Robotlegs

&

Signals

Just what are Robotlegs exactly?

Robotlegs is a pure AS3 micro-architecture (framework) with a light footprint and limited scope. Simply put, Robotlegs is there to help you wire your objects together. It provides the glue that your application needs to easily function in a decoupled way. Through the use of automated metadata based dependency injection Robotlegs removes boilerplate code in an application. By promoting loose coupling and avoiding the use of Singletons and statics in the framework Robotlegs can help you write code that is highly testable.

Robotlegs is a pure AS3 micro-architecture (framework) with a light footprint and limited scope. Simply put, **Robotlegs is there to help you wire your objects together.** It provides the glue that your application needs to easily function in a decoupled way. Through the use of automated metadata based dependency injection Robotlegs removes boilerplate code in an application. By promoting loose coupling and avoiding the use of Singletons and statics in the framework Robotlegs can help you write code that is highly testable.

Robotlegs is a pure AS3 micro-architecture (framework) with a light footprint and limited scope. Simply put, Robotlegs is there to help you wire your objects together. It provides the glue that your application needs to easily function in a decoupled way. **Through the use of automated metadata based dependency injection** Robotlegs removes boilerplate code in an application. By promoting loose coupling and avoiding the use of Singletons and statics in the framework Robotlegs can help you write code that is highly testable.

?!

Wikipedia says:

Dependency injection (**DI**) in object orientated programming is a technique for supplying an external dependency (i.e. a reference) to a software component - that is, indicating to a part of a program which other parts it can use. It is a specific form of inversion of control where the concern being inverted is the process of obtaining the needed dependency.

Dependency Injection!

OR:

Assigning variables at runtime.

Dependency Injection!

OR:

Assigning variables at runtime.

This is dependency injection:

```
myProperty = myReference
```

Dependency Injection!

OR:

Assigning variables at runtime.

This is dependency injection:

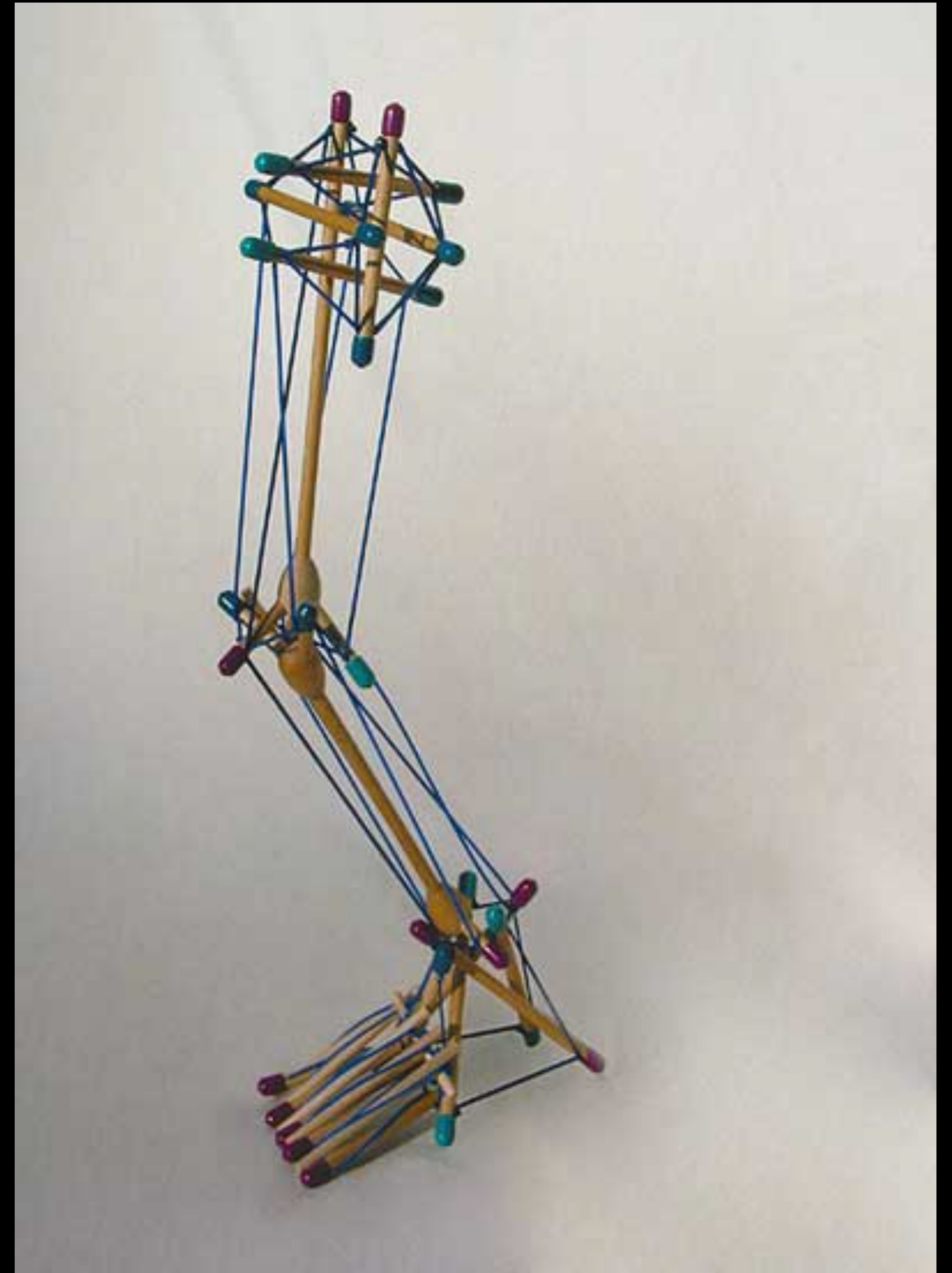
```
myProperty = myReference
```

This is how RL does it:

```
[Inject] public var myProperty:IMyInterface;
```

The Main Parts

Actor	model
Mediator	view
Command	controller?
Context	magic glue



Example 1

Just Robotlegs

http://www.robotlegs.org/
http://github.com/tschneidereit/SwiftSuspenders

Signals

Signals

The Concept

- A **Signal** is essentially a mini-dispatcher specific to one event, with its own array of listeners.
- A Signal gives an event a concrete membership in a class.
- Listeners subscribe to real objects, not to string-based channels.
- Event string constants are no longer needed.

Signals

Philosophy

- Composition and interfaces are favored over inheritance.
- Events in interfaces are a good thing.
- Event types are classes, not strings.
- Event classes should focus on the data they contain, not on who sends them.
- Event classes should not contain string constants that various other classes use.
- Test-Driven Development (TDD) is essential and fun.
- So far, not a single static keyword in the source.
- No singletons.

Signals

```
signal = new Signal(valueClasses)

ISignal:
add(listener:Function)
addOnce(listener:Function)
remove(listener:Function)
removeAll()

IDispatcher:
dispatch()
```

Example 2

Robotlegs & Signals

http://www.robotlegs.org/
http://github.com/tschneidereit/SwiftSuspenders
http://github.com/robertpenner/as3-signals
http://github.com/richtextformat/RichTextFormat-Classes/tree/master/uk/co/richtextformat/signals/

Context

mapValue

mapClass

mapSingleton

mapSingletonOf

mapRule

mediatorMap.mapView

commandMap.mapEvent

Context

```
mapValue(whenAskedFor:Class, useValue:Object,
named:String = “”)
```

Use when:

You have an existing instance that you want to use for the mapping. Useful when you need to set properties before the class is mapped.

Context

```
mapClass(whenAskedFor:Class,
instantiateClass:Class, named:String = “”)
```

Use when:

You want to create a new instance every time the mapping is requested.

Context

```
mapSingleton(whenAskedFor:Class, named:String = “”)
```

Use when:

You always want to use the same object for the mapping.

Context

```
mapSingletonOf(whenAskedFor:Class,
useSingletonOf:Class, named:String = “”)
```

Use When:

You want to create an instance of another type than the whenAskedFor class to use for the mapping.

Context

```
mediatorMap.mapView(viewClassOrName:*,
mediatorClass:Class, injectViewAs:Class = null,
autoCreate:Boolean = true, autoRemove:Boolean =
true)
```

Use When:

You want to map a mediator class to a view.

Use the injectViewAs parameter to perform a cast when injecting the view class.

Example 3

Katari

http://www.robotlegs.org/
http://github.com/tschneidereit/SwiftSuspenders
http://github.com/robertpenner/as3-signals

Gotchas

Gotchas

PROBLEM:

One component vs many Mediators

Gotchas

PROBLEM:

One component vs many Mediators

SOLUTION:

Create an empty class or interface,
use injectViewAs parameter

Gotchas

PROBLEM:

PopUpManager

Gotchas

PROBLEM:

PopUpManager

SOLUTION:

DIY Mediation, use autoCreate and autoRemove parameters

Gotchas

```
// In the class that maps the mediator:
mediatorMap.mapView( MyPopupView, MyPopupViewMediator,
false ); //disable auto mediation

// In a command or mediator:
var popup:MyPopupView = new MyPopupView();
PopUpManager.addPopUp( popup, contextView ); //
contextView is defined in Command
mediatorMap.createMediator( popup );
```

google: robotlegs popupmanager

More

RL

http://www.robotlegs.org/

http://knowledge.robotlegs.org/faqs/

http://github.com/robotlegs/robotlegs-demos-Bundle

Signals

http://robertpenner.com/flashblog/

End.