



UCC Scripting With Groovy

UNICORE Team
unicore-support@lists.sourceforge.net

November 23, 2009

Contents

1 Introduction	1
1.1 A first script	2
1.2 Introspection	3
1.3 Command Line	4
2 Storages And Files	4
2.1 Storages	4
2.2 File Transfers	7
3 Jobs	7
3.1 Accessing Jobs	7
3.2 Submitting Jobs	8

1 Introduction

UCC can execute Groovy scripts. Groovy¹ is a dynamic scripting language similar to Python or Ruby, but very closely integrated with Java. The scripting facility can be used for automation tasks or implementation of custom commands.

To use and write Groovy scripts, nothing more than a running UCC installation and a text editor is needed. While this is clearly an advantage, Groovy scripts directly access the UCC and UNICORE 6 API, which needs a bit of insight into how UNICORE 6 and UCC work. For a higher level API to UNICORE 6, you might want to have a look at HILA.²

This document gives a brief introduction the the Groovy language in the context of UCC scripting. It is aimed at an audience already familiar with programming in other languages.

¹<http://groovy.codehaus.org>

²<http://www.unicore.eu/community/development/hila-reference.pdf>

variable	description	Java Type
registry	A preconfigured client for accessing the registry	de.fzj.unicore.uas.client.IRegistryQuery
securityProperties	Security configuration (keystore, etc)	de.fzj.unicore.uas.security.IUASSecurityProperties
registryURL	the URL of the registry	java.lang.String
messageWriter	for writing messages to the user	de.fzj.unicore.ucc.MessageWriter
commandLine	the command line	org.apache.commons.cli.CommandLine
properties	defaults from the user's properties file	ava.util.Properties

Figure 1: Predefined variables

1.1 A first script

Let's see how a typical Groovy script looks like:

```

1  /*
   * list all TSSs
3  */
5  lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister(
           registry, securityProperties, messageWriter)
7
9  //iterate over TSSs
   lister.each {
11     messageWriter.message it.targetSystemName
   }

```

Listing 1: List target systems

This script lists the name of all available target systems, you can run it with the command

```

$ ucc run-groovy -f myscript.groovy
Bravo-Site
Alpha-Site

```

If you don't get any output, run `ucc connect` first.

As you can see, Groovy has a similar syntax to Java, and like in Java comments are denoted by `/*` and `*/`. Note that no semicolons are needed at the end of a line, but they won't confuse Groovy either.

Things start to get interesting in line 5. Here we use the predefined variable `registry` to get a list of target systems from the global registry. Other predefined variables that you can use in your Groovy scripts are shown in Table 1. The `TargetSystemLister` filters the registry entries and returns a list of target systems accessible to us. The `ucc.helpers` package contains other useful tools, see the API documentation in the installation directory of UCC. Note that since Groovy is a dynamic language, no type declarations are needed.

In the lines 10 ff, we loop over the list of target systems and print out the name of each. The way Groovy loops over lists is a bit different from Java and other languages you may know. A list has a method `each` which takes one argument, the code block that is to be executed in each iteration. Such a code block, a closure, is given in curly braces. You can think of it as an anonymous function (lambda) if you are familiar with

the concept. The closure itself takes on argument, `it`, which holds the list element of the current iteration.

The object referenced by `it` is a `TSSClient` object, you can view its documentation at <http://www.unicore.eu/documentation/manuals/unicore6/unicorex/apidocs>. You will notice that the `TSSClient` provides lots of getters for various information. In Groovy scripts, instead of using the getter, you can access the corresponding attribute directly. This is why in line 11, instead of using `it.getTargetSystemName()`, we just wrote `it.targetSystemName`. (Note that the first letter is now lowercase.)

In case you already know Groovy, you may be wondering why we used the `message` method of the predefined `messageWriter` instead of the common `print` or `println` functions. While `message` just writes to standard out like `print`, `messageWriter` has two other useful methods: `verbose` prints output only when UCC is invoked with the verbose option (`-v`), and `error` can be used to write messages to standard error in case an exception occurred.

Just for fun, we now can rebuild UCC's `list-sites` command:

```
1 lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister (  
2     registry, securityProperties, messageWriter)  
3  
4 lister.each {  
5     messageWriter.message it.targetSystemName + " "  
6     + it.EPR.address.stringValue  
7 }  
8
```

Listing 2: List sites

The output of the above script looks like this:

```
Bravo-Site https://...:6000/Bravo-Site/services/TargetSystemService?res=...  
Alpha-Site https://...:6000/Alpha-Site/services/TargetSystemService?res=...
```

Exercise 1 *Enhance the script of listing 2 so that it prints the address of the end point reference only in case the verbose option is turned on. Additionally, the script should print the number of jobs (`numberOfJobs`) on each target system.*

1.2 Introspection

Groovy provides a few neat methods to inspect objects and classes. If you find yourself confused about what objects you are dealing with and what methods and attributes they provide, take a look at this:

```
1 messageWriter.message registry.class.name  
2 registry.properties.each { messageWriter.message it.toString() }
```

Listing 3: Inspecting a class

This gives you a very long output:

```
de.fzj.unicore.uas.client.RegistryClient  
...  
Keystore </home/rbreu/.ucc/keystore.jks>  
Truststore </home/rbreu/.ucc/keystore.jks>  
Identity: CN=Rebecca Breu,OU=JSC,OU=Forschungszentrum Juelich GmbH,...  
Signing messages: true  
Trust delegation: true  
...
```

The first line is the class of the variable `registry`. Then follows a list of all attributes of `registry` with their corresponding values. Interesting!

We can also inspect classes:

```
import de.fzj.unicore.uas.client.*
2 RegistryClient.constructors.each { messageWriter.message it.toString() }
4 RegistryClient.interfaces.each { messageWriter.message it.toString() }
RegistryClient.methods.each { messageWriter.message it.toString() }
```

Listing 4: Inspecting a class

The first thing you may notice is that we can import any Java package into a Groovy script. Of course, now we are interested in one of the UNICORE/X's client classes. You will get another long output:

```
...
public long de.fzj...BaseWSRFClient.getUpdateInterval()
public void de.fzj...BaseWSRFClient.setUpdateInterval(long)
...
```

Note the signatures of the methods of `RegistryClient`. Of course you can access those attributes via `registry.class.methods` etc. as well.

1.3 Command Line

You can access positional parameters passed to UCC via the `commandLine` variable. To get the whole list of positional parameters:

```
commandLine.argList.each { messageWriter.message it }
```

This gives the following result:

```
$ ucc run-groovy -f myscript.groovy hello world
run-groovy
hello
world
```

Note that the first positional parameter is always the name of the UCC command. Of course you can access a single element of the argument list:

```
messageWriter.message commandLine.argList[1]
```

As you see, in Groovy list indexes are given in brackets.

For evaluating single expressions, the `run-groovy` command understands the option `-e`:

```
$ ucc run-groovy -e "println(registry.class)"
class de.fzj.unicore.uas.client.RegistryClient
```

2 Storages And Files

2.1 Storages

You may have noticed that the `TSSClient` provides a list of a storages belonging to it. Let's see what we can do with those:

```

2 lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister(
                                registry, securityProperties, messageWriter)
4 lister.each {
    it.storages.each {
6         messageWriter.message it.address.stringValue
    }
8 }

```

Listing 5: List TSS storages

And here's the output:

```

https://...:6000/Bravo-Site/services/StorageManagement?res=34e26631-...
https://...:6000/Alpha-Site/services/StorageManagement?res=62594c2c-...

```

That looks a bit cryptic, but we will deal with that in a moment. Mind that in Listing 5 we only list storages belonging to target systems, like the user's home directory on that targets system. However, in a Grid you will most likely have global storages, which are listed as independent services in the global registry. So we have to ask the registry for a list of global storages. Since there is no helper available like the `TargetSystemListener` for getting a list of target systems, we have to filter the entries of the registry ourselves:

```

import javax.xml.namespace.QName
2 SMSPORT = new QName("http://unigrids.org/2006/04/services/sms",
                    "StorageManagement")
4
6 registry.listAccessibleServices(SMSPORT).each {
    messageWriter.message it.address.stringValue
}

```

Listing 6: List global storages

Note that we import the `QName` class. Now look at line 5: We use the method `listAccessibleServices` to list services of a certain port type. That's why we declared the port type for storage services back in line 2.

Now we will look at what we can do with the lists we have gathered in listings 5 and 6. What we got is a couple of end point references, which are quite boring by themselves. In order to work with the storages, we need a `StorageClient`.

```

import de.fzj.unicore.uas.client.*
2
4 def findName(epr) {
    sms = new StorageClient(epr.address.stringValue, epr, securityProperties)
    return sms.resourcePropertiesDocument.storageProperties.fileSystem.name
6 }
8
10 lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister(
                                registry, securityProperties, messageWriter)
12 lister.each {
    it.storages.each {
14         messageWriter.message findName(it) + " " + it.address.stringValue
    }
}

```

Listing 7: List storage names

Note that this script imports the UNICORE/X client classes. In line 3 we define a function which takes an end point reference of a storage service, creates a `StorageClient`

from it and returns the name of that `StorageClient`. The output of this script should now look like this:

```
Home https://...:6000/Bravo-Site/services/StorageManagement?res=34e26631...
Home https://...:6000/Alpha-Site/services/StorageManagement?res=62594c2c...
```

Exercise 2 Enhance the script from listing 7 so that it prints names and end point references of both target system storages and global storages. The function definition should turn out to be quite handy.

With the method `listDirectory` of the `StorageClient` we can get the contents of a storage:

```
import de.fzj.unicore.uas.client.*
2
3 lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister(
4         registry, securityProperties, messageWriter)
5
6 def ls(epr) {
7     sms = new StorageClient(epr.address.stringValue, epr, securityProperties)
8     name = sms.resourcePropertiesDocument.storageProperties.fileSystem.name
9     messageWriter.message "Files on " + name + ":"
10
11     sms.listDirectory(".").each { messageWriter.message it.path }
12 }
13
14 lister.each {
15     it.storages.each { ls(it) }
16 }
```

Listing 8: List files

Which gives us the following output:

```
Files on Home:
./test
./myscript.sh
./bashrc
Files on Home:
./bla
./bashrc
```

Again, only storages belonging to a target system are listed. We can easily get more information about the files, for example if we only want to list directories, we can replace line 11 of listing 8 with the following lines:

```
sms.listDirectory(".").each {
    if (it.isDirectory) { messageWriter.message it.path }
}
```

Here you can see how `if` constructs work in Groovy: The `if` condition is given in parenthesis followed by a closure, the `if` body.

Exercise 3 Write a script which lists directory contents recursively. Example output:

```
Files on Home:
./bla
./bla/hello.sh
./bla/test.txt
./myscript.sh
./bashrc
```

Exercise 4 If you know regular expressions, you can implement *find*. In Groovy, regular expressions are given between slashes, and you can match regular expressions with the `==~` operator:

```
if ("somestring" ==~ /\.*(\sh)$/ ) { ... }
```

Or, if you want to create the pattern from a string (e.g. to read it from the command line):

```
import java.util.regex.Pattern
pattern = Pattern.compile(".*(\\sh)$")
if (it.path ==~ pattern ) { ... }
```

2.2 File Transfers

Listing 9 shows how to transfer a file from the local machine to a UNICORE storage.

```
1 import de.fzj.unicore.uas.client.*
import org.w3.x2005.x08.addressing.EndpointReferenceType
3
4 url = "https://...:6000/Alpha-Site/services/StorageManagement?res=..."
5 epr = EndpointReferenceType.Factory.newInstance()
6 epr.addNewAddress().setStringValue(url)
7
8 sms = new StorageClient(url, epr, securityProperties)
9
10 ftc = sms.getImport("mytestfile.txt")
11 fis = new FileInputStream("/home/rbreu/.bashrc")
ftc.writeAllData(fis)
```

Listing 9: Put a file on a storage

Instead of looping over a list of available storages, we want to send the file to one specific storage. We address a storage by its end point reference. In lines 4 ff we create such an end point reference, you can take one of the URLs you got from executing scripts 5 or 6. In line 8 we create a `StorageClient` from the end point reference. In lines 10 ff, we create a `FileTransferClient` for file import (i.e. importing a file to the storage). Then the local file is opened and written to the storage via the `writeAllData` method.

Analogously, the relevant part for exporting a file looks like this:

```
ftc = sms.getExport("mytestfile.txt")
fos = new FileOutputStream("/tmp/hello")
ftc.readAllData(fos)
```

Exercise 5 Write a script that transfers the content of a local directory to a storage. You can get a list of local files with

```
filelister = new File(".").listFiles()
```

3 Jobs

3.1 Accessing Jobs

Accessing all your jobs is quite similar to accessing the storages belonging to target systems:

```

1 import de.fzj.unicore.uas.client.*
3 lister = new de.fzj.unicore.ucc.helpers.TargetSystemLister(
           registry, securityProperties, messageWriter)
5
6 lister.each {
7     it.jobs.each{
8         messageWriter.message "Job: " + it.address.stringValue
9         job = new JobClient(it.address.stringValue, it, securityProperties)
10        messageWriter.message "  Status      : " + job.status
11        messageWriter.message "  Exit Code   : " + job.exitCode
12        messageWriter.message "  Submission Time: " + job.submissionTime
13    }
14 }

```

Listing 10: Put a file on a storage

An example output of the above script:

```

Job: https://...:6000/Alpha-Site/services/JobManagement?res=2b86838a-...
  Status      : SUCCESSFUL
  Exit Code   : 0
  Submission Time: 2009-01-30T15:09:55.865+01:00

```

Exercise 6 Write a script which removes all jobs which have finished successfully. Use the `destroy` method of the `JobClient`. Warning: Before executing this script, make sure that you really want to remove those jobs and lose their output.

Exercise 7 The `JobClient` provides the attribute `uspaceClient`, which is a `StorageClient` to access the `Uspace`, the job's working directory. Write a script that transfers all files from a `Uspace` to the local machine.

3.2 Submitting Jobs

The simplest way to run a job is by reading the job definition from a UCC job description file:

```

1 import de.fzj.unicore.ucc.util.Builder
2 import de.fzj.unicore.ucc.helpers.Runner
3
4 builder=new Builder(new File("../date.u"))
5
6 runner = new Runner(registry, securityProperties, builder)
7 runner.run()

```

Listing 11: Run a job

The `ucc.util.Builder` helps setting up the job definition, we can pass a job description file to its constructor, see line 4. The thus-constructed job is run synchronously in line 7 with the help of `ucc.helpers.Runner`. You can also let the job run asynchronously by calling

```
runner.setAsyncMode(true)
```

before calling the `run` method.

We need not use a job description file, here is a simple date job constructed from scratch:

```
builder = new Builder()
builder.setApplication("Date", "1.0")
```

Here we don't specify any arguments for the `Builder`'s constructor but manually choose the `Date` application with version 1.0. If you don't want a specific application version, set the second parameter of the `setApplication` method to `null`.

Instead of specifying applications, you can directly specify executables and their arguments:

```
builder = new Builder()
builder.setExecutable("/bin/uname")
builder.hasArgument("-a")
```

Now consider a simple script job, which needs to import the script to be executed. Recall how the according job description file for UCC would look like:

```
{
  ApplicationName: Bash shell,
  Environment: ["SOURCE=script.sh"],
  Imports: [ {File: "/susehome/rbreu/myscript.sh", To: script.sh} ]
}
```

Building an according job in a Groovy script works quite similar, you have to specify the file import from your local machine and the environment variable `SOURCE`:

```
builder = new Builder()
builder.setApplication("Bash shell", null)
builder.setEnvironment("SOURCE", "script.sh")
builder.withImport("/home/rbreu/myscript.sh", "script.sh")
```

Not surprisingly, the method `withExport` is used in case of exporting output files your local machine.

Exercise 8 *Let your script produce an additional output file and export that file to your local machine. Also use the methods `setJobName` and `setSite` in your Groovy script.*

Exercise 9 *Simulate a parameter sweep task: Write a Groovy script which submits multiple `echo` jobs, where each `echo` job takes a different parameter. Note that in Groovy you can specify a list using the syntax `params = ["one", "two"]` and then use its `each` method to iterate over it.*

Index

application, 9
application version, 9
arguments, 9
asynchronous, 8

block of code, *see* code block
Builder, 8

closure, 2
code block, 2
command line arguments, 4
comment, 2

directory, 6

each, 2
environment variable, 9
executable, 9

file export, 7, 9
file import, 7, 9
FileTransferClient, 7
function, 5

getter, 3
global registry, 2
global storage, *see* storage
Groovy, 1
 run script, 2

hasArgument, 9

if, 6
import, 4
index, *see* list index
introspection, 3

job definition, 8
job description file, 8
JobClient, 8

list, 9
list index, 4
listAccessibleServices, 5
listDirecory, 6
loop, 2

message, 3
messageWriter, 3

numberOfJobs, 3

port type, 5
positional parameters, 4
print, 3
println, 3

registry, 2
regular expression, 7
Runner, 8

semicolon, 2
setApplication, 8
setAsyncMode, 8
setExecutable, 9
storage, 4
 global, 5
StorageClient, 5
submit job, 8
synchronous, 8

target system, 2
TargetSystemLister, 2
targetSystemName, 3
TSSClient, 3
type declaration, 2

ucc.helpers, 2
Uspace, 8

variable
 predefined, 2
verbose, 3

withExport, 9
withImport, 9