GLU Script

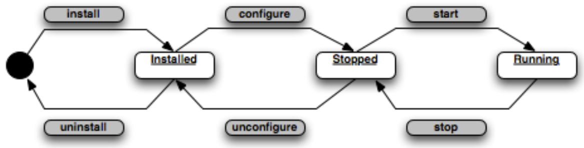
Introduction
A GLU script is a set of instructions backed by a state machine that the agent knows how to run. In general, and by default, a GLU script represents the set of instructions which defines the lifecycle of what it means to install, configure, start, stop, unconfigure and uninstall an application.

Groovy Class
```
class MyGluScript
{
  def port
  def pid

  def install     = { /* install code */ }
  def configure   = { /* configure code */ }
  def start       = { /* start code */ }
  def stop        = { /* stop code */ }
  def unconfigure = { /* unconfigure code */ }
  def uninstall   = { /* uninstall code */ }
}
```

A GLU script is a groovy class which contains a set of closures where the name of each closure matches the name of the actions defined in the state machine. This example shows the default closure names. The script can also store state in attributes (like port and pid in this example). The code of each closure can be any arbitrary groovy/java code but remember that the agent offers some capabilities to help you in writing more concise code.

State machine
Each GLU script is backed by a state machine which is an instance of `org.linkedin.groovy.util.state.StateMachine`. The default state machine is the following:



This is how the default state machine is defined.
```
def static DEFAULT_TRANSITIONS =
[
  NONE: [[to: 'installed', action: 'install']],
  installed: [[to: 'stopped', action: 'configure'], [to: 'NONE', action: 'uninstall']],
  stopped: [[to: 'running', action: 'start'], [to: 'installed', action: 'unconfigure']],
  running: [[to: 'stopped', action: 'stop']]
]
```

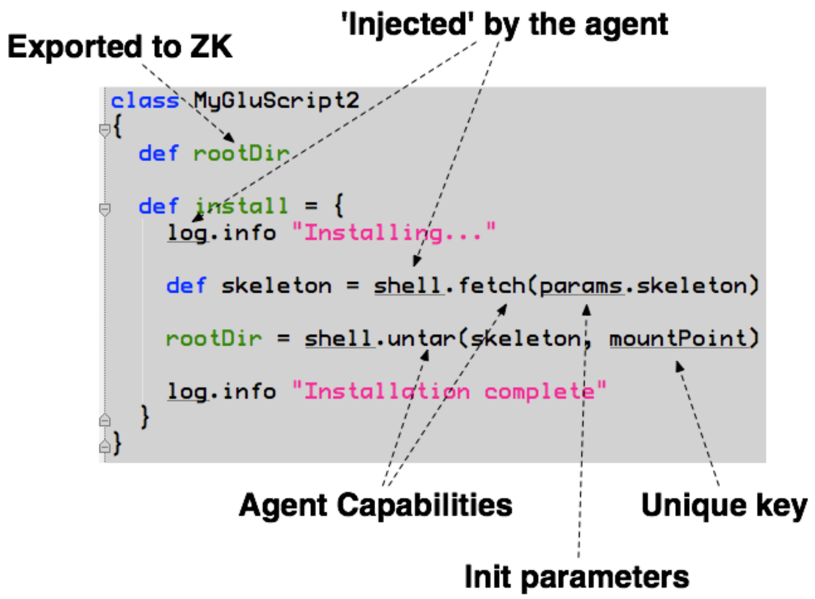
You can define your own state machine by defining a static attribute called `stateMachine`

This is how the `AutoUpgradeScript` GLU script defines different states and actions:
```
class AutoUpgradeScript {
  def static stateMachine =
  [
      NONE: [[to: 'installed', action: 'install']],
      installed: [[to: 'NONE', action: 'uninstall'], [to: 'prepared', action: 'prepare']],
      prepared: [[to: 'upgraded', action: 'commit'], [to: 'installed', action: 'rollback']],
      upgraded: [[to: 'NONE', action: 'uninstall']]
  ]
…
}
```

The minimum (usefull) state machine that you can define could look like:
```
  def static stateMachine =
  [
      NONE: [[to: 'running', action: 'start']],
      running: [[to: 'NONE', action: 'stop']]
  ]
}
```

Note: If an action is empty you don't even have to define its equivalent action but you still need to call all prior actions to satisfy the state machine.

An example of GLU script



```
class MyGluScript2
{
  def rootDir

  def install = {
    log.info "Installing..."

    def skeleton = shell.fetch(params.skeleton)

    rootDir = shell.untar(skeleton, mountPoint)

    log.info "Installation complete"
  }
}
```