# Asynchronous functions

GPars provides several ways to run tasks in the background asynchronously.

## Usage of Asynchronous functions

### Closures enhancements

Groovy provides a lot of the infrastructure enabling efficient functional style of programming. Closures can be stored in variables, passed around as parameters and return values, composed, memoized, trampolined or applied partially. GPars adds asynchronicity to the mix. Closures can now be either synchronous or asynchronous, or, may you wish so, even both at the same time. You can mix synchronous and asynchronous closures in a single calculation without restrictions.

Examples:

```
GParsExecutorsPool.withPool {
    /**
     * The callAsync() method is an asynchronous variant
     * of the default call() method to invoke a closure.
     * It will return a Future for the result value.
     */
    assert 6 == {it * 2}.call(3)
    assert 6 == {it * 2}.callAsync(3).get()
}
```

```
GParsPool.withPool {
    Closure longLastingCalculation = {calculate()}

    //create a new closure, which starts the original closure on a thread pool
    Closure fastCalculation = longLastingCalculation.asyncFun()

    //returns almost immediately
    Promise result=fastCalculation()

    //do stuff while calculation performs …
    ...

    //finally ask for the result, blocking, if not yet available
    println result.get()
}
```

For more details on the Asynchronous computations please visit the Asynchronous Invocation section of the User Guide .