# Contents

[1 Contents 1](#_Toc397675338)

[2 About the Introspect Framework 3](#_Toc397675339)

[2.1 Why develop yet another application framework 3](#_Toc397675340)

[2.1.1 Reason 1: Because its fun 3](#_Toc397675341)

[2.1.2 Reason 2: Because I learned from it 3](#_Toc397675342)

[2.1.3 Reason 3: Because I wanted to do it better 3](#_Toc397675343)

[2.2 Introspect Core Values 4](#_Toc397675344)

[2.2.1 Based on the Naked Objects Design Pattern 4](#_Toc397675345)

[2.2.2 Provide a good structure for applications 4](#_Toc397675346)

[2.2.3 Lightweight 4](#_Toc397675347)

[2.2.4 Simple to configure 4](#_Toc397675348)

[2.2.5 No dependencies with tools 4](#_Toc397675349)

[2.3 Introspect Licence 4](#_Toc397675350)

[3 Introspect Domain Concepts 4](#_Toc397675351)

[3.1 Domain Object 5](#_Toc397675352)

[3.1.1 Domain Object Naming 5](#_Toc397675353)

[3.1.2 Domain Object Construction 5](#_Toc397675354)

[3.1.3 Domain Object Presentation 5](#_Toc397675355)

[3.2 Domain Object Properties 5](#_Toc397675356)

[3.3 Property Types 6](#_Toc397675357)

[3.4 Domain Object Methods (Actions) 6](#_Toc397675358)

[3.4.1 Domain Object Presentation 6](#_Toc397675359)

[3.5 Service Objects 7](#_Toc397675360)

[3.6 Front End Service Object 9](#_Toc397675361)

[3.6.1 Front End Service Object Naming 9](#_Toc397675362)

[3.6.2 Front End Service Object Construction 9](#_Toc397675363)

[3.6.3 Front End Service Object Presentation 9](#_Toc397675364)

[3.7 Back end Service Object 9](#_Toc397675365)

[3.7.1 Back End Service Object Naming 9](#_Toc397675366)

[3.7.2 Back End Service Object Construction 9](#_Toc397675367)

[3.7.3 Back End Service Object Presentation 10](#_Toc397675368)

[4 Adding object behaviour 10](#_Toc397675369)

[4.1 Title 10](#_Toc397675370)

[4.2 Visible In Form 10](#_Toc397675371)

[4.3 Visible In Table 10](#_Toc397675372)

[4.4 Enabled 11](#_Toc397675373)

[4.5 Order in Form 11](#_Toc397675374)

[4.6 Order in Table 11](#_Toc397675375)

[4.7 Method Execution Mode 11](#_Toc397675376)

[4.8 Parameter Factory 11](#_Toc397675377)

[4.9 Icons 11](#_Toc397675378)

[4.10 FieldMode 11](#_Toc397675379)

[4.11 Validation 11](#_Toc397675380)

[4.12 MultiLanguage 11](#_Toc397675381)

[5 Introspect Framework Concepts 12](#_Toc397675382)

[5.1 Architecture of an introspect application 12](#_Toc397675383)

[5.2 The Introspect class 13](#_Toc397675384)

[5.3 IntrospectApplication 13](#_Toc397675385)

[5.4 Initializing the framework 14](#_Toc397675386)

[5.5 Providers 14](#_Toc397675387)

[5.6 DomainProvider 15](#_Toc397675388)

[5.7 ValidationProvider 15](#_Toc397675389)

[5.8 UserInterfaceProvider 15](#_Toc397675390)

[5.9 AuthorizationProvider 15](#_Toc397675391)

[5.10 PathProvider 15](#_Toc397675392)

[5.11 LanguageProvider 15](#_Toc397675393)

[5.12 Dependency injection 15](#_Toc397675394)

[5.13 Lifecycle of modules 15](#_Toc397675395)

[6 Getting Started 15](#_Toc397675396)

[6.1 Integrated Development Environment(IDE) 16](#_Toc397675397)

[6.2 Create your domain object's 16](#_Toc397675398)

[6.3 Creating your Introspect application 16](#_Toc397675399)

[6.4 Create a command line application 16](#_Toc397675400)

[6.5 Create a desktop application 16](#_Toc397675401)

[6.6 Create a mobile application 16](#_Toc397675402)

[6.7 Create a web application 16](#_Toc397675403)

[6.8 Create a soap application 16](#_Toc397675404)

[6.9 Create a restfull XML application 16](#_Toc397675405)

[6.10 Icons 16](#_Toc397675406)

[*6.11* *Language* 16](#_Toc397675407)

[6.12 Validation 16](#_Toc397675408)

[6.13 Authorization 17](#_Toc397675409)

[6.14 Reports 17](#_Toc397675410)

[7 Demo’s 17](#_Toc397675411)

[8 Downloads 17](#_Toc397675412)

[9 Development 17](#_Toc397675413)

# 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 did I develop a new framework while there are so many Java application frameworks out there?

### Reason 1: Because its fun

Probably the best reason ever (see [Dutch article](http://www.sdn.nl/SDN/Artikelen/tabid/58/view/View/ArticleID/524/Just-because-its-fun.aspx))

### Reason 2: Because I learned from it

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.

### Reason 3: Because I wanted to do it better

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 its wrong!

* 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 a different approach: 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

See the Wiki page 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) (see [Architecture of an introspect application](#_Architecture_of_an)).
* Facilitate [domain driven design](http://en.wikipedia.org/wiki/Domain-driven_design).
* TODO links+ TODO decide to use FrontEndServiceObjects+BackEndServiceObjects or ServiceObjects+InfrastructureObjects.
* The Introspect Framework should not create a [vendor lock-in](http://en.wikipedia.org/wiki/Vendor_lock-in) (not that there is such a thing as an Introspect Framework Vendor, since it is open source). Domain classes, service classes and infrastructure classes should have no (are almost no) dependencies with the IntrospectFramework so that the IntrospectFramework could easily (within a few hours) be replaced with another dependency injection framework (like [Spring](http://en.wikipedia.org/wiki/Spring_Framework), [Guice](http://en.wikipedia.org/wiki/Google_Guice) or [Pico container](http://picocontainer.codehaus.org/)).
* 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

The Introspect framework has no configuration files (see [“Code or configuration files”](http://www.martinfowler.com/articles/injection.html#CodeOrConfigurationFiles) chapter in Martin Fowlers article for the arguments why).

There are 2 things that need to be configured:

* 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)

Furthermore the Introspect framework prefers “Configure by Exception”. This means that the Introspect framework uses reasonable defaults wherever possible. These default values can be overridden by the developer.

### No dependencies with tools

* 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.

Domain objects already exist in an application that needs to be re-written, or are created by the developer . They can be created from scratch or generated from a schema (in example from a [database schema](http://en.wikipedia.org/wiki/Database_schema), [XML schema](http://nl.wikipedia.org/wiki/XML_Schema) or [web service](http://en.wikipedia.org/wiki/Web_Services_Description_Language))

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 properties 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 (Actions)

An action is a method that is intended to be invoked by a user - though it may also be invoked programmatically from within another method or another object.

For example:

* An Order class may have an action method addOrderLine(OrderLine orderLine)
* An OrderService May have an action method findOrdersOfCustommer(Customer customer)

By default, any public instance method that you add to a class (whether it is a domain class or a service) will be treated as a user action, provided that it complies with the following:

* The method has no or one parameter. If it has a parameter it needs to be of a type that is recognized by the Introspect framework
* Its return type (if any) are types recognized by the Introspect framework.

A method will also not be treated as an action if it represents a property or another recognized method. Note also that static methods are ignored by the Introspect framework.

If you have a method that you don't want to be made available as a user-action you should either:

* Give it a non-public (private) access modifier
* Mark it with the VisibleInForm and/or VisibleInTable attribute with value false
* Mark it with the Enabled attribute with value false

You can specify certain things about both the behavior and presentation of actions by adding specific attributes or methods. See [object behavior](#_Adding_object_behaviour)

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.

### Domain Object Presentation

An Introspect framework can display domain objects in multiple ways:

As a table:

TODO table picture of customers

Objects are displayed as rows in a table if an method of a [front end service object] returns any of the recognised collection types (in example: the allCustommers() method of the CustomerService class returns a collection of customers)

As a read-only form:

TO-DO: show read-only form of a customer

An object is displayed as an read-only form when a [front end service object] returns any of the recognised reference types. (in example: the viewCustomer(TODO) method of the CustomerService class returns a Customer object)

TO-DO EDITABLE FORM

TO-DO REFERENCE FIELD IN FORM

TO-DO COLLECTION FIELD IN FORM

## 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 POJO 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.

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.

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.

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.

### Service Object Naming

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

### 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.

TODO chapter on constructors and dependency injector: 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.

### Service Object Presentation

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

## Infrastructure Object

The principle of “naked objects” is that any '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 (with no arguments). This allows easy instantiation within the framework.

### Infrastructure Object Naming

Back end service objects are normally named to ….

Examples:

TODO

GoogleMapsHttpClient

MailClient

CustomerReporitory

### Infrastructure Object Construction

TODO

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.

### Infrastructure Object Presentation

The methods of infrastructure object are not visible to the IntrospectUserinterfaceTODO link.

# 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.

Behavior can be added with:

- methods that are recognized bin the introspect framework

- methods that are recognized book the Introspect framework

TO-DO RECOGNIZED METHODS (see naked objects doc)

TO-DO RECOGNIZED ANNOTATIONS

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

## Title

The Application object, domain objects and service objects have a title which is displayed in the user interface. This title defaults to the class name, but you can override the title.

(TODO title annotation + example)

(TODO title method + example)

(TODO toString method + example)

(TODO titlebuilder + example)

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

Sometimes applications need to support multiple languages. There is as separate section on Multilanguage (see [LanguageProvider](#_LanguageProvider))

TODO example

# Introspect Framework Concepts

This section describes the key concepts of the Introspect framework.

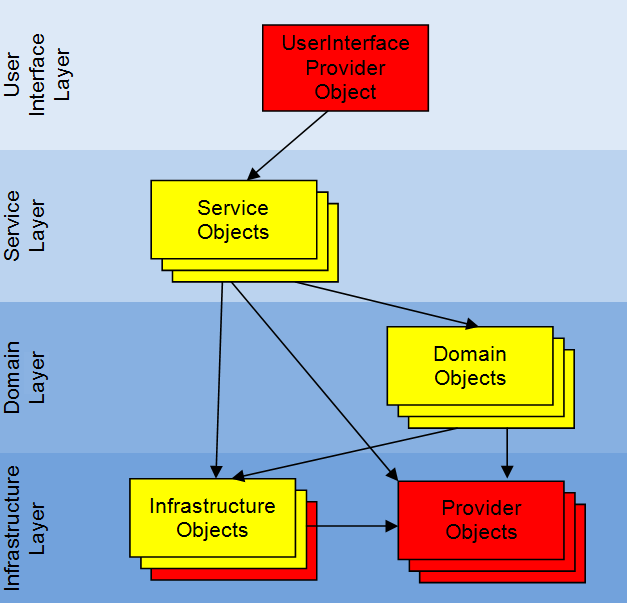
## Architecture of an introspect application

The Introspect Framework helps you to create a [multi layered architecture](http://en.wikipedia.org/wiki/Multilayered_architecture) for your application. A multi layer architecture has several [design principles](http://martinfowler.com/bliki/LayeringPrinciples.html):

* Separation of concerns (separate responsibilities within the application into different layers).
* Low coupling between layers, high cohesion within them.
* User interface modules should contain no business logic.
* Layers should be testable individual.
* Business logic layers contain no user interface and don't refer to user interface modules.
* No circular references between layers.
* Business layer only uses abstractions of technological services.
* Lower layers should not depend on upper layers.
* Layers should be shy about their internals.
* Layers may share infrastructural aspects (eg security)
* Inbound external interface modules (eg web service handlers) should not contain business logic.

These multi layer architecture design principles try to prevent [spaghetti code](http://en.wikipedia.org/wiki/Spaghetti_code), which is hard to extend, hard to trouble shoot, hard to test, and hard to keep [bug](http://en.wikipedia.org/wiki/Software_bug#Etymology) free.

There are many different opinions on how a multi layered architecture [should look like](https://www.google.nl/search?q=layered+architecture&tbm=isch). It really doesn’t matter what it should look like, as long as your multi layered architecture complies with the design principles above. Below you will find one representation of a multi tier application, to explain how a application with the introspect application could (/should) look like.



Red objects are provided by the Introspect framework

Yellow objects need to be written or included by the developer.

The Introspect Framework has 2 additional objects that do not fit in any of these layers.

* IntrospectFramework object \*, which is basically a dependency injection container that creates, links and holds references to all the objects above
* IntrospectApplication object \*, provides all the information needed to initialize the Introspect Framework object

Please read Eric Evans brilliant book Domain Driven Design. In his book he explains (among other things) the well accepted concept of domain objects being the core layer of an application and the functions of the other layers (in Eric Evans words):

### User Interface Layer

Also known as presentation layer.

Responsible for showing information to the user and interpreting the user’s commands. The external actor might sometimes be another computer system rather than a human user.

### Service Layer

Also known as application layer.

Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems. The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems. This layer is kept thin (see Martin Fowlers article on [Anemic Domain Model](http://martinfowler.com/bliki/AnemicDomainModel.html) on why the service layer should be thin). It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down. It does not have state reflecting the

business situation, but it can have state that reflects the progress of a task for the user or the program.

### Domain Layer

Also known as business layer, business logic layer, domain model layer.

Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure. This layer is the

heart of business software.

### Infrastructure Layer

Also known as [data access layer](http://en.wikipedia.org/wiki/Data_access_layer), [persistence layer](http://en.wikipedia.org/wiki/Persistence_layer), logging, networking, and other services which are required to support the service or domain layer.

Provide generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, etc. The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

## 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).

## IntrospectApplication

An Introspect Application class basically provides all the information needed (configuration) to start the Introspect Framework.

The Introspect Framework favours configuration by code over configuration by configuration files (see [“Code or configuration files”](http://www.martinfowler.com/articles/injection.html#CodeOrConfigurationFiles) chapter in Martin Fowlers article for the arguments why). Configuring the Introspect Framework configuration is done by implementing the IntrospectApplication interface. If you favour to use configuration files, it should not be difficult to write an application class that implement the IntrospectApplication interface which gets the information from a configuration file.

An Introspect Application class serves several purposes:

* It provides the name, description and an optional icon of the application. See section [Object Behavior](#_Adding_object_behaviour)
* It provides the information needed to initialize the Introspect framework. See section [initializing the introspect framework]:
* [provider classes]
* [front end service classes]
* [back end service classes]
* In some cases the application class needs to extend a class of the user interface framework that is being used in the background:
  + For mobile applications (Android) the application class needs to extend ....!!!
  + For web applications (Vaadin) the application class needs to extend ....!!!

TODO example

## Initializing the framework

The [Introspect class] (todo link) (framework) needs to be initialized before it can be used. This is done once using an [IntrospectApplication class] (todo link).

You can choose to implement this interface yourself, but to make things easy, you can also choose to extend one of the default implementations of the application interface depending on the type of application you are writing:

* A command line application (IntrospectCommandLineApplication)
* A desktop application (IntrospectSwingApplication)
* A mobile application (IntrospectAndroidApplication)
* A web application (IntrospectVaadinApplication)

These default application classes implement the get…Provider methods of the [IntrospectApplication class] (TODO ref) so that the correct provider implementations are given for the application type they represent.

Steps to initialize the introspect framework/ your application:

Create an application class by extending one of the default application class (or by doing it the hard way by implementing the full [IntrospectApplication] interface yourself)

Implement the remaining getFrontEndServiceClasses() and getBackEndServiceClasses() methods to provide the [service object types]. Note that service classes can only be registered as [front end service objects](#_Front_End_Service) (with action methods that are accessible to the user) or as [back end service objects](#_Back_end_Service) (with methods not visible to the user) but not as both!

TODO: Initialize the introspect framework with the Introspect.initialize(yourApplication Instance)

change to: new IntrospectFramework(yourApplicationInstance)

TODO code example of a swing AcmeeSalesApplication

For more examples see the [Getting started] section.

## 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 [[IntrospectApplication](#_Initializing_the_framework)] (TODO link) class.

## DomainProvider

## ValidationProvider

## UserInterfaceProvider

## AuthorizationProvider

## PathProvider

## LanguageProvider

## Dependency injection

The Introspect Framework provides a basic implementation of [dependency injection](http://en.wikipedia.org/wiki/Dependency_injection) framework.

Please read Martin Fowlers [“Inversion of Control Containers and the Dependency Injection pattern”](http://www.martinfowler.com/articles/injection.html) article, to understand the advantages of a Dependency Injection

The introspect Framework was designed with the following considerations:

* The Introspect Framework only supports constructor injection. See the [“Constructor versus Setter Injection”](http://www.martinfowler.com/articles/injection.html#ConstructorVersusSetterInjection) chapter in Martin Fowlers article, to understand why we favour constructor injection. TODO Example
* All injected classes need to be defined by the Application object (TODO ref). The following classes can be injected:
  + [Application Object](#_The_Introspect_Application)
  + [IntrospectionFramework Object](#_The_Introspect_class)
  + [Provider Objects](#_Providers)
  + [Front End Service Objects](#_Front_End_Service)
  + [Back End Service Objects](#_Back_end_Service)
* Domain objects can not be injected: Service object know domain objects and not the other way around.

## Lifecycle of modules

TODO when are they created

TODO when are they killed

TODO lifecycle listeners

# 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)