the concept of contributed actions whereby methods are written on a service but appear to the user as actions on a domain object. For more information see Section 2.2.7, “Contributed action”.

In whichever of these roles, a service is just an ordinary POCO class but without any state - just methods. Just like a domain object it does not have to inherit from any special class, nor implement any interface, nor include any specific attribute. What makes it a service is simply that it is registered as a service in the Services property of the Run class. Registering the service serves two purposes:

* It makes the service's actions available to the user
* It instructs the Naked Objects framework to inject that service into any domain object that needs access to it. See [Dependency Injection](#_Dependency_Injection).

A service may provide multiple methods any of which may appear as user actions - following the same rules as for actions on a domain object.

Warning: adding properties to a service - for example to cache retrieved objects in memory - can result in unreliable behaviour and hard-to-diagnose errors.

TODO service objects can use other service objects. This is eased with [dependency injection](#_Dependency_injection).

TODO: explain that introspect only accepts methods with one parameter. You can use methods with multiple paramaters, but these methods can only be used internally (calling from code) and won’t be visible for the outside world.

## Front End Service Object

TODO explain what it is

### Front End Service Object Naming

Front end service objects are normally named to the domain object that they service (i.e. CustomerService).

### Front End Service Object Construction

The principle of “naked objects” is that any ['Plain Old Java Object'](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) (POJO) can function as a domain object. In other words: a service class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

Extending service classes is allowed.

The service class must have a public [default constructor](http://en.wikipedia.org/wiki/Default_constructor) (with no arguments). This allows easy instantiation within the framework.

### Front End Service Object Presentation

The Introspect Interface will display the public methods as menu items.

## Back end Service Object

TODO explain what it is

### Back End Service Object Naming

Back end service objects are normally named to ….

Examples:

TODO

GoogleMapsHttpClient

MailClient

CustomerReporitory

### Back End Service Object Construction

The principle of “naked objects” is that any ['Plain Old Java Object'](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) (POJO) can function as a domain object. In other words: a service class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

Extending service classes is allowed.

The service class must have a public [default constructor](http://en.wikipedia.org/wiki/Default_constructor) (with no arguments). This allows easy instantiation within the framework.

### Back End Service Object Presentation

The Introspect Interface will display the public methods as menu items.

# Adding object behaviour

This section is concerned with adding to, or modifying, the behaviour of domain objects and service objects, in order to build a richer application.

(TODO verify if we missed a chapter by inspecting ClassInfo and MethodInfo and PropertyInfo)

## Title

Domain objects and service objects have a title which is displayed in the user interface. They use their class name as a title by default but you can override the title.

(TODO title annotation)

(TODO title method)

(TODO toString method)

(TODO titlebuilder)

For domain objects it makes sence to override the title so that the domain object can be better identified. For example: the default title for a customer object will be “Customer”. It makes since to override the title with the name of the customer, so that the customer object can be better identified in the user interface.

## Visible In Form

(TODO specific on form)Public properties and methods are visible by default but they can be hidden.

Methods that are not public are not visible or access able.

(TODO visible annotation)

(TODO visible method)

## Visible In Table

(TODO specific on table)Public properties and methods are visible by default but they can be hidden.

(TODO properties of type collection are not displayed in a table)

Methods that are not public are not visible or access able.

(TODO visible annotation)

(TODO visible method)

(TODO refer to Authorization Provider)

## Enabled

Properties and methods are enabled by default but they can be disabled.

* Public properties that are disabled are visible to the user but can’t be edited.
* Public methods that are disabled are visible to the user but can’t be executed.

(TODO annotation)

(TODO method)

(TODO refer to Authorization Provider)

## Order in Form

(TODO description what this does)

(TODO annotation)

(TODO method)

## Order in Table

(TODO description what this does)

(TODO annotation)

(TODO method)

## Method Execution Mode

(TODO description what this does)

(TODO annotation)

## Parameter Factory

(TODO description what this does)

(TODO annotation)

(TODO method)

## Icons

(TODO description what this does)

(TODO annotation)

(TODO method)

## FieldMode

(TODO description what this does)

(TODO annotation)

(TODO method)

## Validation

Domain objects are validated before a method is executed. There is a separate section on validation (see [ValidationProvider](#_ValidationProvider))

## MultiLanguage

TODO see [LanguageProvider](#_LanguageProvider)

# Introspect Framework Concepts

This section describes the key concepts of the Introspect framework.

## Hexagon architecture

## The Introspect class

The introspect framework evolves around the static Introspect class. This class contains references to objects that are responsible for different [concerns](http://en.wikipedia.org/wiki/Separation_of_concerns) (called [providers](#_Providers) in the Introspect-Framework).

## Initializing the framework

The **Introspect class** needs to be initialized before it can be used. This is done once using the **IntrospectInitializer** interface. This interface basically provides the implementations of the different [provider](#_Providers) interfaces.

You can choose to implement this interface your self, but to make things easy, you can also choose to use one of the default implementations of the **IntrospectInitializer** interface depending on the type of application you are writing:

* A command line application (**CommandLineIntrospectInitializer**)
* A desktop application (**SwingIntrospectInitializer**)
* A mobile application (**AndroidIntrospectInitializer**)
* A web application (**VaadinIntrospectInitializer**)

Steps to initialize the introspect framework/ your application.

* Choose the **IntrospectInitializer** implementation you need.
* Implement the abstract methods to assign the application object(TODO link) and the [service objects](#_Service_Objects).
* Initialize the introspect framework with the **Introspect.initialize(yourIntrospectInitializer)**

TODO **code example of a swing application**

## Providers

Providers are interfaces for an specific [concern](http://en.wikipedia.org/wiki/Separation_of_concerns). The introspect framework distinguishes the following concerns/ provider interfaces:

* Managing the domain (DomainProvider)
* Managing validation (ValidationProvider)
* Managing the user interface (UserInterfaceProvider)
* Managing authorization (AuthorizationProvider)
* Managing paths (PathProvider)
* Managing multi languages (LanguageProvider)

Most providers have a default implementation. For example: the **DomainProvider** interface has a **DefaultDomainProvider** implementation.

Depending on the type application that you are going to write, you need to use a specific provider implementation. For example there are different implementations of the **UserInterfaceProvider** for:

* A command line application (**CommandLineUserInterfaceProvider**)
* A desktop application (**SwingUserInterfaceProvider**)
* A mobile application (**AndroidUserInterface**)
* A web application (**VaadinUserInterface**)

You could decide to write your own implementation of a [provider](#_Providers) interface (or extend an existing implementation) and plug it into the Introspect Framework using the [IntrospectInitializer](#_Initializing_the_framework).

## DomainProvider

## ValidationProvider

## UserInterfaceProvider

## AuthorizationProvider

## PathProvider

## LanguageProvider

## Dependency injection

The Introspect Framework provides a very basic implementation of dependency injection. It is build into the domain provider and helps domain objects, front end service objects and back end service objects to use eachother.

(TODO internal links + link to Dependency injection wiki)

(TODO how it works)

# Getting Started

This section is a how-to guide and explains all you need to know to create and deploy your own applications.

## Integrated Development Environment(IDE)

Get your favourite IDE installed on your computer. If you are a newby in developing applications, you could try and install Eclipse. It is a great free IDE that you can download from the internet. Any other IDE will do just fine. (TODO youtube video)

Download the Introspect Libraries from GIT into your IDE (TODO youtube video)

## Create your domain object's

## Creating your Introspect application

With introspect you can create different types of applications right out of the box:

* Create a command line application
* Create a desktop application
* TODO Create a mobile application
* TODO Create a web application
* TODO Create a soap application
* TODO Create a restfull XML application

TODO Each application type has it’s own “getting started” video.

## Create a command line application

(TODO youtube video)

## Create a desktop application

(TODO youtube video)

## Create a mobile application

(TODO youtube video)

## Create a web application

(TODO youtube video)

## Create a soap application

(TODO youtube video)

## Create a restfull XML application

(TODO youtube video)

## Icons

The Introspect user interface supports icons (see section on [icons](#_Icons)) or see the following tutorial video (TODO youtube video)

## *Language*

The Introspect user interface supports multi languages (see section on [LanguageProvider](#_LanguageProvider)) or see the following tutorial video (TODO youtube video)

## Validation

The Introspect user interface supports multi languages (see section on [LanguageProvider](#_LanguageProvider)) or see the following tutorial video (TODO youtube video)

## Authorization

(TODO youtube video)

## Reports

(TODO youtube video)

# Demo’s

TODO

# Downloads

[GitHub pages](https://github.com/ntenhoeve/Introspect-Framework)  
[Download introspect projects as a zip](https://github.com/ntenhoeve/Introspect-Framework/archive/master.zip)

Download the Introspect Manual TODO:add link

# Development

[Issue tracker](https://github.com/ntenhoeve/Introspect-Framework/issues)  
[Roadmap](https://github.com/ntenhoeve/Introspect-Framework/issues/milestones)  
[About the Developer](https://github.com/ntenhoeve/Introspect-Framework/wiki/About-the-Developer)