Writing libraries

At the base level, any valid Groovy code is okay for use. Different data structures, utility methods, etc, such as:

```
// src/org/foo/Point.groovy
package org.foo;

// point in 3D space
class Point {
  float x,y,z;
}
```

Accessing steps

Library classes cannot directly call steps such as `sh` or `git`. They can however implement methods, outside of the scope of an enclosing class, which in turn invoke Pipeline steps, for example:

```
// src/org/foo/Zot.groovy
package org.foo;

def checkOutFrom(repo) {
  git url: "git@github.com:jenkinsci/${repo}"
}
```

Which can then be called from a Scripted Pipeline:

```
def z = new org.foo.Zot()
z.checkOutFrom(repo)
```

This approach has limitations; for example, it prevents the declaration of a superclass.

Alternately, a set of `steps` can be passed explicitly using `this` to a library class, in a constructor, or just one method:

```
package org.foo
class Utilities implements Serializable {
  def steps
  Utilities(steps) {this.steps = steps}
  def mvn(args) {
    steps.sh "${steps.tool 'Maven'}/bin/mvn -o ${args}"
  }
}
```

When saving state on classes, such as above, the class **must** implement the `Serializable` interface.