



## WG-7 GAT Software components and documentation<sup>1</sup>

Deliverable	7.4
Authors	Alexander Beck-Ratzka
Date	June 14, 2007
Document Version	1.0
Current Version	1.0
Previous Versions	0.1

### A: Status of this Document

Final version.

### B: Reference to project plan

This deliverable refers to the task TA VII-4 "*GAT Software components and documentation*".

### C: Abstract

This document describes the installation of JavaGAT, and the usage of the extensions of JavaGAT developed within the AstroGrid project.

---

<sup>1</sup>This work is part of the AstroGrid-D project and D-Grid. The project is funded by the German Federal Ministry of Education and Research (BMBF).

**D: Changes History**

<b>Version</b>	<b>Date</b>	<b>Name</b>	<b>Brief summary</b>
0.1.0	11 May 2007	Alexander Beck-Ratzka	Working Draft Creation
0.1.0	14 June 2007	Alexander Beck-Ratzka	Final Version Creation

E:

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>How to get and install the Java CoG Kit and the JavaGAT</b>	<b>5</b>
2.1	The installation and configuration of the Java CoG Kit . . . . .	5
2.1.1	Download and installation . . . . .	5
2.1.2	Configuration . . . . .	5
2.2	Download and installation of JavaGAT . . . . .	6
2.2.1	The one step installation . . . . .	6
2.2.2	Separate build . . . . .	6
2.2.3	Build the supplemental AstroGrid packages . . . . .	6
<b>3</b>	<b>Usage description of the command line interfaces</b>	<b>8</b>
3.1	Submitting (Grid) jobs using the GATJobRun package . . . . .	8
3.2	File operations with GATFileOps . . . . .	9
<b>4</b>	<b>How to program against the JavaGAT-API</b>	<b>11</b>
4.1	File operations on entire files with JavaGAT . . . . .	11
4.2	Job submission with the ResourceBroker class . . . . .	12

## 1 Introduction

Within the AstroGrid community project, there was mainly a need for Java-GAT, therefore the Java-GAT extensions will be described in the deliverable.

Beside the PBS-, SGE-, and SGE-DRMAA-adaptor we have developed two command line interfaces for JavaGAT:

1. **GATJobRun**: the command line interface for submitting jobs via JavaGAT;
2. **GATFileOps**: the command line interface for file operations on entire datasets via JavaGAT.

This document describes

- how to install and configure JavaGAT, and the Java Cog Kit, which is still needed for `grid-proxy-init`;
- the usage of the command line interfaces `GATJobRun` and `GATFileOps`;
- how to program against the JavaGAT-API (in two examples);

## 2 How to get and install the Java CoG Kit and the JavaGAT

### 2.1 The installation and configuration of the Java CoG Kit

#### 2.1.1 Download and installation

To download the Java CoG Kit, please go to the URL:

<http://www.cogkit.org/php/download.html>,

and select

Java Cog Kit 4.

Download it to the directory, where you like to install it later on.

For installation, please unpack the file `cog-4.1.4-bin.tar.gz` using the command

```
tar -xzf cog-4.1.4-bin.tar.gz
```

and the Java CoG Kit will be extracted into a sub-directory `cog-4.1.4`.

#### 2.1.2 Configuration

To use the Java CoG Kit properly, some configurations need to be carried out. This is possible using a Java based GUI, which can be called by the CoG-CLI script `cog-setup`, which is located in the `bin` sub-directory of the Java CoG Kit. During the configuration one is asked after the location of the certificates, the location of the proxy, the IP adress of the workstation where CoG Kit is running on, etc. . . Usually the properties are stored in the file `$HOME/.globus/cog.properties`, and it might look like this:

```
Java CoG Kit Configuration File
Thu Jul 20 07:32:32 EDT 2006
usercert=/home/ali/.globus/usercert.pem
userkey=/home/ali/.globus/userkey.pem
proxy=/tmp/x509up_u500
cacert=/home/ali/.globus/cog-certificates
tcp.port.range=20000,20100
ip=dhcp-199.aei.mpg.de
```

The IP adress specified with `ip` is used e.g. for callbacks from the remote job execution host, `tcp.port.range` specifies the range of ports, where a callback should come in.

The `bin` sub-directory of the Java CoG Kit holds scripts for submitting jobs, copying or deleting files, and also a `grid-proxy-init` command is available. So if you don't need `gsissh`, no Globus installation is required any more.

## 2.2 Download and installation of JavaGAT

JavaGAT can be checked out from:

```
svn://svn.gac-grid.org/software/gat/java-gat
```

Later on (if the next Java-GAT version is to be released, you can download it from:

```
https://gforge.cs.vu.nl/projects/javagat
```

Before installing it, extract it to a directory of your choice. The installation of JavaGAT requires a Java version of at least 1.5; we recommend SUN JDK. There are two ways to install JavaGAT:

1. installation of engine, adaptors, tests and javadoc in one step;
2. separate installation of the engine and the adaptors.

### 2.2.1 The one step installation

For the one step installation:

1. Change to the directory, where you have extracted JavaGAT to.
2. Set the environment variable `GAT_LOCATION` to point to this directory. If `GAT_LOCATION` is not set, the build process will fail!
3. Enter `ant` build the engine, the adaptors, some tests, and the documentation.

The documentation is of type javadoc, and the entry page to this documentation is

```
$GAT_LOCATION/doc/javadoc/index.html
```

The jar-files necessary for the GATengine resides in `$GAT_LOCATION/engine/lib`, those of the adaptors can be found in `$GAT_LOCATION/engine/lib`

### 2.2.2 Separate build

If you are only interested in the engine and the adaptors, you can build them separately. Before starting the build, please ensure that the environment variable `$GAT_LOCATION` points to the parent directory of JavaGAT. To build the engine, change to the directory `$GAT_LOCATION/engine`, and enter `ant`. For building the adaptors, please change to `$GAT_LOCATION/adaptors`, and again enter `ant`. That's all.

### 2.2.3 Build the supplemental AstroGrid packages

The command line interfaces **GATJobRun** and **GATFileOps**, which have been developed within the AstroGrid project, need to be build separatly. The build of the packages require a previous build of the GAT engine and the GAT adaptors.

To build the **GATJobRun** package, change to the directory

```
$GAT_LOCATION/astrogrid-packages/GATJobRun,
```

and enter

```
ant -f build-standalone.xml
```

for building it. During the build the message

```
SGE_ROOT = '/opt/SGE' found.  If empty, you can't submit jobs to SGE.  
Please set SGE_ROOT and rebuild GATJobRun, if you like to use SGE
```

is prompted. If SGE\_ROOT is not set (empty), then it is not possible to submit jobs via the SGE adaptor of JavaGAT to Sun Grid Engine. SGE\_ROOT is usually empty, if you haven't installed a Sun Grid Engine.

To build the GATFileOps package please change to

```
$GAT_LOCATION/astrogrid-packages/GATFileOps,
```

and enter

```
ant -f build-standalone.xml
```

to build it.

We recommend to have a look at the Readme files, before building the packages.

### 3 Usage description of the command line interfaces

The command line interface (CLI) packages GATJobRun and GATFileOps represent an extension to JavaGAT. They have been developed within in the AstroGrid project by the AEI in Potsdam-Golm.

#### 3.1 Submitting (Grid) jobs using the GATJobRun package

The entry point to GATJobRun is the script `gat-job-run` which is located in the directory `$GAT_LOCATION/astrogrid-packages/GATJobRun/scripts`.

It is recommended to add this directory to the `PATH` environment variable, to enable an easy usage from everywhere in your file system. If `gat-job-run` is called without arguments, it prompts a list of options and arguments:

Missing host and/or executable string

Usage: `gat-job-run [OPTIONS] hostname executable [ARGUMENTS]`

OPTIONS:

`-username [STRING]` username for security context

`-password [STRING]` password for security context

`-SubmitOnly [true/false]` submit only (true) or poll still exit

`-stdin [FILE]` path to input file

`-stdout [FILE]` path to output file

`-stderr [FILE]` path to error output file

`-prestage [FILE],...` path to prestage file(s) - comma separated

`-poststage [FILE],...` path to poststage file(s) - comma speparated

`-RB.adaptor [STRING]` force the use of a specific Resource Broker adaptor

`-RB.jobmanager [STRING]` force the use of a specific Resource Broker jobmanager

If the host where the executable shall be executed is a grid host which can only be accessed via Globus, it is necessary to create a `grid-proxy`, before using `gat-job-run` successfully. The necessary command `grid-proxy-init` is available within the Java CoG Kit (in its `bin` sub-directory). In near future it is planned that JavaGAT itself provides `grid-proxy-init`.

Some comments to the options / arguments.

- `hostname`, `executable`: The name of the execution host, and the name of the executable. It is mandatory, to enter them.
- `-SubmitOnly`: The default behaviour of `gat-job-run` is to submit a job, and to poll after the status of the job until it has finished. After the end of the job, `gat-job-run` exits. This behaviour is only useful for short test jobs. In case of long term runs, it is desired that

`gat-job-run` exits before the job has finished. This can be achieved, by setting `SubmitOnly` to `true`.

- `-RB.adaptor` Here it is possible to enter the name of the JavaGAT adaptor that shall be used for the job submission. If no adaptor is specified explicitly, JavaGAT uses the first job submission adaptor which can be invoked successfully. The name of an adaptor is obtained by the name of its jar file without the suffix `.jar`. The JavaGAT adaptors are located in `$GAT_LOCATION/adaptors/lib`. The name of the jar file which contains the gram adaptor is `GlobusResourceBrokerAdaptor.jar`. So if you like to invoke the gram adaptor of JavaGAT you have to enter:

```
-RB.adaptor GlobusResourceBrokerAdaptor
```

- `-RB.jobmanager` Enables to select the usage of a job manager as SGE or PBS. If no job manager is specified here, the default job manager will be invoked; usually this would be `fork`.

The script `gat-job-run` consists mainly of the java call for the `GATJobRun` package. However, you can define some variables for JavaGAT in this call; the most important one are:

- `-Dgat.adaptor.path=<path name>` the name of the directory where the adaptors (there jar files) are located;
- `-Dgat.debug` set the logging level to debug, JavaGAT writes debug and info messages to the console, very helpful in case of problems.

## 3.2 File operations with GATFileOps

Like `GATJobRun` `GATFileOps` also uses a script as entry point. It is called `gat-file-operation`, and it is located in the directory

```
$GAT_LOCATION/astrogrid-packages/GATJobRun/scripts.
```

Again we recommend to add this directory to the `PATH`. Calling `gat-file-operation` without arguments, will deliver list of possible options and arguments:

```
Usage: gat-file-operation [OPTIONS] <operation> <source> <dest>
```

OPTIONS:

```
-Adaptor [STRING] force the use of a specific file adaptor
```

```
<operation>: RemoteCopy, RemoteMove or DeleteFile
```

```
<source>: source file
```

```
<dest>: destination file; not used at <operation> DeleteFile
```

The only option that is available is `-Adaptor`, and it can be used to force the usage of a special adaptor. The other arguments are necessary for a call of `gat-file-operation`, in order to obtain the desired operation...

- `<operation>`: This argument can take the values `RemoteCopy`, `RemoteMove`, or `DeleteFile`. `RemoteCopy` copies a file, if desired from one grid host to another; `RemoteMove` moves a file through the grid, and `DeleteFile` deletes a file anywhere in the grid.
- `<source>` The source file of the operation.
- `<dest>` The destination file of the operation. The destination file will be ignored in case of `DeleteFile`.

Both `<source>` and `<dest>` can be file names or URIs, dependent on where the files are located. In order to benefit of the flexibility of GAT, Java-GAT offers its own protocol type "any". So a `RemoteCopy` call with `gat-file-operation` might look as follows:

```
gat-file-operation RemoteCopy any://localhost.de//home/ali/bin/test-geo600.sh
                                any://remotehost.de//tmp/tester.sh
```

If `any` is entered instead of `file` or `gsiftp`, any adaptor that can be used for the copy will be invoked. Using `gsiftp` will force the usage of `gridftp` for the transfer.

## 4 How to program against the JavaGAT-API

In this chapter we don't provide a description of the whole JavaGAT-API, this is part of a JavaGAT documentation. However, because we have developed job submission adaptors for SGE and PBS within the AstroGrid project, and command line interfaces for file operations and job submission, we present two short examples on how to program against the JavaGAT-API for file operations, and for job submission.

### 4.1 File operations on entire files with JavaGAT

If you like to copy a file from one URI (or file) to another, your class might look as follows:

```
1     import org.gridlab.gat.*;
2     import org.gridlab.gat.io.File;
3
4     public class RemoteCopy {
5         public static void main(String[] args) throws Exception {
6             GATContext context = new GATContext();
7
8             URI src = new URI(args[0]);
9             URI dest = new URI(args[1]);
10            File file = GAT.createFile(context, src); // create file object
11
12            file.copy(dest); // and copy it
13            GAT.end();
14        }
15    }
```

Some comments to the code. To enable any JavaGAT operation you need to create a new instance of the class `GATContext`, as shown in line 6. This instance is necessary for the creation of a `GATFile`, also for a `GATResourceBroker`, which is needed for the job submission.

The class above is called with two arguments, namely the source, and the destination file. These files must be given in an URI description, e.g. as `ftp://10.0.0.1/output`. Important in this context is that JavaGAT offers the protocol type `any://`, and giving an URI with the protocol `any://`, allows JavaGAT to use any protocol which can be used for a file operation. This is the concept of **late binding** realized in JavaGAT. Due to this concept, JavaGAT uses the first adaptor for a file operation, which can be invoked successfully. If you like to force the usage of a specific adaptor, you must enter the protocol you want to use, e.g. `ftp://`, `gsiftp://`, `http://`, `file://`,...

After getting a new instance of the `GATFile` class with `GAT.createFile(context, src)`, it is possible to access on all method offered by this class. The `GATFile` inherits all methods of the Java File class, and offers extensions as `copy` or `move`. For a detailed description please look at the JavaDoc of JavGAT. The method `copy` is then used, to copy the file. Every JavaGAT program should call `GAT.end` before exiting the class. This terminates all threads which might be still alive.

## 4.2 Job submission with the ResourceBroker class

At the begin again a screenshot of an example class, which enables to submit a job.

```
import org.gridlab.gat.*;
import org.gridlab.gat.io.File;
import org.gridlab.gat.resources.*;

public class SubmitLocalJob {
    public static void main(String[] args) throws Exception {
        GATContext context = new GATContext();

        URI LocalFile=new URI(args[0]);
        URI RemoteFile=new URI(args[0]);
        URI PostRemoteFile=new URI(args[0]);
        URI PostLocalFile=new URI(args[0]);

        SoftwareDescription sd = new SoftwareDescription();
        sd.setLocation("file:///bin/hostname");
        File stdout = GAT.createFile(context, "hostname.txt");
        sd.setStdout(stdout);
        sd.addPreStagedFile(GAT.createFilecontext,LocalFile),
            GAT.createFile(context,RemoteFile));
        sd.addPostStagedFile(GAT.createFilecontext,PostRemoteFile),
            GAT.createFile(context,PostLocalFile));

        Hashtable hardwareAttributes = new Hashtable();
        hardwareAttributes.put("machine.node", "exec-host.lrz.de");
        hardwareAttributes.put("memory.size", "64M");
        rd = new HardwareResourceDescription(hardwareAttributes);

        JobDescription jd = new JobDescription(sd,rd);
        ResourceBroker broker = GAT.createResourceBroker(context);
        Job job = broker.submitJob(jd);
        while (job.getState() != Job.STOPPED &&
            job.getState() != Job.SUBMISSION_ERROR) Thread.sleep(1000);
    }
}
```

To submit a job, you need to create a SoftwareDescription, a HardwareResourceDescription, both are needed to create a JobDescription, and the JobDescription is the only argument needed to create an instance of the ResourceBroker class.

Among others, the SoftwareDescription sd

offers the following methods:

- `sd.setLocation("executable", to add the name of the executable;`
- `sd.SetArguments(String[] jobargs), to add the executables arguments;`
- `sd.setStdin(GATFile Input, sd.setStdout(GATFile Output)), allows to define stdin and stdout,`
- `sd.addPreStagedFile(GATFile File, sd.addPostStagedFile(GATFile File, can be used, to define files for pre, and poststaging`

The `SoftwareDescription` contains a list of variables which must be known, to execute the program For a complete description please have a look at the Java documentation of JavaGAT.

The `HardwareResourceDescription` is realized as a free form key-value-pair table. In the example above the value for `machine.node` is `exec-host`, this is meant as the execution host, and for the key `memory.size` we get the value `64M`. The point here is that the adaptors must need to know the values of the keys, so the solution which is realized right now, isn't really generic. However, in one of the next JavaGAT releases it is planned ot use the JSDL description for setting the attribute of soft- and hardware. After the Hashtable has been filled, you can create the `HardwareResourceDescription` by calling

```
rd = new HardwareResourceDescription(hardwareAttributes);
```

The `JobDescription` is necessary for submitting a job, checking its status, killing it, etc... You can create this `JobDescription` with the command:

```
JobDescription jd = new JobDescription(sd,rd);
```

The last step before submitting a job, is to get a `ResourceBroker`, what is done in the example by:

```
ResourceBroker broker = GAT.createResourceBroker(context);
```

Having the the `ResourceBroker broker` available, allows you to submit a job with:

```
Job job = broker.submitJob(jd);
```

The `Job` class `job` allows some operations on the job, e.g. as:

- `job.getState`, to check the status of the job;
- `job.stop`, to stop a job;
- `job.hold`, to hold a job;
- `job.resume`, to resume a job;
- `job.getExitStatus`, to get exit status of a job.

However, the job scheduler must support the operations.