

AHE Client User Guide

TABLE OF CONTENTS

1: Installing the AHE Java command line and GUI clients	2
1.1 Client Installation Prerequisites	2
1.2 Installation	2
1.3 Post-Installation	2
1.4 Setting up the keystore	2
1.5 Configuring the client	2
1.6 Client logging	3
2: Using the GUI Client	4
2.1 Starting the GUI client	4
2.2 The Settings panel	4
2.3 The Manage certificates panel	4
2.4 Prepare a new job panel	5
2.5 View current jobs panel	5
3: Using the Command Line Clients	7
3.1 ahe-destroy	7
3.2 ahe-monitor	7
3.3 ahe-start	7
3.4 ahe-getoutput	8
3.5 ahe-list	8
3.6 ahe-prepare	9
3.7 ahe-terminate	9
3.8 ahe-getproperties	10
3.9 ahe-listapps	10
3.10 ahe-refresh	10
4: Creating a Client Input File Parser	12
4.1 Developing application parser plug-ins	12
4.2 Example configuration file parser	12
4.3 Compiling the plug-in	16

1: Installing the AHE Java command line and GUI clients

1.1 Client Installation Prerequisites

The AHE Java clients require Java Runtime Environment 1.4.2 or above. The latest version of the Java Runtime Environment can be downloaded from <http://java.sun.com/>

1.2 Installation

Download the client tar-ball `aheclient-1.0.0.tar.gz` from <http://www.realitygrid.org>. To install the Java clients, untar the archive as follows:

```
$> tar zxvf aheclient-1.0.0.tgz
```

This will create the directory `aheclient-1.0.0`

1.3 Post-Installation

Add the following line to your `~/.bash_profile`:

```
export AHECLIENT_HOME=path to aheclient
```

1.4 Setting up the keystore

Before running the client you need to set up a Java keystore containing the X.509 digital certificate issued by your certificate authority. The certificate should be stored in P12 format (the format used when exporting it from your browser). See <http://www.grid-support.ac.uk/content/view/67/42/> for more information on exporting your certificate.

From the `$AHECLIENT_HOME/bin` directory type:

```
$> ./kssetup path_to_p12_cert
```

Enter the password for your P12 certificate then enter a password for the keystore you are creating (this will be used to unlock the GUI and command line clients). Repeat the keystore password and type yes, when asked to trust the certificate.

1.5 Configuring the client

The client configuration settings are by default stored in the Java `.properties` file `$AHECLIENT_HOME/conf/aheclient.properties`. Most of the settings are managed by the ahe GUI client. The settings are:

- `uk.ac.ucl.chem.ccs.aheclient.ahedavserver`: url of file stage webdav server
- `uk.ac.ucl.chem.ccs.aheclient.ahedavpasswd`: password of webdav server account
- `uk.ac.ucl.chem.ccs.aheclient.ahedavuser`: username of webdav server account
- `uk.ac.ucl.chem.ccs.aheclient.caroot`: alias of trusted ca in keystore (default = rootca)

- `uk.ac.ucl.chem.ccs.aheclient.jobfactoryepr`: url of job factory server endpoint
- `uk.ac.ucl.chem.ccs.aheclient.jobregepr`: url of job registry server endpoint
- `uk.ac.ucl.chem.ccs.aheclient.keystore`: path to Java keystore
- `uk.ac.ucl.chem.ccs.aheclient.mpcheck`: specifies whether GUI client will check the MyProxy server for a proxy credential at startup (true/false)
- `uk.ac.ucl.chem.ccs.aheclient.mpwarn`: specifies whether GUI client will warn when your MyProxy certificate is about to expire (true/false)
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-dn`: DN of MyProxy server's X.509 certificates
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-lifetime`: lifetime of MyProxy credential retrieved by GridSAM (in seconds)
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-port`: MyProxy server port
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-pw`: MyProxy server pass
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-server`: address of MyProxy server
- `uk.ac.ucl.chem.ccs.aheclient.myproxy-un`: MyProxy server username
- `uk.ac.ucl.chem.ccs.aheclient.cache`: the location of the command line client's local job cache

The `jobfactoryepr`, `jobregepr`, `ahedavserver`, `ahedavpasswd`, `ahedavuser` will be provided by your AHE server administrator. If you are running applications on grid resources that use proxy certificate authentication, your grid resource provider should be able to supply the correct MyProxy server settings. See section 2.2 for details of how to configure these settings using the GUI client.

By default the command line clients will prompt for the keystore password before executing a query. The following property: **`uk.ac.ucl.chem.ccs.aheclient.passwd`** can be used to specify the keystore password to the command line clients (for use when automatically scripting operations). Remember: if using the properties file to store passwords be sure to set appropriate file permissions.

1.6 Client logging

By default, the command line clients will write logging information to `$(AHECLIENT_HOME)/log/clitools.log`. The GUI client will write logging information to the file `$(AHECLIENT_HOME)/log/guiclient.log` and standard out. Logging behaviour can be changed by editing the respective `log4j.properties` files in the `$(AHECLIENT_HOME)/conf` directory.

2: Using the GUI Client

The GUI client provides a self contained environment to create and upload proxy credentials, to build, start and monitor jobs, and to stage input and output files.

2.1 Starting the GUI client

From the \$AHECLIENT_HOME/bin directory, type:

```
$> ./ahe-guiclient
```

Enter the password for the keystore you created above to unlock the client. Clicking **Cancel** will allow you to configure the GUI client without entering a password (for example to change the location of your keystore). To unlock, click **File > Unlock** and enter your password.

2.2 The Settings panel

The setting panel allows you to configure both the GUI and command line clients. Settings are stored in the aheclient.properties file (see section 1.5). In the **General Settings** section, you can set the following:

- _ Job Registry Endpoint
- _ Job Factory Endpoint
- _ Keystore Location
- _ Filestage (webdav) server
- _ Filestage username
- _ Filestage password

In the **MyProxy Settings** panel you can configure the following:

- _ MyProxy Server
- _ MyProxy DN
- _ MyProxy Port
- _ MyProxy Lifetime
- _ MyProxy Username
- _ MyProxy Password
- _ Whether you are warned when your MyProxy certificate is about to expire
- _ Whether the client checks for a MyProxy certificate on startup

After making changes to the settings, click **Save**, or **File > Save Settings**. Note: when generating a new proxy credential on the Manage certificates screen, the MyProxy username and password will be automatically saved.

2.3 The Manage certificates panel

The GUI client allows you to generate a MyProxy certificate based on the X.509 certificate stored in your Java keystore, and upload it to a MyProxy server. To create and upload a MyProxy certificate, in the Upload New Proxy Credential section, enter a password for your proxy certificate twice. The X.509 certificate you loaded into your keystore should be already selected. Set the number of days you would like the certificate to be kept on the myproxy server and the strength of the certificate, then click **Create and Upload**. Information about the new certificate will be displayed in the Current Proxy Certificate panel.

If you have generated and uploaded a MyProxy certificate using a third party tool you can check the information associated with it (for example the expiry time) using the GUI client. First, ensure that the MyProxy settings (username, password and server) are correct in the Settings panel, then click **Update Credential Information**.

To remove a proxy certificate from the server, click Delete Credential From Server. Note, you will not be able to submit jobs to the UK National Grid Service (NGS) or US TeraGrid without a proxy certificate.

2.4 Prepare a new job panel

From here you can search for job factories to manage your job, and launch the job submission wizard. The **Select application type to run** drop-down list lists all of the applications that have configuration parser plug-ins installed (see the Creating parser plug-ins tutorial in section 4 of this guide). Either select the type of job that you want to run, or select **all apps** to view all possible applications, then click **Find Job Factories**. A list of potential factories will be returned.

Select the job factory to use, then click **Launch Wizard**. The wizard displays a series of steps used to build and submit your job to the target machine. The steps are:

- _ Step 1 - Enter a name to identify the job, and specify the number of processors to use. You can also enter constraints on the types of resources that will be returned. If you don't specify a wall time limit, the job will default to 12 hours.
- _ Step 2 - Select the machine to run the job on. The machines that you can submit jobs to relate to the instances of GridSAM configured in the AHE server. For example GridSAM instances can be installed that submit jobs to the NGS and TeraGrid.
- _ Step 3 - Select the job configuration file, or skip if there isn't one. If the application that you want to run has a configuration parser installed (by default parsers for the namd and lammps molecular dynamics codes are installed) then the client will attempt to discover the input and output files, and also rewrite the configuration file, removing relative paths, ready for submission on the target machine.
- _ Step 4 - The input and output files will be displayed. These can be edited/added/removed by clicking **Edit**. Click **Stage** and the files will be staged to the intermediate webdav server, ready for GridSAM to stage to the target machine.
- _ Step 5 - Review the job details. Any extra command line arguments can be added here. Click **Finish** to start the job. Unticking **Start job running now** will allow you to start the job from the **View current jobs** panel.

2.5 View current jobs panel

The View current jobs panel allows you to manage current jobs, and stage output files back to the local machine. Any jobs just created via the wizard will be displayed here on clicking **Finish**. If you have restarted the client since starting a job, or started a job via a different client such as the Java command line tools, click **Update Job List** to query the AHE server registry of jobs and update the list.

Double-click on a job to view its details, or right-click on it and choose **View details**. The job's details will be displayed in a new tab. You can also right-click on a job and choose **Destroy** to remove it from the registry, or **Destroy & Remove Staged Files** to remove the job from the registry and delete any staged files from the webdav server.

On the job details display panel, click **Update Job** to check the status of the job, or set the polling timer to automatically query the job status at a regular interval. You

can also terminate a job (stop it from running) or destroy a job (remove it from the AHE job registry), by clicking the **Terminate Job** and **Destroy Job** buttons respectively. Ticking **Delete staged files when destroying job** will remove any files associated with the job from the webdav server.

The Gridsam Status tab will report the job status from GridSAM, and a description of the job's status. If the job fails, the failure cause supplied by GridSAM will be reported here.

The files to be staged back can be viewed by clicking on the **Staged Files** tab. Once the job status is GRIDSAM DONE, the output files can be staged back to your local machine by clicking **Download**. By default they will be staged to the location specified in the job configuration file. This can be changed by clicking the **Local Dir** button.

3: Using the Command Line Clients

The command line clients provide much the same functionality as the GUI client, but have the added advantage that they can be included in scripts to create complex job workflows. The command line clients can be found in the `$AHECLIENT_HOME/bin` directory.

3.1 `ahe-destroy`

SYNOPSIS

```
usage: ahe-destroy [-s simname] [-i index] [-r] [-e endpoint]
[-help]
```

DESCRIPTION

Destroys the simulation from the AHE job registry.

One of the following three options is required:

- **-e:** the URL of the job endpoint
- **-s:** job name
- **-i:** index job in local job cache

Optional arguments:

- **-r:** remove staged files from webdav server
- **-help:** display command help

RESULTS

Upon successful submission the command reports that the job has been destroyed.

3.2 `ahe-monitor`

SYNOPSIS

```
usage: ahe-monitor [-s simname] [-i index] [-e endpoint] [-
help]
```

DESCRIPTION

Reports the status of job.

One of the following three options is required:

- **-e:** the URL of the job endpoint
- **-s:** job name
- **-i:** index job in local job cache

Optional arguments:

- **-help:** display command help

RESULTS

Upon successful submission the command reports the GridSAM status of the job.

3.3 `ahe-start`

SYNOPSIS

```
usage: ahe-start [-s simname] [-i index] [-n cpucount] [-
wallTimeLimit time] [-config file] [-RM rmname] [-e endpoint]
[-help]
```

DESCRIPTION

Processes the job configuration file to discover job input and output files, stages those files to the intermediate webdav server, and submits the job to the specified target machine.

One of the following three options is required:

- **-e:** the URL of the job endpoint
- **-s:** job name
- **-i:** index job in local job cache

Required arguments:

- **-config:** configuration file for the job
- **-RM:** the common name of the machine to run the job on
- **-wallTimeLimit:** the wall time limit for the job
- **-n:** the number of processors to run the job on

Optional arguments:

- **-help:** display command help

RESULTS

Upon successful submission the command reports the GridSAM status of the job.

3.4 ahe-getoutput

SYNOPSIS

```
usage: ahe-getoutput [-s simname] [-i index] [-d] [-e  
endpoint] [-l localdir] [-help]
```

DESCRIPTION

Retrieves a job's output files to the local machine. By default files will be staged back to the location specified in the job's configuration file. Note that output can only be retrieved when the job's status is GRIDSAM DONE (see ahe-monitor).

One of the following three options is required:

- **-e:** the URL of the job endpoint
- **-s:** job name
- **-i:** index job in local job cache

Optional arguments:

- **-l:** specifies the local directory path to stage the files to
- **-d:** specifies whether to create a directory for the staged files based on the job's resource ID
- **-help:** display command help

RESULTS

Upon successful submission the command will retrieve the job's output files to the specified location.

3.5 ahe-list

SYNOPSIS

```
usage: ahe-list [-e endpoint] [-help]
```

DESCRIPTION

Lists the jobs owned by the submitting user in the AHE job registry, and caches the result locally.

Optional arguments:

- **-e:** the URL of the job registry endpoint. If not specified the system default is used

- **-help:** display command help

RESULTS

Upon successful submission the command displays the jobs contained in the registry and updates the local job cache.

3.6 ahe-prepare

SYNOPSIS

```
usage: ahe-prepare [-RMVirtualMemory virtualmemory] [-RMMemory
memory] [-e endpoint] [-RMIP ipaddress] [-help] [-s
simulationName] [-wallTimeLimit time] [-RMArch arch] [-RMDisk
disk] [-app application] [-RMType NGSorTeragrid] [-
RMCommonName rmname] [-RMOpSys opsys][-RMCPUCount cpucount]
```

DESCRIPTION

Creates a stateful resource on the AHE server to manage the job, and returns a list of potential machines to run th job. Optional arguments can be used to constrain the list of machines returned.

Required arguments:

- **-e:** the URL of the job factory endpoint, discovered from the ahe-listapps command
- **-s:** identifying name for the job
- **-app:** application to run

Optional arguments:

- **-help:** display command help
- **-RMVirtualMemory:** virtual memory of the potential target machine
- **-RMMemory:** memory of the potential target machine
- **-RMIP:** IP address of the potential target machine
- **-wallTimeLimit:** minimum wall time limit available on the potential target machine
- **-RMArch:** architecture of the potential target machine
- **-RMCommonName:** common name of the potential target machine
- **-RMOpSys:** operating system of the potential target machine
- **-RMCPUCount:** minimum number of CPUs on the potential target machine

RESULTS

Upon successful submission the command displays a list of the potential target machines that the job could be run on.

3.7 ahe-terminate

SYNOPSIS

```
usage: ahe-terminate [-s simname] [-i index] [-e endpoint] [-
help]
```

DESCRIPTION

Terminates a running job.

One of the following three options is required:

- **-e:** the URL of the job endpoint
- **-s:** job name
- **-i:** index job in local job cache

Optional arguments:

- **-help:** display command help

RESULTS

Upon successful submission the command reports the job has been terminated.

3.8 **ahe-getproperties**

SYNOPSIS

```
usage: ahe-getproperties [-s simname] [-i index] [-e endpoint]
[-help]
```

DESCRIPTION

Lists the properties of the specified job.

One of the following three options is required:

- **-e**: the URL of the job endpoint
- **-s**: job name
- **-i**: index job in local job cache

Optional arguments:

- **-help**: display command help

RESULTS

Upon successful submission the command displays the properties associated with the specified job, including the job's input and output files and status.

3.9 **ahe-listapps**

SYNOPSIS

```
usage: ahe-listapps [-e endpoint] [-a application] [-help]
```

DESCRIPTION

Lists the applications available in the AHE application registry.

Optional arguments:

- **-e**: the URL of the application registry endpoint
- **-a**: the application to search for. If omitted, all applications in the registry will be returned
- **-help**: display command help

RESULTS

Upon successful submission the command displays the applications available from this AHE server installation, with job factory endpoints.

3.10 **ahe-refresh**

SYNOPSIS

```
usage: ahe-refresh [-e endpoint] [-help]
```

DESCRIPTION

Updates the local job cache from the AHE job registry.

Optional arguments:

- **-e**: the URL of the job registry endpoint
- **-help**: display command help

RESULTS

Upon successful submission the command updates the local job cache.

4: Creating a Client Input File Parser

4.1 Developing application parser plug-ins

The AHE Java command line and GUI clients attempt to discover a job's input and output files by parsing its configuration file, which it then uses to decide which files need to be staged to and from the remote resource. By default the clients come with parsers for namd and lammmps configuration files, but features a plug-in architecture to allow configuration parsers to be written for any kind of input file.

A configuration parser is created by writing a Java class that is contained in the `uk.ac.ucl.chem.ccs.aheclient.confparser` and implements the `uk.ac.ucl.chem.ccs.aheclient.confparser.AHEConfParser` interface. See the client JavaDoc for further information on the AHEConfParser interface. The implemented class must follow a specific naming convention: APPConfParser where APP is the name of the application the parser relates to (note the name of the application must be in uppercase).

The class must implement methods to return two `java.util.Vector` objects containing `uk.ac.ucl.chem.ccs.aheclient.util.JobFileElement` objects. JobFileElements added to the vectors can be tagged by setting the description field to, to represent certain files required by the AHE to submit a job. The possible tags are:

- _ `stdin` – the stdin directed to the application
- _ `conf-file` – the jobs original configuration file
- _ `stderr` – the stderr directed from the application
- _ `stdout` – the stdout directed from the application
- _ `argument` – the argument passed to the application

The `stderr` and `stdout` files are required, with the rest optional. Usually an application would take its input from either `stdin` or `argument`.

4.2 Example configuration file parser

The following is a parser for the sample sorting application that comes with the AHE server. The application takes a simple configuration file as its input, shown below:

```
inputfile /path/to/input.txt
outputfile /path/to/output.txt
```

The input file contains a list of words, with a single word on each line. The application sorts the input file and rewrites it to the output file. The configuration parser plug-in must read the above file, add the input file to the Vector of input files and the output file to the output files vector. In addition, the parser must rewrite the configuration file, stripping out file paths so that the application can be run on the remote machine. The rewritten file is also added to the input files vector, and `stdout.txt` and `stderr.txt` are added to the output files vector. The following code shows how the sample parser is implemented:

```
package uk.ac.ucl.chem.ccs.aheclient.confparser;

import java.util.Vector;
import java.io.FileReader;
import java.io.StreamTokenizer;
import java.io.FileOutputStream;
import java.io.PrintStream;
```

```

import java.io.File;

import uk.ac.ucl.chem.ccs.aheclient.util.JobFileElement;
import uk.ac.ucl.chem.ccs.aheclient.util.Tools;

public class SORTConfParser implements AHEConfParser {
    //vectors of input and output JobFileElements
    private Vector inFiles, outFiles;

    //local path and remote path
    private String path, stagePath;
    private String error = ""; //error to display

    //initialise the config file parser with the url
    //to stage files to
    public void init(String stagePath) {
        inFiles = new Vector();
        outFiles = new Vector();
        this.stagePath = Tools.checkURL(stagePath);
        path = null;
    }

    //parse the file filename
    public boolean parse(String filename) {
        FileReader fr = null;
        File file; //input file to read
        //count the line we are on, for error reporting
        int lineno = 1;

        //string containing the rewritten output
        String confOutFile = "";

        //check and open the file
        if (!filename.equals("") && filename != null) {
            try {
                file = new File(filename);
                path = file.getParent() + File.separator;
                fr = new FileReader(file);
                String name = Tools.getTarget(filename);
                //add the config file to our list of input
                //files to stage
                inFiles.add(new JobFileElement("conf-file", name,
                    filename, stagePath + name));
            } catch (Exception e) {
                error = "File " + filename + " cannot be read";
                return false;
            }
        }

        String previousToken = ""; //the last token we found

        //tokenise and process the input file
        try {
            StreamTokenizer stk = new StreamTokenizer(fr);
            //set up the file syntax
            stkordinaryChars('\u0000', '\u007F');
            stk.whitespaceChars('\u0000', '\u0020');

```

```

stk.wordChars('\u0021', '\u007F');
stk.commentChar('#');
stk.eolIsSignificant(true);

//tokenise file
while (stk.nextToken() != StreamTokenizer.TT_EOF) {
    //look for input file token;
    if (previousToken.equals("inputfile")) {

        //check if path is relative
        if (!stk.sval.startsWith("/")) {
            stk.sval = path + stk.sval;
        }
        //remove path from file name
        String name = Tools.getTarget(stk.sval);
        //add file to rewritten input file, removing path
        confOutFile = confOutFile + "\t" + name;
        //add file to input files vector
        inFiles.add(new JobFileElement("inputfile",
            name, stk.sval, stagePath + name));
    } else if (previousToken.equals("outputfile")) {
        //check if path is relative
        if (!stk.sval.startsWith("/")) {
            stk.sval = path + stk.sval;
        }
        String name = Tools.getTarget(stk.sval);
        confOutFile = confOutFile + "\t" + name;
        outFiles.add(new JobFileElement("outputfile",
            name, stk.sval, stagePath + name));
    } else if (stk.ttype == StreamTokenizer.TT_EOL) {
        //add newline to rewritten file at the right place
        confOutFile = confOutFile + "\n";
        lineno++; //increment line number
    } else if (confOutFile.endsWith("\n") ||
        confOutFile.equals("")) {
        //if this is the first token add to the begining
        //of the line
        confOutFile = confOutFile + stk.sval;
    } else {
        //else add token to rewritten file
        confOutFile = confOutFile + "\t" + stk.sval;
    }
}

if (stk.sval != null) {
    //set previous token
    previousToken = stk.sval.toLowerCase();
}
} //end while

//add stdout and stderr to output files vector
outFiles.add(new JobFileElement("stdout", "stdout.txt",
    path + "stdout.txt", stagePath + "stdout.txt"));
outFiles.add(new JobFileElement("stderr", "stderr.txt",
    path + "stderr.txt", stagePath + "stderr.txt"));

} catch (Exception e) {
    //check for error paring file

```

```

        System.err.println("Error parsing file " + filename +
            " at line number " + lineno);
        e.printStackTrace();
        error = "Error parsing file " + filename +
            " at line number " + lineno;
        return false;
    }

    //write parsed file out without relative file paths
    FileOutputStream out; // declare a file output object
    PrintStream p; // declare a print stream object

    try
    {
        // Create a new file output stream
        // connected to rewritten output file (.ahe)
        filename = filename + ".ahe";
        out = new FileOutputStream(filename);
        // Connect print stream to the output stream
        p = new PrintStream(out);
        p.println (confOutFile);
        p.close();
    }
    catch (Exception e)
    {
        //check for error rewriting file
        System.err.println ("Error writing to file");
        e.printStackTrace();
        error = "Error writing output to " + filename;
        return false;
    }
    //add rewritten config file to input files vector
    String name = Tools.getTarget (filename);
    inFiles.add(new JobFileElement("argument", name,
        filename, stagePath + name));

    return true;
}

//return input files vector
public Vector getInFiles() {
    return inFiles;
}

//return output files vector
public Vector getOutFiles() {
    return outFiles;
}

//return error to display in gui
public String getError() {
    return error;
}
}

```

The source code above can be found in the file
\$AHECLIENT_HOME/src/uk/ac/ucl/chem/ccs/aheclient/confparser/SORTConfParser.
java

4.3 Compiling the plug-in

Configuration file parser plug-in class files are stored in the directory:
\$AHECLIENT_HOME/lib/uk/ac/ucl/chem/ccs/aheclient/confparser

To install the sort configuration parser described above, save the file
SORTConfParser.java in the plug-in directory. Next add the AHE_GUI.jar to your
classpath:

```
$> export CLASSPATH=$CLASSPATH:$AHECLIENT_HOME/lib/AHE_GUI.jar
```

then CD to the plug-in folder and compile the plug-in as follows:

```
$> javac SORTConfParser.java
```

Once compiled, restart the AHE GUI client if open. If the plug-in is installed correctly,
you should see an option for your new application in the **Select application type to
run** in the GUI client **Prepare a new job panel**.