

Gant – the lightweight and Groovy scripting framework

Dr Russel Winder

Partner, Concertant LLP

russel.winder@concertant.com

Aims and Objectives of the Session

- Look at *Gant* as a way of:
 - Defining a set of targets.
 - Scripting Ant tasks.
 - Using AntBuilder.
 - Using the Groovy meta-object protocol.

*Have a structured “chin wag”
that is (hopefully) both
illuminating **and** enlightening.*

Structure of the Session

- Do stuff.
- Exit stage (left | right).

There is significant dynamic binding to the session so the above is just a set of place holders.

Protocol for the Session

- A sequence of slides, interrupted by various bits of code.
- Example executions of code – with the illuminating presence of a system monitor.
- Questions (*and, indeed, answers*) from the audience as and when they crop up.

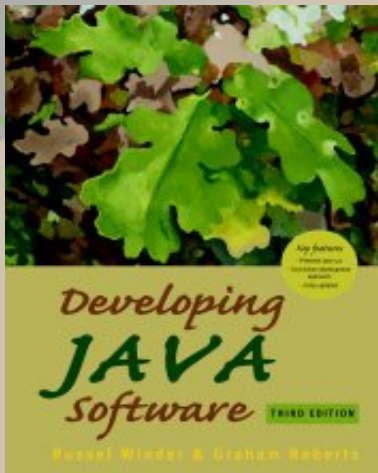
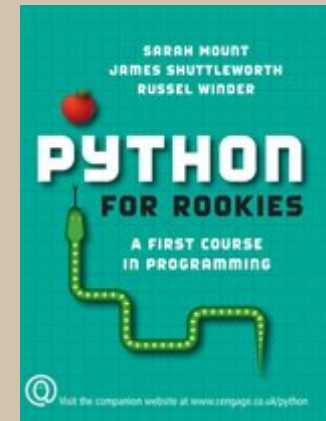
If an interaction looks like it is getting too involved, we reserve the right to stack it for handling after the session.

Blatant Advertising

Python for Rookies

Sarah Mount, James Shuttleworth and
Russel Winder

Thomson Learning Now called *Cengage Learning*.



Developing Java Software Third Edition

Russel Winder and Graham Roberts

Wiley

Buy these books!



XML

- Does anyone like programming using XML?
 - Ant has always claimed Ant scripts are declarative programs – at least that is better than trying to write imperative programs using XML.
 - Most Ant scripts exhibit significant control flow tendencies.

gs/XML/Groovy/g

- *Gant* started as an Ant-style build tool using Groovy instead of XML to specify the build.
- Ant <-> Maven warfare, configuration vs. convention.
- *Gradle* (and *SCons*) shows that build is better handled using a system that works using a directed acyclic graph of all dependencies.

Real Programmers use ed.

If Gradle, why Gant?

- *Gant* is a lightweight target-oriented scripting system built on Groovy and its AntBuilder.
- *Gradle* is a build framework.

Gant is lightweight, Gradle is a little more serious.

Lightweight means embeddable, hence it being used in Grails.

Gant Scripts . . .

- Are Groovy scripts.
- Have targets as well as other code.
- Are executed directly – no data structure building phase as with Gradle and SCons.

Use functions, closures, etc. to structure the code and make the targets the multiple entries into the codebase.

Targets are . . .

Think Fortran entry statement

- The entry points.
- Callable -- targets are closures and hence callable from other targets and indeed any other code, but this is a bad coding strategy.
- Independent – there is no way of specifying dependencies of one target on another except by executing code. Cf. depends function.

Although targets become closures, treat them as entry points, not callables.

Hello World

```
def helloWorld ( ) {  
  println ( 'Hello World.' )  
}  
target ( 'default' : 'The default target.' ) {  
  helloWorld ( )  
}
```

Parameter is a Map

|> gant -p -f helloWorld.gant

default The default target.

Default target is default.

|>

Hello World – 2

```
def helloWorld ( ) {  
    println ( 'Hello World.' )  
}  
target ( printHelloWorld : 'Print Hello World.' ) {  
    helloWorld ( )  
}  
setDefaultTarget ( printHelloWorld )
```

|> gant -p -f helloWorld_2.gant

printHelloWorld Print Hello World.

Default target is printHelloWorld.

|>

HelloWorld – 3

```
def helloWorld ( ) {  
  println ( 'Hello World.' )  
}  
target ( doHelloWorldPrint : 'Print Hello World.' ) {  
  helloWorld ( )  
}  
target ( printHelloWorld : 'Force doHelloWorldPrint to be achieved.' ) {  
  depends ( doHelloWorldPrint )  
}  
setDefaultTarget ( printHelloWorld )
```

Hello World – 4

```
def helloWorld ( ) {  
    println ( 'Hello World.' )  
}  
target ( printHelloWorld : 'Force doHelloWorldPrint to be achieved.' ) {  
    depends ( helloWorld )  
}  
setDefaultTarget ( printHelloWorld )
```

This does not work :-)

Hello World – 5 & 6

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( printHelloWorld : 'Print Hell World.' ) {  
  depends ( helloWorld )  
}  
setDefaultTarget ( printHelloWorld )
```

These work, but is it better to make the variable local or use the global binding?

```
helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( printHelloWorld : 'Print Hell World.' ) {  
  depends ( helloWorld )  
}  
setDefaultTarget ( printHelloWorld )
```

Hello World – 7 & 8

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
globalPreHook = helloWorld  
target ( printHelloWorld : 'Print Hello World.' ) { }  
setDefaultTarget ( printHelloWorld )
```

The global hooks are just variables in the binding – callable or list of callables assumed.

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
globalPostHook = [ helloWorld ]  
target ( printHelloWorld : 'Print Hello World.' ) { }  
setDefaultTarget ( printHelloWorld )
```

Hello World – 9 & 10

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( name : 'printHelloWorld' , description : 'Print Hello World.' ,  
  addprehook : helloWorld ) { }  
setDefaultTarget ( printHelloWorld )
```

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( name : 'printHelloWorld' , description : 'Print Hello World.' ,  
  addposthook : helloWorld ) { }  
setDefaultTarget ( printHelloWorld )
```

Hello World – 11 & 12

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( name : 'printHelloWorld' , description : 'Print Hello World.' ,  
prehook : helloWorld ) { }  
setDefaultTarget ( printHelloWorld )
```

```
def helloWorld = {  
  println ( 'Hello World.' )  
}  
target ( name : 'printHelloWorld' , description : 'Print Hello World.' ,  
posthook : helloWorld ) { }  
setDefaultTarget ( printHelloWorld )
```

Hello World – 13 & 14

```
target ( 'Hello World.' : 'Print Hello World.' ) { println ( it.name ) }  
setDefaultTarget ( 'Hello World.' )
```

```
target ( printHelloWorld : 'Hello World.' ) { println ( it.description ) }  
setDefaultTarget ( printHelloWorld )
```

Doing More Than One Thing At Once

- The world is a parallel place – multicore is now the norm.
- Workstations have 2, 3, 4, 6, 8, . . . cores.
- Sequential, uniprocessor thinking is now archaic.

GPars is Groovy Parallelism

- Was GParallelizer is now GPars.
- <http://gpars.codehaus.org>
- Will have a selection of tools:
 - Actors
 - Active objects
 - Dataflow
 - CSP

CSP – Communicating Sequential Processes

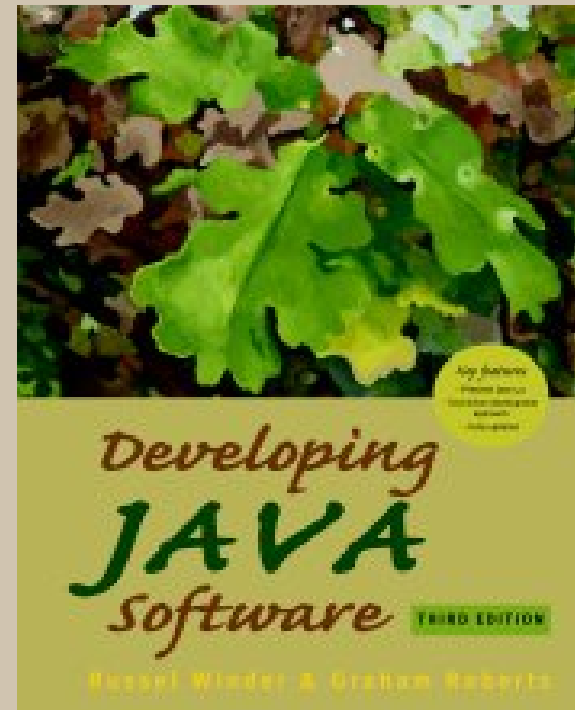
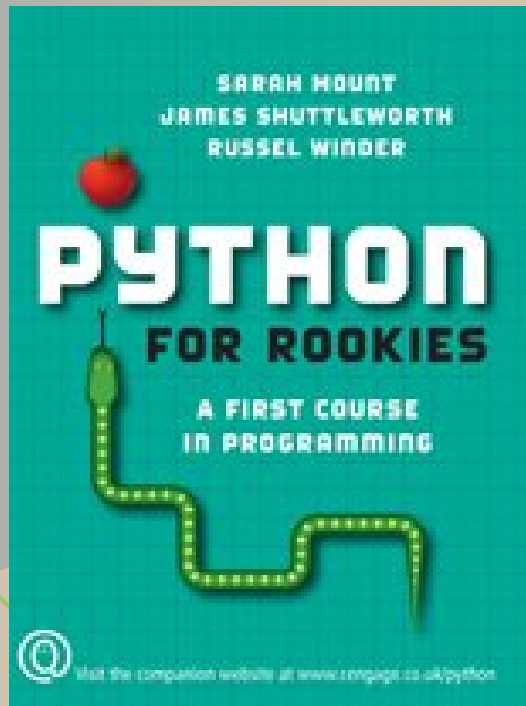
Actors and Data Parallelism

- Some examples of using GParas in Groovy.
- Some examples of using GParas in Gant.

Summary

- Interesting things were said.
- People enjoyed the session.

Unabashed Advertising



You know you want to buy them!



GParas Logo Contest

- GParas project is having a logo contest: See <http://docs.codehaus.org/display/GPARS/Logo+Contest>
- Closing date for entries was 2009-11-30.
- Voting is now happening.

Late entries are not accepted, though if significant bribes to the organizer are involved . . .

