

## Software Basics

[Basic account and software setup](#)

[Initial login](#)

[On lxplus](#)

[At Tier3s](#)

[Setup the UserInterface](#)

[On lxplus](#)

[At Tier3s](#)

[Grid Certificates](#)

[Registering with AMI](#)

[Tutorial Requirements](#)

[Define ALRB TutorialData](#)

[Getting started with Athena OPTIONAL](#)

[Running a simple Athena job](#)

[OPTIONAL/ADVANCED Simple Athena job using acm](#)

[Adding a configurable property to your algorithm](#)

[Loop over a ROOT TTree](#)

[Where to learn more](#)

## Basic account and software setup

### Initial login

You can run the tutorial hands-on sections using either option:

**(A) lxplus**, or

**(B) Tier3 machine** (a site, your own laptop, or the CernVM guest machine). You may first want to ensure that you are able to do password-less logins or preferably Kerberos authentications to these machines. Instructions to do this can be found [here](#).


Below you will find instructions for either lxplus or the Tier3 option, choose one.

### On lxplus

These machines are ATLAS ready and you do not have to install additional software.

To log into lxplus you should type:

```
ssh -Y -l <username> lxplus.cern.ch
```

 Go to <https://resources.web.cern.ch/resources/> and select **List Services** (top). In the search box type "afs". You should see a link called "AFS Workspaces" appear. Click on the link.

Select "Settings" in the left menu, and you should find that you are able to increase your quota for both you home area and workspace area. This is granted by increments so if you click it several times, you can get the maximum allowed quota.

You can use either your home or workspace area for the tutorial.

To check X11 forwarding is done correctly (the -Y part after ssh) type something like `xterm` (if this exists).

### At Tier3s

Check that your machine is ATLAS-ready. if you are managing your machine, the minimal requirements are

- cvmfs installed, mounted and working; see [Instructions](#).

but otherwise this will be checked in the next sections.

Login to your Tier3 (if you are not already logged on) with the -Y option so that your X-windows are exported back.

```
ssh -Y -l <username> <your tier3 hostname>
```

To check X11 forwarding is done correctly (the -Y part after ssh) type something like `xclock` or `xlogo` (if these exists).

## Setup the UserInterface

The user interface we will use will be ATLASLocalRootBase which is described [here](#). Note that this is a menu driven UI and it also has `helpMe`, `showVersions` and the `--help` or `-h` option works for all commands. In the examples below, you will see the command `setupATLAS`; you will have to type this only once per session. To see the menu again, type `printMenu`.

### On lxplus

First determine the type of shell you are using (bash, zsh or tcsh) by typing `ps -p $$`. If

- bash : edit `~/.bashrc` and add the lines

```
export RUCIO_ACCOUNT=<your lxplus username>
```

- zsh : edit ~/.zshrc and add the lines

```
export RUCIO_ACCOUNT=<your lxplus username>
```

- tcsh: edit ~/.cshrc and add the lines

```
setenv RUCIO_ACCOUNT <your lxplus username>
```

Simply type `setupATLAS` afterwards.

You do not need to define anything else.

## At Tier3s

If your site admin has not defined it, or if it is unavailable for some reason, you can define these in your login scripts.

First determine the type of shell you are using (bash, zsh or tcsh) by typing `ps -p $$`. If

- bash : edit ~/.bashrc and add the lines

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
```

```
alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh'
```

```
export RUCIO_ACCOUNT=<your lxplus username>
```

- zsh : edit ~/.zshrc and add the lines

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
```

```
alias setupATLAS='source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh'
```

```
export RUCIO_ACCOUNT=<your lxplus username>
```

- tcsh: edit ~/.cshrc and add the lines


```
setenv ATLAS_LOCAL_ROOT_BASE /cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
```

```
alias setupATLAS 'source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.csh'
```

```
setenv RUCIO_ACCOUNT <your lxplus username>
```

Next logout and login again and type `setupATLAS`. For more details, see these [Instructions](#).

## Grid Certificates

 It is important that your grid certificate is working prior to your attending the tutorial; otherwise you may be unable to fully participate in the hands-on exercises which require a valid grid proxy. If you do not have a grid certificate, you may be able to easily obtain one from <https://ca.cern.ch/ca/>; you will then need to apply for VO membership. The organisers may be able to expedite this for you. Please note that you will need to obtain and use a new certificate from the CERN Certificate Authority for each new machine you wish to certify.

In order to access data, submit your analysis jobs to the Grid, and access some restricted web pages, you will need to

- have a Grid certificate,
- be registered with the Atlas VO,
  - at a minimum, you must apply for the following roles: `/atlas` and `/atlas/<your country code>` (eg. your country code can be `ca` for Canada.)
  - your nickname must be the same as your lxplus username
  - Click [here](#) to view your VO memberships
- have the certificate installed
  - in both a web browser (recommend firefox) and
  - in the `~/.globus` directory of every machine you use. The `~/.globus/userkey.pem` and `~/.globus/usercert.pem` files must exist at the end of these instructions.

If your certificate is in your browser, you need to export (backup) the certificate: (Depending on OS and browser this may be ) for Firefox: Preferences (or Tools) -> Advanced -> Encryption -> View Certificates -> **Your Certificates** -> Backup

Use `scp` (or similar) to copy this file across to your lxplus account (e.g. `scp cert.p12 username@lxplus.cern.ch:.` ).

You need to convert your certificate (here assumed to be called `mycert.pfx` into the correct form using:

```
> openssl pkcs12 -in mycert.pfx -clcerts -nokeys -out usercert.pem
```

```
> openssl pkcs12 -in mycert.pfx -nocerts -out userkey.pem
```

```
> chmod 400 userkey.pem
```

```
> chmod 444 usercert.pem
```

**(Word of advice:** When executing `openssl pkcs12 -in mycert.pfx -nocerts -out userkey.pem` you must enter a PEM pass phrase or it could lead to problems.)

Move these two files (`userkey.pem` and `usercert.pem`) to the `.globus` directory (If you haven't got one then `mkdir ~/.globus`). You probably need to remember two passwords, one for the original certificate and one for the converted one. If all is well try:

```
lsetup rucio
```

```
voms-proxy-init --voms atlas
```

Then do:

```
voms-proxy-info -all
```

and you should see something as follows:

...  
attribute : nickname = dvanders (atlas)  
...  
  
(This assumes that you have set up using CVMFS; the old-fashioned way would be to call `source /afs/cern.ch/project/gd/LCG-share/current/etc/profile.d/grid_env.sh` and then create your proxy.)

- Alternative instructions to do this - see these pages for your cloud:
- [Canada](#)
  - [US](#)
  - [CERN](#)

After your registration with LCG for the Atlas VO has been approved and also your voms roles approved, you can then check that everything is working by doing

```
setupATLAS
diagnostics
gridCert
```

and follow the instructions regarding protections.

All tests must pass as described at the end of the `gridCert` command.

Registering with AMI

This will confirm that your browser certificate is also installed and working correctly.

In order to be able to search for data and MC samples in AMI (as well as using its more advanced features), you need to register yourself. AMI itself will be covered later in the tutorial, but we will make sure that you are set-up now. Go to the ami page <https://ami.in2p3.fr>. Sign-in (top right corner) if you are not already signed in. Then in the pull down menu under your name (top right corner) click on 'Account Status'. If you have a fancy picture that says "Your account is valid" then nothing further is required. If not, then you will need to create an account and /or upload a new certificate into AMI. Please follow the instructions provided on the screen, and let us know if you have any questions or problems.

For anybody experiencing problems in creating an account, or in logging in, please follow [these](#) steps.

**Also please note that if you have a firewall on your laptop, it may prevent your certificate from being presented to ami. A symptom of this will be that once your certificate is installed on your laptop or on the browser, when you navigate to ami your browser does not ask for the certificate.**

Tutorial Requirements

The next thing you need to do prior to attending the tutorial is to check that your machine is ATLAS-ready and you are setup properly. Do this on lxplus as well as your Tier3 machine so that you can always fall-back to lxplus in case of an issue.

```
setupATLAS
diagnostics
setMeUp <tutorial version>
```

where <tutorial version> is given by the organizers and unique for each tutorial (e.g.. `triumf-sep2014` for the September 2014 tutorial at TRIUMF).

Tutorial date	tutorial version
<a href="#">CERN January 2018</a> (recommended)	cern-jan2018
<a href="#">CERN September 2017</a>	cern-sept2017
<a href="#">CERN July 2017</a>	cern-july2017
<a href="#">CERN October 2016</a> (recommended if using 2.4.X or earlier)	cern-oct2016
<a href="#">UChicago/ANL March 2016</a>	chicago-mar2016
<a href="#">CERN February 2016</a>	cern-feb2016
<a href="#">CERN May 2015</a>	cern-may2015

**PLEASE NOTE:** Due to the lxplus change to use **CentOS7**, `setMeUp` actually fails for the **CheckOS** part of the checks. This is a known issue and this is fine, please ignore if this is your only error. For the remainder of the tutorial, please make sure all other checks are successful.

**Optional**

You can also download the data files for the tutorial to your local area (usually on large sites). To do this:

```
setupATLAS
diagnostics
setMeUpData <tutorial version> <local dir location for data files>
```

and you will then have to define the `$ALRB_TutorialData` environment variable in your login file as described at the end of the command. You can run `setMeUpData` as many times as you like - each time it will check and only download updates and fix corrupted files.

What this test does is [documented](#).

Define ALRB\_TutorialData

We will define the environment variable \$ALRB\_TutorialData which will point to the location of the input data used for the tutorial (this depends on which tutorial and where you are running). This variable may be provided to you by the tutorial organizers, but if you are working from lxplus you can do one of the following (depending on which shell you are using):

Tutorial	zsh, bash shells	csH shell
<a href="#">CERN January 2018</a> (recommended)	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-jan2018/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-jan2018/
<a href="#">CERN September 2017</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-sept2017/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-sept2017/
<a href="#">CERN July 2017</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-july2017/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-july2017/
<a href="#">CERN October 2016</a> (recommended if using 2.4.X or earlier)	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-oct2016/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-oct2016/
<a href="#">CERN June 2016</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-june2016/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-june2016/
<a href="#">UChicago/ANL March 2016</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/chicago-mar2016/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/chicago-mar2016/
<a href="#">CERN February 2016</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-feb2016/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-feb2016/
<a href="#">CERN May 2015</a>	export ALRB_TutorialData=/afs/cern.ch/atlas/project/PAT/tutorial/cern-may2015/	setenv ALRB_TutorialData /afs/cern.ch/atlas/project/PAT/tutorial/cern-may2015/

Now you can use this environment variable instead of the full path to the files (this allows the tutorial instructions to be site independent). Note, the environment variable \$ALRB\_TutorialData can also be set in ~/.bashrc for the purpose of this tutorial in the same way as \$RUCIO\_ACCOUNT above.

Getting started with Athena OPTIONAL

**PLEASE NOTE:** The following part of the tutorial is only for people who have everything setup properly and are waiting for others to catch up in the hands-on session. [AthAnalysis](#) and [AnalysisBase](#) will be covered in a lot more detail later in the ATLAS Software Tutorial.

Here we will show you a simple example of running Athena. This will serve to make sure all of your environment settings have been implemented correctly.

See [AtlasSetup](#), if you wish to see the full set of options, but please keep to this simple setup until after the tutorials.

This only needs to be done once.

Now that you have your account and certificate set up we will begin.

First, log into lxplus.

Check that you have free space (type `fs lq`) in your home area. If you are almost full, you should try to clear up some space

We will now create your testing area, which is a directory (with name of your choosing) from which you will run this part of the tutorial. You can create as many testing areas as needed for your own work, but only one can be active in each terminal at one time. Think of it like a project area for any given project you are working on.

```
mkdir -p ${HOME}/MyFirstProject
cd ${HOME}/MyFirstProject
```

For this example, we will use version 21.2.93 of the [AthAnalysis](#) project. To set up this version of the atlas software run the following commands from MyFirstProject:

```
mkdir source build run
cd build
asetup 21.2.93,AthAnalysis
```

To check that we have set up the testarea correctly type `echo $TestArea`, and it should return the name of the current directory.

If you need to log back into lxplus and set up the same test area, then you will do the following:

```
ssh -Y <username>@lxplus.cern.ch
cd MyFirstProject/build
setupATLAS
asetup --restore
```

## Running a simple Athena job

We will now run our first athena job. Athena is the event loop framework that comes with all ATLAS software releases (with the exception of the so-called [AnalysisBase](#) releases, which you will encounter later this week). To use athena we create a configuration file, called the **job options**, which specifies what we want to happen. Job options are written in python. The minimal job options are very simple and require little understanding of python (job options are really just a glorified list of property values in the end anyway).

1. We will start in our `run/` directory. This is the dedicated folder where we run the athena jobs from (it can have any name), so that any output produced by the jobs doesn't interfere with the rest of the package structure. Then inside this `run/` directory create a file called `myFirstJobOptions.py` with just the following content (If you are new to linux then `emacs myFirstJobOptions.py` will open the file in a standard editor):

```
theApp.EvtMax=10
```

2. Then run athena with your joboptions:

```
athena myFirstJobOptions.py
```

Take a look at the output, you should see that athena does the following:

- Reads the joboptions to set up the job, including reading YOUR joboptions, i.e. look for a line like:

```
Py:Athena      INFO including file "myFirstJobOptions.py"
```

This is called the **configuration** stage of the job

- Initializes the athena application and any components that were specified in the job options. Look for messages about things being initialized. This is called the **initialization** stage
- Does 'something' 10 times. Look for messages suggesting something is happening over and over 10 times. This is called the **execution** stage.
- Cleans up at the end of the loop. Look for message about things being finalized. This is called the **finalization** stage.

So what you see is that athena is just a fancy for-loop 😊

You can also control the number of 'events' you want to process from the command line, which will override whatever is specified in the joboptions. For example, try:

```
athena --evtMax=20 myFirstJobOptions.py
```

You will learn a lot more about Athena and setting up a package skeleton later in the tutorial but the following optional section is a short introduction using `acm`. However, the use of `acm` is no longer recommended and will not be used for the remainder of the tutorial.

## OPTIONAL/ADVANCED Simple Athena job using acm

⚠ Please note that `acm` is no longer recommended or supported but this short exercise is available during the tutorial if you have time and get set up quickly.

Now we want to make it print "hello world" 10 times during that execution stage. To do this, we will create our own package, and in that package we'll create an algorithm, which is a component that can hold some code to be executed once per event. It's the basic building block of an athena job.

We can create a skeleton package ready for analysis work like this:

```
acm new_skeleton MyPackage
```

This should have created a set of directories and files under your `source/` directory. You would put your analysis code in these files and directories. If you have any questions about the structure of the package, ask a tutor. Hopefully you'll learn more later in the week too.

Note there are two files in the `src/` directory of the package (under `source/MyPackage`), as well as adding some extra information to some other files in the package (you'll learn about those other files throughout the week). Open up the `MyPackageAlg.cxx` file that is now in the `src/` directory, and take a look at the structure of an algorithm. It has an `initialize` method, an `execute` method, and a `finalize` method. `initialize` gets executed once at the start of the event loop, `finalize` gets executed once at the end of the event loop, and `execute` gets executed once per event. Add some simple c++ code in to one of the methods, e.g. put in the `execute` method:

```
std::cout << "Hello world" << std::endl;
```

Save the file. Now we have to compile our package. To do this, type :

```
acm compile
```

Assuming everything compiles ok (if not, ask a tutor), you now need to add your algorithm to the joboption file, so that it will get executed once per event. Return to your `run/` directory and open up `myFirstJobOptions.py`, to add this line:

```
athAlgSeq += CfgMgr.MyPackageAlg("MyFirstAlg")
```

This is how you add algorithms to the main sequence of an analysis athena job. Any algorithm in the main alg sequence will be executed during the job. Save your joboptions and execute them again:

```
athena myFirstJobOptions.py
```

Compare the output of this run to the output from your first run. Confirm that your algorithm has executed and you see your 'Hello world' message! Congratulations, you just ran your first athena job with your own code. Carry on to the next section if you want to learn how to do more things, or feel free to ask the tutors any questions! We will also often use Athena again later in the tutorial week.

## Adding a configurable property to your algorithm

A key concept of the athena framework is that you use the python joboptions to configure the components of your job (components are algorithms, tools, services) - i.e. you separate the actual C++ code that does the work from the simple python that specifies configuration options. Here we will add a configurable property to the algorithm you just created, and see how to set it through the joboptions.

First go back to your algorithm source code ([MyPackageAlg.cxx](#)) and look in the algorithms constructor for the following comment line:

```
// declareProperty( "Property", m_nProperty = 0, "My Example Integer Property" ); //example property declaration
```

This is how you declare a property in your algorithm. Uncomment that line. You'll now also have to add m\_nProperty as a member variable to your [MyPackageAlg](#) class. Open up the [MyPackageAlg.h](#) header file and add it in, after the private: line (so it's a private data member):

```
private:
    double m_nProperty;
```

Here i've made it a number property, you could make it a string, or a vector of strings, or a vector of doubles. You can have lots of different types of properties! Now open up our [MyPackageAlg.cxx](#) source file again, and perhaps add some code to print out the value of your property (put it in initialize method, for example). If it's a number like I have done here, you can do:

```
std::cout << "Property value = " << m_nProperty << std::endl;
```

It's also good practice for the case of number properties to specify the 'default' value of the property, which is conventionally done in the declareProperty line like this:

```
declareProperty( "Property", m_nProperty = 4, "My Example Integer Property" ); //example property declaration ... defaults to 4
```

Recompile your package (acm compile) and then run your job options again. Verify that the property value is printed out at whatever stage you decided to make the algorithm print it out.

Now we want to set the property value in our joboption. Open up myFirstJobOptions.py and add the following line:

```
athAlgSeq.MyFirstAlg.Property = 6
```

Rerun your job and verify that the new value has been transmitted to your algorithm.

## Loop over a ROOT TTree

Now lets see how we would actually run over a file containing events. Lets suppose you wanted to process every entry in a TTree inside a ROOT file (i.e. a flat ntuple). Later in the week you will learn how to read the content of an xAOD, which is the official run-2 data format. But for now we will start simple with a TTree (nb: reading xAOD amounts to changing one line in your job options).

We will try to read the following file:

```
/afs/cern.ch/user/a/asgbase/patspace/data/ntuples/data11_7TeV.00178044.physics_Egamma.merge.NTUP_FASTMON.f354_m765_m777._0001.1.root
```

It contains a tree inside the 'Exotics/Dielec' subfolder called 'Dielec'. And there's a float branch in there called 'elecPt1' which we will try to print out during our event loop. Open your joboptions and add the following two lines:

```
jps.AthenaCommonFlags.AccessMode = "TreeAccess" #activates reading of ROOT TTrees in athena analysis
jps.AthenaCommonFlags.TreeName = "Exotics/Dielec/Dielec" #name of the tree to read
```

The first line are the magic words needed to configure athena to read an ntuple as it's input. The second is to set the [TupleName](#) property of the [EventSelector](#) service (a service is another athena component, like algorithms) to say which tree to read (if you open the file directly in ROOT, you'll see that there is a tree called Dielec inside a couple of ROOT TDirectories).

When athena reads in an input file, it will populate a service called the [StoreGateSvc](#) with information from the input file. We can ask the [StoreGateSvc](#) to print out (each event) what information it contains by adding the following line to the joboptions:

```
StoreGateSvc.Dump=True
```

Save your joboptions and execute the job to run over just 1 event of the input file, like this:

```
athena --evtMax=1 --filesInput=/afs/cern.ch/user/a/asgbase/patspace/data/ntuples/data11_7TeV.00178044.physics_Egamma.merge.NTUP_FASTMON.f354_m765_m777._0001.1.root myFirstJobOpti
```

You'll see a printout of the content of the [StoreGateSvc](#):

