Groovy Fast Track

Using the Groovy Fast Track you will start experimenting with GPars in about 3 minutes. We assume you have Groovy installed on your system.

Step 1 - Start up the Groovy Console

Start fresh Groovy Console or open up an empty groovy script source in your favorite IDE

Step 2 - Add dependencies

Note: GPars comes bundled with Groovy 1.8.x so this step is only requited if you are using an older version of Groovy.

We'll use Groovy's Grape functionaity to grab all the requited dependencies for us. You may check ot the GPars Integration page for alternative ways to integrate GPars with your project.

Add the following line to the groovy script:

```
@Grab(group='org.codehaus.gpars',
module='gpars', version='0.12')
```

Step 3 - Experiment with parallel collection processing

Believe it or not, now, we're ready to experiment. Try the following script, which will concurrently query a collection of strings with regular expressions:

```
@Grab(group='org.codehaus.gpars',
module='gpars', version='0.12')
import groovyx.gpars.GParsPool

GParsPool.withPool {
    def animals = ['dog', 'ant', 'cat',
'whale']
    println(animals.anyParallel {it ==~
/ant/} ? 'Found an ant' : 'No ants found')
    println(animals.everyParallel
{it.contains('a')} ? 'All animals contain a'
: 'Some animals can live without an a')
}
```

Run the script and you should get the following output:

```
Found an ant
Some animals can live without an a
```

Now feel free to experiment changing the regular expressions, using different collections or different methods, like eachParallel(), collectParallel(), maxParallel(), sumParallel() and others. You get the idea, right?

To find out more about parallel collection processing, visit the Parallel Colections section of the User Guide.

Step 4 - Actors

Now we could try to build an actor and send it a couple of messages to see it acting.

```
@Grab(group='org.codehaus.gpars',
module='gpars', version='0.12')
import
groovyx.gpars.actor.DynamicDispatchActor
```

```
import
org.codehaus.groovy.runtime.NullObject

final class MyActor extends
DynamicDispatchActor {
    private int counter = 0

    void onMessage(String message) {
        counter += message.size()
        println 'Received string'
    }

    void onMessage(Integer message) {
        counter += message
        println 'Received integer'
    }

    void onMessage(Object message) {
        counter += 1
        println 'Received object'
    }

    void onMessage(NullObject message) {
        println 'Received a null object.
Sending back the current counter value.'
        reply counter
    }
}

final def actor = new MyActor()
actor.start()
```

```
actor.send 1
actor << 2
```

```
actor 20
actor 'Hello'
println actor.sendAndWait(null)
```

Our actor maintains a private counter and accepts different types of messages, which result in updating the counter. Sending a null value will make the actor reply the current counter value back to us. Notice the *send()* method name is optional and can be replaced by the << operator or ommited altogether.

The Actor section of the User Guide will help you dive deeper into GPars actors.

Further steps

Now when you have GPars runing on your system, the time is up you opened up the User Guide, browsed the GPars code examples and continued experimenting. You may also consider checking out the Java Fast Track, in case you need to use GPars high-level concurency abstractions from Java code. Good luck!