

**HOW TO
INSTALL GLOBUS
AND WRITE A
HELLO WORLD SERVICE**

Drs. Wico Mulder
Ing. A.M. Kuipers

November 2004
Version 0.22

THIS IS A WORKING DOCUMENT

Index

1	Globus 3.2 Single node binary Setup Linux RH9	3
1.1	Installing Supporting Software.....	3
1.1.1	The Java language.....	3
1.1.2	The Ant builder.....	3
1.1.3	The Junit testing framework.....	4
1.1.4	The GNU C-compiler.....	4
1.1.5	The PostgreSQL database and JDBC connection	5
1.2	The user logins.....	6
1.2.1	Adding users.....	6
1.2.2	The .bashrc login script.....	6
1.3	Installing Globus Toolkit.....	7
1.3.1	Installation of the binary.....	7
1.4	Certificate Authority and user certificates.....	7
1.4.1	Quick introduction.....	7
1.4.2	Creating a simple CA.....	8
1.4.3	Creating a certificate for your host.....	8
1.4.4	Creating a certificate for your user.....	9
1.4.5	Test your user certificate.....	10
1.4.6	Run permission script.....	10
1.4.7	Connect users to certificates.....	10
1.4.8	Start the whole thing.....	11
2	Hello world service.....	12
2.1	Start environment.....	12
2.2	Build a Grid application.....	12
2.2.1	Project introduction.....	12
2.2.2	Ant script.....	13
2.2.3	Interface template.....	13
2.2.4	Server implementation.....	13
2.2.5	First build steps.....	13
2.2.6	Building a Grid archive.....	14
2.2.7	Deployment of the grid archive.....	15
2.2.8	Build and run the Grid client.....	15
3	Grid network setup.....	16
3.1	Infrastructure.....	16
3.1.1	Servers.....	16
3.1.2	Users and groups.....	16
3.1.3	Fileserver setup.....	16
4	Resources.....	16
5	Appendix: Application interface (HelloWorld.java).....	17
6	Appendix: Application server (HelloWorldImpl.java).....	17
7	Appendix: Application client (HelloWorldClient.java).....	17
8	Appendix: Application webservice (server-deploy.wsdd).....	18
9	Appendix: Ant properties (build.properties).....	18
10	Appendix: Ant build file (build.xml).....	19

1 Globus 3.2 Single node binary Setup Linux RH9

This technical document describes the installation and development procedure of a “HelloWorld” grid service. Starting point for this document is a fresh installation of RedHat 9 workstation.

Expected is that you are the root user of this RedHat system. You will install java, a database, globus toolkit software, generate and sign certificates and develop, build and use a grid service.

1.1 Installing Supporting Software

1.1.1 The Java language

Since the underlying code of the globus toolkit (GT3) is written in Java, you have to install the Java platform on every machine running the Toolkit. The recommended version is 1.4.2, and you can download this from the Sun website at <http://java.sun.com/j2se>.

The major steps for installing Java are;

1. Install java in path /usr/local/j2sdk1.4.2_03
2. Set environment variables

Now we describe the above steps in more detail. If your installation is a bit different you might find some useful information in this document. You probably will download the software or install it from a CD-Rom.

Here we connect from our machine to another machine that has a shared (NFS) folder containing the installations that we will use. This folder is mounted first.

```
root # ping 192.168.0.1
      # mkdir /mnt/grid1
      # mount 192.168.0.1:/mnt/shared /mnt/grid1
      # cd /mnt/grid1/installs
```

The java binary for Linux is then copied and installed. Add execute rights to the bin-file if needed.

```
root # cp j2sdk-1_4_2_03-linux-i586.bin /usr/local
      # cd /usr/local
      # ./j2sdk-1_4_2_03-linux-i586.bin
      # rm j2sdk-1_4_2_03-linux-i586.bin
```

To use the Java installation, edit the shell login script, in our situation this is “~/bashrc”.

```
root # cd ~
      # vi .bashrc
```

Add a JAVA_HOME and change the PATH environment setting.

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_03
export PATH=$JAVA_HOME/bin:$PATH
```

Then you need to login again to use the new settings. And you can verify that your environment uses the correct java version.

```
root # java -version
```

1.1.2 The Ant builder

Ant is a Java-based build tool required for the GT3 installation and development of grid services. With build scripts, used by Ant, we can install, compile and copy files and applications. Ant is required for the installation of Grid services. The recommended version: 1.6.0 and can be retrieved from <http://jakarta.apache.org/ant>.

The main tasks to install Ant are;

1. Install Ant in path /usr/local/apache-ant-1.6.1
2. Set the environment variables

Here we also use the mounted folder to retrieve the tar-file containing Ant. Then it is installed.

```
root # cd /mnt/grid1/installs
      # cp apache-ant-1.6.0-bin.tar /usr/local
      # cd /usr/local
      # tar xvf apache-ant-1.6.0-bin.tar
      # rm apache-ant-1.6.0-bin.tar
```

Now Ant is installed, and you can modify the already mentioned “.bashrc” file to use Ant.

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_03
export ANT_HOME=/usr/local/apache-ant-1.6.0
export PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$PATH
```

To use these settings, login again.

Note: Some linux systems include an older Ant 1.5. Check and modify the “/etc/ant.conf” file.

1.1.3 The Junit testing framework

Junit is a Java-based testing framework that facilitates regression tests. If you run tests from source, the junit.jar application must be included with Ant for the GT3 installation.

It is required for grid services, however it is optional if you are only installing the GT3 Core component. The recommended version is 3.8.1 and can be downloaded from <http://www.junit.org>.

The main tasks for installing Junit are;

1. Install Junit in path /usr/local/junit3.8.1
2. Copy junit.jar to the \$ANT_HOME/lib or put it on the \$CLASSPATH.

In our detailed script, the zip file is retrieved from the mounted folder and unzipped.

```
root # cd /mnt/grid1/installs
      # cp junit3.8.1.zip /usr/local
      # cd /usr/local
      # unzip junit3.8.1.zip
        Archive:  junit3.8.1.zip
         creating:  junit3.8.1/
         ...
      # rm junit3.8.1.zip
        rm: remove regular file `junit3.8.1.zip'? y
```

Junit is then added to the Java class path in the “.bashrc” file;

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_03
export ANT_HOME=/usr/local/apache-ant-1.6.0
export JUNIT_HOME=/usr/local/junit3.8.1
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JUNIT_HOME
export PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$PATH
```

Login again to use these new settings.

1.1.4 The GNU C-compiler

Verify that a GNU C-compiler is installed in your RedHat 9 workstation installation. Normally this should be, but you can easily verify it;

```
root # gcc --version
```

In case you need to download this compiler, verify this link <http://www.gnu.org/software/gcc/> for GCC, but also other vendor compilers will work.

1.1.5 The PostgreSQL database and JDBC connection

The Globus grid container needs a database to store persistent information.

Here is some background information.

Name	Description
PostgreSQL	is a Java Database Connectivity (JDBC) compliant Database. Used for: the Reliable File Transfer (RFT) service and Replica Location Service (RLS), which require a database backend. For licensing reasons, we ship the PostgreSQL JDBC driver, but other JDBC compliant databases should be compatible. Download Link: http://www.postgresql.org
JDBC	is an API for Java that allows access to a wide range of SQL databases. JDBC is similar to the open standard API Open Database Connectivity (ODBC), which is aligned with The Open Group.

If you download PostgreSQL you only need 3 out of 11 rpm installation packages. Check these below.

But our PostgreSQL installation is copied from the shared folder again.

Root	<pre># cd ~ # mkdir tmp-postgres # cd /mnt/grid1/installs # cp postgresql-* ~/tmp-postgres/ # cd ~/tmp-postgres/</pre>
------	--

Only three rpm files are needed to install PostgreSQL, check their names and order of installation below.

Root	<pre># cd ~/tmp-postgres/ # ll total 5044 -rwxr-xr-x 1 root root 1987686 postgresql-7.4-0.5PGDG.i386.rpm -rwxr-xr-x 1 root root 142147 postgresql-libs-7.4-0.5PGDG.i386.rpm -rwxr-xr-x 1 root root 3016461 postgresql-server-7.4-0.5PGDG.i386.rpm # rpm -ivh postgresql-libs-7.4-0.5PGDG.i386.rpm Preparing... ##### [100%] 1:postgresql-libs ##### [100%] # rpm -ivh postgresql-7.4-0.5PGDG.i386.rpm Preparing... ##### [100%] 1:postgresql ##### [100%] # rpm -ivh postgresql-server-7.4-0.5PGDG.i386.rpm Preparing... ##### [100%] 1:postgresql-server ##### [100%] # cd .. # rm -R tmp-postgres/ rm: descend into directory `tmp-postgres/'? y rm: remove regular file `tmp-postgres//postgresql-7.4-0.5PGDG.i386.rpm'? y rm: remove regular file `tmp-postgres//postgresql-libs-7.4-0.5PGDG.i386.rpm'? y rm: remove regular file `tmp-postgres//postgresql-server-7.4-0.5PGDG.i386.rpm'? y rm: remove directory `tmp-postgres/'? y</pre>
------	--

There is a file in /etc/init.d/ that needs a slight change. Edit the line that is shown below with the grep command. And change the '-p \${PGPORT}' to -i.

root	<pre># pwd /etc/init.d # grep "postmaster -o" postgresql su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D \$PGDATA -p /usr/bin/postmaster -o '-p \${PGPORT}' start > /dev/null 2>&1" < /dev/null # vi postgresql</pre>
------	---

Verify that you changed it correctly and then you can start PostgreSQL.

root	<pre># grep "postmaster -o" postgresql su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D \$PGDATA -p /usr/bin/postmaster -o -i start > /dev/null 2>&1" < /dev/null # /etc/init.d/postgresql start Initializing database: [OK] Starting postgresql service: [OK]</pre>
------	---

Note that the “/var/lib/pgsqli/data/pg_hba.conf” file is not changed, this file is mentioned in another installation document but here we do not use it.

Important: Make sure the database is running! Otherwise it might seem ok, but later on when signing certificates things appear to go wrong.

1.2 The user logins

1.2.1 Adding users

Our work up to this point was done by the root user. To add a new user you can use a graphical tool in RedHat.

Use the RedHat start-menu and select in the System Settings the Users and Groups. On the Users tab-page click the button to add a user and type “globus” for the name and password. Select create a private group.

This will result in a user that is placed in a group with the same “globus” name.

Repeat this task for the “logicacmg” user.

1.2.2 The .bashrc login script

During the previous installations the “.bashrc” script was gradually extended. Here the final result is presented, and this script will be copied to the home directories of the other users, in our case “globus” and “logicacmg”. Note that it contains more settings than we prepared, but we need them in advance to the globus installation.

The resulting “.bashrc” script looks like;

```
# .bashrc

# User specific aliases and functions

# Globus, java, ant, junit, ssl settings
export JAVA_HOME=/usr/local/j2sdk1.4.2_03
export ANT_HOME=/usr/local/apache-ant-1.6.0
export JUNIT_HOME=/usr/local/junit3.8.1
export GLOBUS_LOCATION=/usr/local/globus
export SSLEAY_CONFIG="-config /CA/openssl.cnf"
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JUNIT_HOME
export PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$GLOBUS_LOCATION/bin:$PATH

. $GLOBUS_LOCATION/etc/globus-user-env.sh

#alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

In the example below, the user globus is logged-in. Then it switches to root without changing the current working directory (do not specify the minus) and copies the “.bashrc” file to the globus home directory. It checks to see if the user permissions and owner are OK, and they are.

```
$globus [globus@grid4 globus]$ pwd
/home/globus
[globus@grid4 globus]$ su root
#root Password:

bash: /usr/local/globus/etc/globus-user-env.sh: No such file or directory
[root@grid4 globus]#
[root@grid4 globus]# cp /root/.bashrc .
cp: overwrite `./bashrc'? y
[root@grid4 globus]# exit
$globus [globus@grid4 globus]$ ls -al .bashrc
-rw-r--r--  1 globus  globus    599 Oct 22 13:48 .bashrc
```

Copy this also to the home directory of the logicacmg user and any other user you want.

Please ignore the error message for the “globus-user-env.sh” for now, you will install the globus software in just a moment.

1.3 **Installing Globus Toolkit**

Now we are finished with the preparations and we can install the Globus Toolkit 3 (GT3) software. The process takes some time installing, but you will also be very busy generating and signing security certificates.

1.3.1 **Installation of the binary**

Ok, the root user will create a new folder and install the globus software in it. If you get a message “invalid user” at the chown command, check the previous “adding users” chapter.

```
root # pwd
      /root
      # mkdir /usr/local/globus
      # chown globus:globus /usr/local/globus
      # su - globus
```

Make sure the “.bashrc” file for the globus user is the same as the one we prepared earlier. So you will get a “no such file or directory” warning like described earlier at login or su.

The software is copied from the mounted folder, untarred and installed.

```
globus $ mkdir install
        $ cd install
        $
        $ cp /mnt/grid1/installs/gt3.2-all-linux-rh9-installer.tar /usr/local/globus/
        $ cd /usr/local/globus/
        $ tar xvf gt3.2-all-linux-rh9-installer.tar
        $ rm gt3.2-all-linux-rh9-installer.tar
          rm: remove write-protected regular file `gt3.2-all-linux-rh9-installer.tar'? y
        $ cd gt3.2-all-linux-rh9-installer/
        $ ./install-gt3-bin /usr/local/globus |tee handy-log-file-during-install.log
```

Yes, you are now installing globus - *a historic moment in your career* – and you can enjoy it for about 19 minutes. Go get a coffee, and do not worry about the messages and a few warnings too much.

Then login as root, and copy a java jar file.

```
$globus $ su -
#root Password:
        # mkdir $JAVA_HOME/jre/endorsed
        # ls $GLOBUS_LOCATION
        ...
        endorsed
        ...
        # cp $GLOBUS_LOCATION/endorsed/xalan.jar $JAVA_HOME/jre/endorsed
```

Now the software is installed and we will start with generating and signing security certificates.

1.4 **Certificate Authority and user certificates**

1.4.1 **Quick introduction**

Globus is using a security infrastructure with certificates per user, so users or processes running for this user can be identified using this certificate. A user can generate a certificate for him/herself, but to really use it, the certificate must be signed by someone else, a certificate authority (CA). You can visualize this as a hierarchy of people signing for other people.

There are commercial CA's that will sign your organisation's certificate for some cash and a prove that you are who you say you are. That signed certificate can then be used to sign users within your organisation.

For demo and easy installation purposes, the globus installation offers us a way to avoid the above procedures, but you should definitely consider an official certificate for production systems.

These steps will be done:

- Setup a CA as globus
- Generate a host certificate as root, sign by globus
- Generate a user certificate as logicacmg, sign by globus

1.4.2 Creating a simple CA

The globus installation provides a simpleCA option for demo purposes and we will use this. Normally the following should be generated once per grid.

```
globus $ $GLOBUS_LOCATION/setup/globus/setup-simple-ca
```

You have to answer some questions, and you can accept the default answers. Except for the passphrase, which is in our case "camanager". Note that you shouldn't use spaces.

This process generates a CA that is stored in;

```
/home/globus/.globus/simpleCA/globus_simple_ca_c7881362_setup-0.17.tar.gz
```

And the c7881362 is a short identification hash for your certificate.

```
globus $ su -
root # /usr/local/globus/setup/globus_simple_ca_c7881362_setup/setup-gsi -default
```

The next step is to generate and sign a certificate for your host machine.

1.4.3 Creating a certificate for your host

The name of our host "grid4" is used to request a certificate for it. Check your hostname and do the same.

```
root # echo $HOSTNAME
grid4
# grid-cert-request -host 'grid4'
```

That process generated three files in the "/etc/grid-security/" folder. These are "hostkey", a private key for your host, "hostcert_request", the certificate request for your host, and "hostcert" the signed certificate for your host. But, you see that this "hostcert" is still empty because we only generated a request yet, and did not sign our request.

Here we will sign the "hostcert_request" using the simpleCA certificate we created in the previous section. If your organisation uses an official certificate, use that one.

Sign it using the simpleCA as globus user and do not worry about the error. The signed certificate is placed somewhere else and we will copy that certificate to the correct location later. And enter the CA's passphrase as entered creating the CA.

```
globus $ cd /etc/grid-security/
$ grid-ca-sign -in hostcert_request.pem -out hostsigned.pem

Enter password for the CA key:

The new signed certificate is at: /home/globus/.globus/simpleCA//newcerts/01.pem

ERROR: Cannot create the requested output certificate file
       hostsigned.pem
Instead manually copy and rename the new signed certificate at
       /home/globus/.globus/simpleCA//newcerts/01.pem
```

Now overwrite the empty "hostcert" file with the signed host certificate request.

```
globus $ su -
root # cd /etc/grid-security/
# cp /home/globus/.globus/simpleCA//newcerts/01.pem hostcert.pem
```


Verify the permissions on the files in this folder, the “hostcert” should be owned by root and read-only for other users. The “hostkey” should be read-only for the root – no-one else.

The host certificate is now ready.

1.4.4 Creating a certificate for your user

So now we have a globus user acting as a certificate authority, a root user with the host certificate, and now we are going to create a user certificate.

Our user is named “logicacmg”, and was already added to your system in a previous section, as you did with the “.bashrc” script. If not, turn back some pages.

Login with this user and request for a certificate.

```
logicacmg $ grid-cert-request
```

This will create three files in “/home/logicacmg/.globus/”. The principle is like generating certificate for the host, but these are user certificates.

Three pem files are the key, certificate request and certificate for this user. The userkey is private for the user, so no one else is allowed to read or use this userkey. The usercert_request will be sent to the CA (globus user) to sign it. After the globus user signed the usercert_request and produced a signed certificate for the logicacmg user, globus will send the certificate back to the logicacmg user.

This signed certificate will be the public certificate for the logicacmg user.

```
logicacmg $ pwd
/home/logicacmg/.globus
logicacmg $ ll
total 8
-rw-r--r-- 1 logicacmg logicacmg 0 Oct 22 17:18 usercert.pem
-rw-r--r-- 1 logicacmg logicacmg 1412 Oct 22 17:18 usercert_request.pem
-r----- 1 logicacmg logicacmg 951 Oct 22 17:18 userkey.pem
```

Because of user permissions the globus user has to be in this “/home/logicacmg/.globus/” folder to sign the usercert_request. Again, like with signing the host certificate, there will be an error but we will copy the resulting file later ourselves. Enter the passphrase for the CA to sign the certificate.

```
logicacmg $ su globus
globus Password:
globus $ pwd
/home/logicacmg/.globus
globus $ grid-ca-sign -in usercert_request.pem -out signed.pem

Enter password for the CA key:

The new signed certificate is at: /home/globus/.globus/simpleCA//newcerts/02.pem

ERROR: Cannot create the requested output certificate file
signed.pem
Instead manually copy and rename the new signed certificate at
/home/globus/.globus/simpleCA//newcerts/02.pem
```

Now the certificate is signed and placed in the 02.pem file. This file needs to be copied to the /home/logicacmg/.globus/usercert.pem file, but this cannot be done directly by the globus or logicacmg user. So the root user is used for this. Note that you have to enter this on one single line and it will overwrite an existing empty file.

```
root cp
/home/globus/.globus/simpleCA//newcerts/02.pem
/home/logicacmg/.globus/usercert.pem
```

This way, the permissions and ownership of the file stays the same, and no `chmod` or `chown` has to be changed. This results in the `"/home/logicacmg/.globus/"` folder containing:

```
logicacmg $ ll
total 12
-rw-r--r--  1 logicacmg logicacmg  2738 Oct 28 10:09  usercert.pem
-rw-r--r--  1 logicacmg logicacmg  1412 Oct 22 17:18  usercert_request.pem
-r-----  1 logicacmg logicacmg    951 Oct 22 17:18  userkey.pem
```

You are now finished creating signed certificates.

1.4.5 Test your user certificate

To verify that you did the right thing, you can verify your certificate with your key with your CA.

```
logicacmg $ pwd
/home/logicacmg/.globus
logicacmg $ grid-proxy-init -debug -verify

User Cert File: /home/logicacmg/.globus/usercert.pem
User Key File: /home/logicacmg/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u500
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=logicacmg
Enter GRID pass phrase for this identity:
Creating proxy .....+++++++
.....+++++++
Done
Proxy Verify OK
Your proxy is valid until: Thu Oct 28 22:17:03 2004
```

1.4.6 Run permission script

This script will change permissions on globus executables.

```
root # $GLOBUS_LOCATION/bin/setperms.sh
```

1.4.7 Connect users to certificates

The root user now has to edit a file and add all users, in our case only the `"logicacmg"` user with its certificate and username. But first, let's see what should be entered in that file later.

```
logicacmg $ whoami
logicacmg
logicacmg $ grid-cert-info -subject
/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=logicacmg
```

Root can edit the file and add this certificate and username on a single (!) line.

```
root # vi /etc/grid-security/grid-mapfile
```

In the file there is one single line with a mapping from the certificate subject to the username. Enter a quoted subject, a space and the username. Note that the following line might be wrapped in your print but it is a single line.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=logicacmg"
logicacmg
```

The system is now ready to be started!

1.4.8 Start the whole thing

The database needs to be started, if it was still running you can choose to keep the database running, or stop and start it again. Here we did the latter.

root	# /etc/init.d/postgresql stop Stopping postgresql service: [OK] # /etc/init.d/postgresql start Starting postgresql service: [OK]
------	---

Then start the globus container as the globus user.

globus	\$ cd \$GLOBUS_LOCATION \$ globus-start-container [10/28/2004 10:40:31:491] org.globus.ogsa.server.ServiceContainer [run:582] INFO: Starting SOAP server at: http://127.0.0.1:8080/ogsa/services/ With the following services: http://127.0.0.1:8080/ogsa/services/core/admin/AdminService ...
--------	---

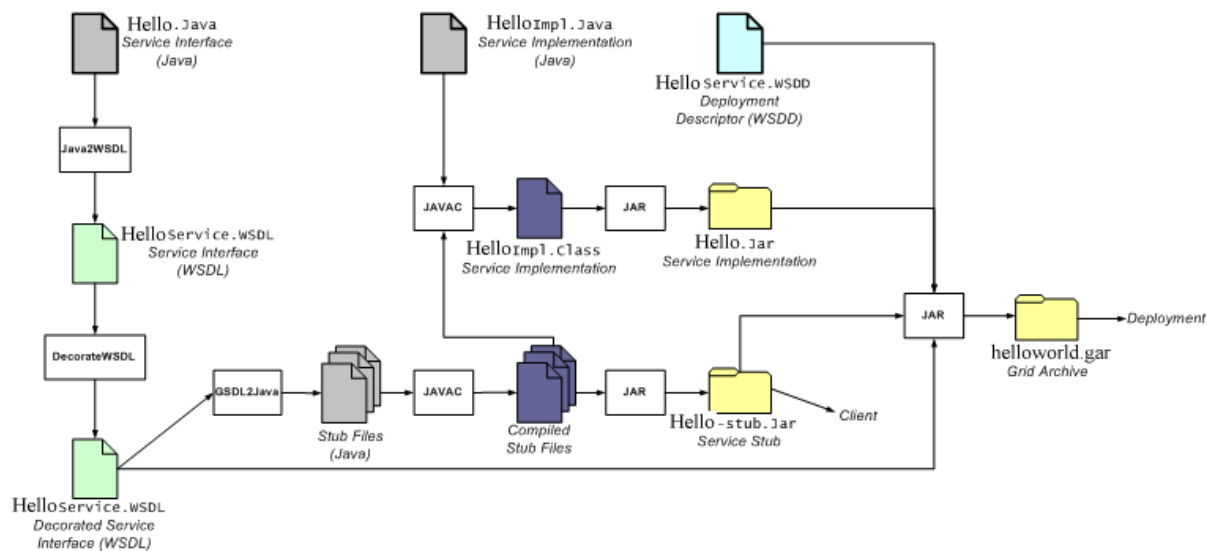
Note that the command prompt will not return. So you might want to run the command with an ampersand (&). But if you leave it like this, you can also easy stop it with control-C and start it.

Ok, we are ready to develop a Grid service!

2 Hello world service

This chapter describes the creation of a Grid service and Grid client. We will concentrate more on how this is done than what we are building. But in case you want to know it; the client will build a connection to the grid service and call a method that gives a response.

Do not be afraid by the following picture from Borja Sotomayors programmers guide (v.0.2.2), we will use a script to build most of these files.



Note that in this chapter some names are slightly different than in the picture.

The major steps are;

- Write software (interface, server and client),
- Write build script,
- Build and run

2.1 Start environment

Like we did in the first chapter we have to start the database and the globus container if they are not already started.

root	# /etc/init.d/postgresql start ...
globus	\$ cd \$GLOBUS_LOCATION \$ globus-start-container ...

2.2 Build a Grid application

2.2.1 Project introduction

To build this example, several tasks need to be done of which the first one is the creation of a folder structure and some files. You can find these files in the appendix. All are relative to the project folder located in the path “/home/logicacmg/dev/HelloProject”.

```
build.properties
build.xml
interface-source/hello/HelloWorld.java
server-source/hello/HelloWorldImpl.java
wsdd-source/server-deploy.wsdd
client-source/helloc/HelloWorldClient.java
```

We will work with these files in the rest of this chapter.

2.2.2 Ant script



The Apache Ant tool is used for building our project. We created two build files for our project, a property file and a script-like file. These are the build.properties and build.xml files you can find in the appendix.

Create these two files now, see appendix.

2.2.3 Interface template

An interface template is written with one simple method. This interface is used as a basis to generate Grid service files. That will be done by the Ant scripts later.

Create the “interface-source/hello/HelloWorld.java” file, see appendix.

2.2.4 Server implementation

The interface template created in the previous section can be seen as an agreement between server and client implementations. Our Ant build script will use this template to generate a new java interface class, named HelloWorldPortType.java. This is the stub and we have to import and implement this class in our server implementation that does the work for our Grid service.

Create the “server-source/hello/HelloWorldImpl.java” server implementation, see appendix.

The HelloWorldImpl class extends a generic grid service GridServiceImpl and implements our generated stub interface HelloWorldPortType.

2.2.5 First build steps

Now we are ready to run our first steps in building the example. The first step is creating a file structure besides the structure we created ourselves.

```
logicacmg $ cd
           $ cd dev/HelloWorld/
           $ ant step0
```

Then we generate the grid webservice (gwsdl) file.

```
logicacmg $ ant step1
```

This gwsdl is generated using Java2WSDL, an axis tool. The resulting file is not a normal wsdl file, but it is decorated with information specifically for Grid purposes. You might want to edit the gwsdl file manually.

The next step generates the stub and skeleton classes using the GSDL2Java tool.

```
logicacmg $ ant step2
```

2.2.6 Building a Grid archive

"Up to this point, we have written all the Java code our Grid Service needs. We have (1) a service interface, (2) a WSDL file, (3) a bunch of stub files, and (4) an implementation of the service. Now, we somehow have to put all these pieces together, and make them available through a Grid Services enabled web server! This step is called the deployment of the Grid Service."

The most important element of deployment is the deployment descriptor. That file contains information about publishing the service on the grid, for example, it contains the URL to access the service.






Create the "wsdd-source/server-deploy.wsdd" file, see appendix.

Build the gar file.






























```
logicacmg $ ant step2
```

The Grid archive (gar) file is a kind of jar file containing the following files;

File: hellogrid.gar (contents might have a different name)

 Hello.jar	
 HelloService.wsdl	schema\hellogrid\Hello\
 Hello-stub.jar	
 Manifest.mf	meta-inf\
 server-deploy.wsdd	

File: hello-stub.jar (contents might have a different name)

 _add.class	org\gridforum\ogsi\
 _addResponse.class	org\gridforum\ogsi\
 _createService.class	org\gridforum\ogsi\
 _createServiceResponse.class	org\gridforum\ogsi\
 _deliverNotification.class	org\gridforum\ogsi\
 _destroy.class	org\gridforum\ogsi\
 _destroyResponse.class	org\gridforum\ogsi\
 _findByHandle.class	org\gridforum\ogsi\
 _findByHandleResponse.class	org\gridforum\ogsi\
 _findServiceData.class	org\gridforum\ogsi\
 _findServiceDataResponse.class	org\gridforum\ogsi\
 _remove.class	org\gridforum\ogsi\
 _removeResponse.class	org\gridforum\ogsi\
 _requestTerminationAfter.class	org\gridforum\ogsi\
 _requestTerminationAfterResponse.class	org\gridforum\ogsi\
 _requestTerminationBefore.class	org\gridforum\ogsi\
 _requestTerminationBeforeResponse.class	org\gridforum\ogsi\
 _sayHello.class	hellogrid\Hello\
 _sayHelloResponse.class	hellogrid\Hello\
 _setServiceData.class	org\gridforum\ogsi\
 _setServiceDataResponse.class	org\gridforum\ogsi\
 _subscribe.class	org\gridforum\ogsi\
 _subscribeResponse.class	org\gridforum\ogsi\
 HelloPortType.class	hellogrid\Hello\
 HelloService.class	hellogrid\Hello\
 HelloServiceGridLocator.class	hellogrid\Hello\
 HelloServiceLocator.class	hellogrid\Hello\
 HelloServiceSoapBindingStub.class	hellogrid\Hello\
 Manifest.mf	meta-inf\

File: hello.jar (contents might have a different name)

```
HelloImpl.class      hellogrid\
Manifest.mf          meta-inf\
```

File: hello-client.jar (contents might have a different name)

```
HelloClient.class
Manifest.mf          meta-inf\
```

2.2.7 Deployment of the grid archive

After the development and build steps, the gar can be deployed. Now, login as the globus user, go to the /usr/local/globus/ folder and enter;

```
globus $ cd /usr/local/globus
        $ chmod o+w gars
```

Then the logicacmg user is able to copy the file to the globus folder.

```
logicacmg $ cp lib/hellogrid.gar /usr/local/globus/gars
```

And now it is possible to deploy the gar file using the globus user.

```
globus $ cd $GLOBUS_LOCATION
        $ ant deploy -Dgar.name=/usr/local/globus/gars/hellogrid.gar
```

Note that globus is using different Ant build scripts than the one we were using earlier.

We have to stop and start the grid container. So, go to the terminal where the grid container was started. Stop with control-C and start the grid by entering the previous command.

Our Grid service is now deployed and running!

2.2.8 Build and run the Grid client

The next step is to build the Grid client software and run it.

Create the “client-source/helloc/HelloWorldClient.java” file, see appendix.

This client uses generated classes to find and use the Grid service. Build the client;

```
logicacmg $ ant step3
```

Now you might want to change the “build.xml” file. It contains the URL of the Grid service.

```
<arg value="http://127.0.0.1:8080/ogsa/services/hello/HelloWorldService"/>
```

And then run the client.

```
logicacmg $ ant run.client
```

Congratulations!

3 Grid network setup

This chapter contains a short description about our “minigrd” Grid network. It needs some verification.

3.1 Infrastructure

3.1.1 Servers

The Grid network contains three servers running Globus and one server running file services and a Tomcat webserver. Also some laptop clients can be connected to this grid to use Grid services.

Name	IP	Description
Grid1	192.168.0.1	The NFS file server, CA manager and Tomcat web-container
Grid2	192.168.0.2	Globus framework + grid services
Grid3	192.168.0.3	Globus framework + grid services
Grid4	192.168.0.4	Globus framework + grid services

The grid1 server can be connected to the LogicaCMG intranet using its second network adapter and a dynamic IP using DHCP.

We use the name “guest” for the laptop that can be connected to this grid network.

3.1.2 Users and groups

Machine	userID	groupID	Password	Activities
grid1,grid2,grid3	Root	root		Super user needs
grid2,grid3	globus	globus	globus	Globus Toolkit environment
grid2,grid3	logiacmg	logiacmg	logiacmg	End-user environment

3.1.3 Fileserver setup

To make it easier for repeated globus installations we created a shared NFS folder called '/mnt/shared/installs/' on grid1. Read 'http://nfs.sourceforge.net/nfs-howto/intro.html' for more information to create a NFS share.

At grid1, server-side, there are three configuration files used for NFS. These are /etc/exports, /etc/hosts.allow and /etc/hosts.deny. Then the service needs to be started, use the start-nfs-daemons script

At client side you create a folder and mount the shared NFS folder to it.

root	# mkdir /mnt/grid1 # mount 192.168.0.1:/mnt/shared/ /mnt/grid1
------	---

In case you would like to start this mount automatically at a boot, add the following line to the /etc/fstab.

'10.16.0.1:/mnt/shared /mnt/grid1 nfs ro 0 0'

```
rpc.portmap, rpc.mountd, rpc.nfsd, rpc.statd, rpc.lockd, rpc.rquotad  
exportfs -ra
```

4 Resources

You can find everything and a lot more at www.globus.org

files	j2sdk-1_4_2_03-linux-i586.bin apache-ant-1.6.0-bin.tar junit3.8.1.zip postgresql-libs-7.4-0.5PGDG.i386.rpm postgresql-7.4-0.5PGDG.i386.rpm postgresql-server-7.4-0.5PGDG.i386.rpm gt3.2-all-linux-rh9-installer.tar
-------	---

5 Appendix: Application interface (HelloWorld.java)

```
// HelloWorld.java
// Interface used for generating

package hello;

public interface HelloWorld {
    public String sayHello(String name);
}
```

6 Appendix: Application server (HelloWorldImpl.java)

```
// HelloWorldImpl.java

package hello;
import hello.HelloWorld.HelloWorldPortType;
import java.rmi.RemoteException;
import org.globus.ogsa.impl.ogsi.GridServiceImpl;

public class HelloWorldImpl extends GridServiceImpl
    implements HelloWorldPortType {
    private int n; // to demo a state

    // Constructor
    public HelloWorldImpl() {
        super("a HelloWorld service");
    }

    // Interface implementation
    public String sayHello(String name) throws RemoteException {
        n++;
        return "Hello "+name+"! (for the "+n+" time)";
    }
}
```

7 Appendix: Application client (HelloWorldClient.java)

```
// HelloWorldClient.java

package helloc;
import hello.HelloWorld.HelloWorldPortType;
import hello.HelloWorld.HelloWorldServiceGridLocator;
import java.net.URL;

public class HelloWorldClient {

    // Static
    public static void main(String[] args) {
        //
        try {
            // Use command line arg
            URL GSH = new java.net.URL(args[0]);
            String name="Your Highness"; // args[1];
            // Get a reference to the Service instance
            HelloWorldServiceGridLocator helloworldLocator =
                new HelloWorldServiceGridLocator();
            HelloWorldPortType helloworld =
                helloworldLocator.getHelloWorldService(GSH);
            // Call remote method
            String result = helloworld.sayHello(name);
            System.out.println (result);
        } catch (Exception e) {
            System.out.println("Exception:\n");
            e.printStackTrace();
        }
    } // main
}
```

8 Appendix: Application webservice (server-deploy.wsdd)

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <!--
  service name=xxx
  This is where you define the remotely accessible name for your
  service. Your service factory URL will have the form of
  [hosting environment URL]/xxx, where hosting environment
  URL typically is http://[host]:[port]/ogsa/services
  -->
  -->
  <service name="hello/HelloWorldService" provider="Handler"
    style="wrapped">
    <parameter name="name"
      value="HelloWorldService"/>
    <parameter name="className"
      value="hello.HelloWorld.HelloWorldPortType"/>
    <parameter name="baseClassName"
      value="hello.HelloWorldImpl"/>
    <parameter name="schemaPath"
      value="schema/hello/HelloWorld/HelloWorldService.wsdl"/>

    <!-- Start common parameters -->
    <parameter name="allowedMethods" value="*/>
    <parameter name="persistent" value="true"/>
    <parameter name="handlerClass"
      value="org.globus.ogsa.handlers.RPCURIProvider"/>
  </service>
</deployment>
```

9 Appendix: Ant properties (build.properties)

```
// --- build.properties
// --- A build properties file for ant, used for globus service
// --- Some settings are strange, but needed...

// PROJECT SOURCE FOLDERS
// these contain server and client packages and sources
project.server=./server-source
project.client=./client-source
project.interface=./interface-source/${package}
project.wsdd=./wsdd-source

// GENERIC SETTINGS FOLDERS
// service/
// service/src/
// service/src/interface/hellogrid/
// service/src/implementation/
// service/build/
// service/build/interface
// service/build/implementation
// lib/
// etc/
service.dir=./service
service.src.dir=${service.dir}/src
service.src.interface.dir=${service.src.dir}/interface/${package}
service.src.implementation.dir=${service.src.dir}/implementation
service.build.dir=${service.dir}/build
service.build.interface.dir=${service.build.dir}/interface
service.build.implementation.dir=${service.build.dir}/implementation
lib.dir=./lib
etc.dir=./etc

// SETTINGS
ogsa.root=/usr/local/globus/
java.interface=true

// CUSTOM PROJECT SETTINGS
```

```

// interface.name=Java class containing the interface definition
// package=The package containing interface.name
// services.namespace=Use the same as package, HelloWorldPortType is produced in
there
package=hello
interface.name=HelloWorld
services.namespace=hello
// amk:added
service.name=Hello

// SETTINGS
// /usr/local/globus/schema
// /usr/local/globus/build-services.xml
// /usr/local/globus/build-packages.xml
//
schema.origin=${ogsa.root}/schema
build.services=${ogsa.root}/build-services.xml
build.packages=${ogsa.root}/build-packages.xml
mapping.file=namespace2package.mappings
src.dir=${service.src.interface.dir}
build.lib=${lib.dir}
build.dest=${service.build.interface.dir}

schema.dest=./schema
// here the .java files relative to the schema dest are planned:
stubs.dest=./stubs
//here the compiled stubs will come
service.build.stubs.dir=${service.build.dir}/stubs

// CUSTOM PROJECT SETTING
// name of the grid archive file
gar.filename=helloworld

// CUSTOM PROJECT SETTING
// locations of the grid client
client.dir=./client
client.mainclass=helloc/HelloWorldClient
client.src.dir=${client.dir}/src
client.build.dir=${client.dir}/build
client.lib.dir=${client.dir}/lib

```

10 Appendix: Ant build file (build.xml)

```

<?xml version="1.0"?>
<!-- build.xml file for ant -->

<project name="HelloWorldGridService" default="all" basedir=".">

  <property file="build.properties"/>
  <property file="${ogsa.root}/ogsa.properties"/>

  <path id="classpath">
    <pathelement location="${java.home}/../lib/tools.jar"/>
    <pathelement location="."/>
    <fileset dir="${lib.dir}">
      <include name="*.jar"/>
    </fileset>

    <fileset dir="${ogsa.root}/lib">
      <!-- BORJA
      The following line should fix the "why don't my stubs get generated??"
      problem
      when you try to recompile a service that is already deployed -->
      <include name="*.jar"/>
      <exclude name="${interface.name}-stub.jar"/>
    </fileset>
  </path>

  <target name="clean" description="cleans everything except srcs">
    <delete dir="${service.build.dir}"/>
    <delete dir="${client.build.dir}"/>
    <delete dir="${schema.dest}"/>
    <delete dir="${lib.dir}"/>
  </target>

```

```

<delete dir="${service.dir}"/>
<delete dir="${client.dir}"/>
<delete dir="./tmp"/>
</target>

<target name="prepare" description="creates the necessary directories and files">
  <echo message="package: ${package}"/>
  <echo message="interface.name: ${interface.name}"/>
  <echo message="service.name: ${service.name}"/>

  <!--
    <echo message="*** The following dirs must already be there: ***"/>
    <echo message="${service.dir}"/>
    <echo message="${client.dir}"/>
    <echo message="${src.dir} (containing the interface)"/>
    <echo message="${service.src.dir} (containing the service implementation)"/>
    <echo message="${client.src.dir}"/>
  -->

  <echo message="Copying interface source to ${service.src.interface.dir}"/>
  <mkdir dir="${service.src.interface.dir}"/>
  <copy toDir="${service.src.interface.dir}">
    <fileset dir="${project.interface}"/>
  </copy>

  <echo message="Copying server source to ${service.src.implementation.dir}"/>
  <mkdir dir="${service.src.implementation.dir}"/>
  <copy toDir="${service.src.implementation.dir}">
    <fileset dir="${project.server}"/>
  </copy>

  <echo message="Copying client source to ${client.src.dir}"/>
  <mkdir dir="${client.src.dir}"/>
  <copy toDir="${client.src.dir}">
    <fileset dir="${project.client}"/>
  </copy>

  <mkdir dir="${lib.dir}"/>
  <mkdir dir="${service.build.dir}"/>
  <mkdir dir="${service.build.interface.dir}"/>
  <mkdir dir="${service.build.stubs.dir}"/>
  <mkdir dir="${service.build.implementation.dir}"/>

  <!-- client stuff -->
  <mkdir dir="${client.build.dir}"/>
  <mkdir dir="${client.lib.dir}"/>

  <!-- SEBASTIEN
    Create build tree and copy all OGSA/GT3 schema files over so that
    they can be accessed from our GWSDL files.
    Explicitly exclude the schema matching the service being built, so
    that an old version is not copied over.
  -->
  <copy toDir="${schema.dest}">
    <fileset dir="${schema.origin}">
      <exclude name="**/${service.name}/**"/>
    </fileset>
  </copy>
</target>

<!-- shortcuts !! -->
<target name="step0" depends="clean,prepare" description="shortcut for clear and
prepare"/>
<target name="step1" depends="generateWSDLfromJava" description="shortcut for task
'generateWSDLfromJava'"/>
<target name="step2" depends="gar" description="shortcut for task 'gar'"/>
<target name="step3" depends="compile.client" description="shortcut for task
'compile.client'"/>

<target name="generateWSDLfromJava" description="provides the service interface
WSDL file">
  <echo message="Using ${interface.name}.java from ${src.dir}"/>
  <property name="schema.file" value="${interface.name}Service.wsdl"/>
  <ant antfile="${build.services}" target="generateWSDL">
    <property name="interface" value="${interface.name}.java"/>
    <property name="interface.package" value="${package}"/>
  </ant>

```

```

    <!-- <property name="interface.dir" value="\${src.dir}/\${package}"/> -->
    <property name="interface.dir" value="\${src.dir}"/>
    <property name="generated.dir" value="\${package}"/>
  </ant>
</target>

<!-- Generate, compile and jar stubs. -->

<target name="generateStubs">
  <echo message="generateStubs. Destination relative to schema dir."/>
  <ant antfile="\${build.services}"
    target="generateStubs">
    <property name="schema.file.dir" value="\${package}/\${interface.name}"/>
    <property name="schema.file" value="\${interface.name}Service.wsdl"/>
    <reference refid="classpath"/>
  </ant>
</target>

<target name="compileStubs" depends="generateStubs">
  <javac srcdir="\${schema.dest}/\${stubs.dest}"
    destdir="\${service.build.stubs.dir}"
    debug="\${debug}"
    deprecation="\${deprecation}"
    classpathref="classpath">
  </javac>
  <jar jarfile="\${lib.dir}/\${interface.name}-stub.jar"
    basedir="\${service.build.stubs.dir}" />
</target>

<!--
  Compile and jar service.
-->

<target name="compileService" depends="compileStubs">
  <echo message="Compile the service itself,
    srcdir="\${service.src.implementation.dir} to "\${service.build.implementation.dir}"/>
  <javac srcdir="\${service.src.implementation.dir}"
    destdir="\${service.build.implementation.dir}"
    debug="\${debug}"
    deprecation="\${deprecation}"
    classpathref="classpath">
  </javac>
  <jar jarfile="\${lib.dir}/\${interface.name}.jar"
    basedir="\${service.build.implementation.dir}" />
</target>

<!-- Gar stubs and service compileStubs,compileService-->

<target name="gar" depends="compileStubs,compileService">
  <copy toDir="\${service.dir}">
    <fileset dir="\${project.wsdd}">
    </fileset>
  </copy>
  <!-- IMPORTANT: make gar.name is an absolute path -->
  <ant antfile="\${build.packages}" target="makeGar">
    <property name="gar.name" value="\${lib.dir}/\${gar.filename}.gar"/>
    <property name="garlib.dir" value="\${lib.dir}"/>
    <property name="garserverdeployment.file" value="\${service.dir}/server-
    deploy.wsdd"/>
    <property name="garschema.origin" value="\${schema.dest}/\${package}"/>
    <property name="garschema.path" value="\${package}"/>
    <property name="build.lib" value="\${lib.dir}"/>
  </ant>
</target>

<!-- CLIENT -->

<path id="client.classpath">
  <pathelement location="\${java.home}/../lib/tools.jar"/>
  <pathelement location="."/ />
  <fileset dir="\${lib.dir}">
    <include name="*.jar"/>
  </fileset>

```

```

<fileset dir="${ogsa.root}/lib">
  <include name="*.jar"/>
  <!-- <exclude name="${interface.name}-stub.jar"/> -->
</fileset>
</path>

<target name="compile.client" description="compiles the (test) client">
  <javac srcdir="${client.src.dir}"
        destdir="${client.build.dir}"
        debug="${debug}"
        deprecation="${deprecation}"
        classpathref="client.classpath">
  </javac>
  <copy toDir="${client.build.dir}">
    <fileset dir="${client.src.dir}">
      <include name="**/*.properties"/>
    </fileset>
  </copy>
  <jar jarfile="${lib.dir}/${interface.name}-client.jar"
        basedir="${client.build.dir}"/>
</target>

<target name="run.client" description="runs the (test) client">
  <java classname="${client.mainclass}" classpathref="client.classpath"
        fork="yes">
    <!-- <jvmarg value="-Dlog4j.configuration=t.t"/> -->
    <arg value="http://127.0.0.1:8080/ogsa/services/hello/HelloWorldService"/>
  </java>
</target>

<target name="all" depends="step0,step1,step2,step3" description="does it all">
  <echo message="*** complete HelloWorld project ***"/>
</target>

</project>

```

<end of document>