

# Groovy Fast Track

Russell Winder

Version 1.0, 2015-10-01

# Table of Contents

|  |   |
|--|---|
| Step 1 — Start up the Groovy Console .....                   | 2 |
| Step 2 — Add dependencies .....                              | 3 |
| Step 3 — Experiment with parallel collection processing..... | 4 |
| Step 4 — Actors .....  | 5 |
| Further steps .....  | 7 |

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 distributions so this step should normally be not required.*

We'll use Groovy's *Grape* functionality to grab all the required dependencies for us. You may check out the [GPars Integration page](#) for other ways to integrate **GPars** with your project.

Add the following line to the groovy script:

*Use the Grape Grab Feature*

```
@Grab(group='org.codehaus.gpars', module='gpars', version='1.2.0')
```

## 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:

*Groovy Concurrent Sample*

```
@Grab(group='org.codehaus.gpars', module='gpars', version='1.2.0')
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:

1. Found an ant
2. 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 Collections 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.

### *An Actor Sample*

```
@Grab(group='org.codehaus.gpars', module='gpars', version='1.2.0')
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 omitted altogether.

The [Actor section](#) of the [User Guide](#) will help you dive deeper into **GPars** actors.



# Further steps

Now when you have **GPars** running on your system, now's the time to open up the [User Guide](#), browse the **GPars** code examples and continue experimenting.

You may also consider checking out the [Java Fast Track](#), in case you need to use **GPars** high-level concurrency abstractions from Java code. Good luck!