Mittie's After-Meeting Proposal

Sorry
In the meeting, I wasn't quick enough in thinking.
Therefore, I need to give my input so late that we cannot discuss it
in the room. I'm sorry for that. This page should serve as the second-best solution.

Overview

Topics to cover:

- rephrasing the issue
- changes to closures
- remaining issue for 'builders'
- list of proposals
- final

rephrasing the issue

Groovy code may contain identifiers (aka 'names') with or without
qualifiers. Examples are:

qualified identifiers	unqualified identifiers	note
this.prop	**prop**	*problematic*
someObject.method()	**method()**	*problematic*
	def localVar; localVar	*new:* statically resolved
	MyClass	*new:* statically resolved

The bold examples above are also called *vanilla* names.

They are problematic because one legally expects each **`vanilla`** name to be the
same as **`this.vanilla`**, which is currently not the case.

In code like

surprising name resolution

```
def builder = new MarkupBuilder()
builder.html{
    body {
        div { }
    }
}
```

it is *surprising* that the `body` method call is *resolved* by `builder` although
`this` refers to its surrounding Closure.

Implementation note on name resolution
When we say a *name* is resolved by an *object*, it is actually that object's metaclass that resolves that name.
The wording *name* is resolved by *object* is only a shorthand in the description.

With the decision described in Paris Groovy Meeting Report the following changes
take place.

changes to closures

To make the handling of vanilla names inside closure less surprising with regard to implicit **`this.`**,
the following decision was taken.

> **'this' in closures now refers to the 'declarer'**
> Inside closures, **`this`** will no longer refer to the closure but to the object in that's scope the closure was declared.
> We will call that object the *declarer*.

This has two effects:

1. inside usual closures, **`vanilla`** is the same as **`this.vanilla`**
2. inside closures that work on builders, explicit **`this`** can be used to refer to the *declarer* (sometimes called *escaping* the builder).

remaining issue for 'builders'

Inside closures that work on builders, the surprising effect of name resolution remains that
`vanilla` is resolved by the builder, while **`this.vanilla`** is resolved by the *declarer*.

Note that this behaviour currently applies to all builders (not only markup): AntBuilder, NodeBuilder, SwingBuilder, and (Streaming)MarkupBuilder.
It also applies to subclasses of BuilderSupport that are in codebases out there.
Since builder's are made by usual Groovy means, any Groovy codebase may contain code that
applies the same method resolution trick that BuilderSupport implements, even for usages beyond 'building'.

Therefore, it is not only a 'builder' issue, it is a general issue of possibly surprising name resolution.

concerns with the remaining issue and currently proposed solutions

Proceeding with the status quo raises the following concerns (mostly contributed by James):

1. no *visual* clue for the casual reader of the program that name resolution rules have changed inside the builder closure
2. compiler cannot recognize the changed namespace and therefore cannot check for possible name typos. This will either lead to too many warnings or no warnings at all since too many warnings make everybody ignoring them.
3. IDE's have the same problem as the compiler when it comes to code completion etc.
4. *please add more here* ...

Concerns when going for a visual-clue-markup solution

1. effort for changing existing codebases that use builders or equivalent concepts
2. visual clutter / less elegant use of builders
3. *please add more here* ...

list of proposals

I feel we can address these fully valid concerns in two ways:

1. use of a to-be-defined namespace declaration is optional
2. find a declaration style that *only* affects the outermost closure of a builder

Let's assume we have a keyword **`xxx`** which would demarcate the namespace for a builder
in the following way

keyword-style namespace demarcation

```
def builder = new MarkupBuilder()
xxx (builder) {      // new line and new indentation level :-(
    html{                          // this line even nicer than before   :-)
        body {
            div { p '' }
        }
    }
}
// when used with logic
def builder = new MarkupBuilder()
xxx (builder) {
    html{
        body {
            [1,2].each {
             xxx(builder) { div { p '' } }  // either namespace or
             builder.div { builder.p ''}    // qualify all
            }
        }
    }
}
```

Beside using the **xxx** keyword as **`xxx(builder)`, we could also have it as `*builder.xxx`**.

Here are some proposals for xxx:

xxx(builder){}	builder.xxx {}	note
with(builder)	builder.with	with keyword already reserved
identity(builder)	builder.identity	construct already known to Groovy users
names(builder)	builder.names	reads nicely as verb or plural noun. But 'names' may be a too common variable name
vanilla(builder)	builder.vanilla	sounds like icecream
namespace(builder)	builder.namespace	for the XML-infected
add here	*add here*	*add here*

final

Builders are currently a library feature, not part of the language. I feel we should be careful with what we make part of the language. Introducing additional tokens would make them part of the language, the above demarcation doesn't. It makes only the xxx part of the language.

Use of the xxx construct is actually left to the programmer, analogous to the decision whether he wants to use static or dynamic typing. Even if we wanted, enforcing the construct wouldn't be possible. At any time, a programmer would be able to come up with a class like BuilderSupport and a MetaClass for it that does the trick.

So we essentially provide a mean that:

- support least-surprising builders
- still allows full flexibility
- relies on the programmer's responsibility for writing self-describing code.

Created by Dierk König
On Sun Nov 27 16:24:52 CET 2005
Using TimTam