# Brainstorming on markup

In statically typed scenarios (closure world etc) - we can do automatic refactoring like java - where the language knows the name resolution, outer classes are closed (not dynamic etc). Let's notice not all refactorings can be done automatically

## Closure world...

- Name resolution rules...
- local variable name
- (all variables declared in closures are considered local vars)
- outer class static names
- outer class dynamic names
- Code can move into and out of closures WITHOUT any changes - irrespective of the method implemetnation detail
- the compiler, IDE, shell can KNOW that the names map to (assuming the outer class is static, not open/dymamic

```
// closure world
// in a script or a dynamic class (open)

def x= 1
title = foo
for (...) {
    def x = 1
    title = foo
}
collect.each {
    title = "foo"
}
```

## Builder world

- The builder decides what vanilla names mean

normal static resolution

- local variables
- delegate to builder.. it decides...
- default order is,
- static builder names
- static outer class names
- dynamic builder names
- dynamic outer class names
- Builders

```
// builder land

idea...

swing.{
    frame() {
        panel {
            button()
        }
    }
    title = "foo"
}

builder (swing).frame {
    panel {
        button(title:"foo")
    }
    title = "hey"
}

use (swing) {
  frame {
    panel {
        button(title:"foo")
    }
    title = "hey"
  }
}

//jez idea...
swing("""
 frame{panel{button()}}
""")

def c = with() {
  frame {
    panel {
        button(title:"foo")
    }
    title = "hey"
  }

swing << c
}
```

## possible precedence rules from within usage of a builder

### Johns proposal...

- local variables
- static outer class names
- dynamic outer class names
- static builder names
- dynamic builder names

### James proposal...

normal static resolution

- local variables
- static outer class names

changed dynamic resolution...

- static builder names
- dynamic builder names
- dynamic outer class names

## Markup ideas

### Stuff Guillaume is happy-enough with

```
// Guillaumes suggestion
// though this conflicts with using quoted names on any method call
swing."frame" {
    "panel" {
    }
}

swing.frame {
    swing.panel {
        swing.button()
    }
}
```

Other ideas

```
builder(foo) {
    frame() {
    }
}

// other alternatives
// magic method name
builder.$build() {
    frame() {
    }
}

builder {{
    frame() {
    }
}}

builder <{
    frame() {
    }
}>

/** wacky **/

/** end wacky **/

swing = new SwingBuilder()
namespace(swing) {
    frame() {
        button()
    }
}

wacky (swing) {
}

@Markup(swing) {
}

dynamic(swing) {
    ...
}
```

## Passing closures into markup

```
def makeMarkup(builder) {
    namesapce(builder) {
        frame {
            cheese.each { println it }
            bar() {
                whatnot()
            }
            this.toString()
         }
    }
}

def c = namesapce(builder) {
        frame {
            cheese.each { println it }
            bar() {
                whatnot()
            }
            this.toString()
         }
    }
}

def c = { builder ->
        namesapce(builder) {
            frame {
                cheese.each { println it }
                bar() {
                 whatnot()
              }
             this.toString()
            }
        }

c(swing)


def c = { b ->
            b.frame {
                cheese.each { println it }
                b.bar() {
                 b.whatnot()
              }
             this.toString()
        }

c(swing)

}

//... time passes

swing = new SwingBuilder()
def c = makeMarkup(swing)
```

Guillaumes suggestion....

- markup

```
def panel() {}

swing.frame {
    "panel" {...}
}
```

(* regular method call

```
    swing."foo"...
```

# Random bits of snippets

In scripts, what should be in the binding? or local variables?

```
binding.x
def a
a
```

Jochen:

```
class Foo {
    def method() { println bar } // this will have to throw an error!
}

class Foo2 extends Foo {
    @Property bar
}
```

- is it dependent on type of value or type of variable
- discussion ensued about static compile time checing
- if Foo has a method added after compiletime such as getBar(),
  then should the compiler give error... (as above in Foo)?

- vanilla names used to catch typos, whereas in markup the typos are ignored?
- late binding on properties and methods, but not on names
- good error handling is important

```
class Foo {
    def method() { println .bar } // is this a way to call unbound variables?
}

class Foo2 extends Foo {
    @Property bar
}
```

- lexical scope has to be handled by compiler
- classnames can't be dynamically bound
- scripts are normally associated with command line shells, and you have the concept of a variable stack.

- threadlocal variable scope?

- maybe when extending Expando, vanilla names are unbound?

```
// todo - need better name, change naming
@NoValidationOnVanillaNames
class Foo {
    def method() { println bar } // is this a way to call unbound variables?
}
```

- Not a lexical name, e.g. $foo is like a dynamic name which no checking is done for.
- $ seems slightly cleaner, because it is like a seperate namespace
- println foo.?bar
- An explicit syntax to mean a dynamic name

- 3 Options

```
@Dynamic class Foo {
  def foo() { println bar }
}
```

or

```
class Foo {
  def foo() {println $bar}
}
```

or

```
groovyc -NoStrict Foo.groovy
```

scope:

```
class Remote() {
  def foo() {println 2}
}

//tug
def foo() {println 1}
remote.startSesssion(1, 2) {cred ->
      foo(cred, 2, 4)
}
```

or

```
//jstrachan
with (remote) {
  startSession(1, 2) { cred ->
    foo(cred, 2, 4)
  }
}
```

or

```
remote.startSesssion(1, 2) {cred ->
      remote.foo(cred, 2, 4)
}
```

```
class Foo {

def exec() {
  for(i in 1..100) {
    foo()
  }
  c = with() {
    foo()
  }
  builder << c
}

def foo(){}
}
```

- there needs to be an explicit rule where names are searched

Choice:

Either...

- Vanilla Name Resolution uses the names in the current editor
  (bound to the lexical context)

or

- Vanilla Name Resolution uses the names in the enclosing object
  (bound to the instance being referenced at the time)

or

both! - which is default, what is syntax for non-default
e.g.

```
foo() could be  this.foo() or with(x) {foo()}
```

jstrachan - "foo should be looked for in the metaclass of the outer class,
it should never look in the metaclass of the closure"