The Athena Startup Kit

W. T. L. P. Lavrijsen
LBNL, Berkeley, CA 94720, USA

The Athena Startup Kit (ASK), is an interactive front-end to the Atlas software framework (ATHENA). Written in PYTHON, a very effective "glue" language, it is build on top of the, in principle unrelated, code repository, build, configuration, debug, binding, and analysis tools. ASK automates many error-prone tasks that are otherwise left to the end-user, thereby pre-empting a whole category of potential problems. Through the existing tools, which ASK will setup for the user if and as needed, it locates available resources, maintains job coherency, manages the run-time environment, allows for interactivity and debugging, and provides standalone execution scripts. An end-user who wants to run her own analysis algorithms within the standard environment can let ASK generate the appropriate skeleton package, the needed dependencies and run-time, as well as a default job options script. For new and casual users, ASK comes with a graphical user interface; for advanced users, ASK has a scriptable command line interface. Both are built on top of the same set of libraries. ASK does not need to be, and isn't, experiment neutral. Thus it has built-in workarounds for known gotcha's, that would otherwise be a major time-sink for each and every new user. ASK minimizes the overhead for those physicists in Atlas who just want to write and run their analysis code.

1. INTRODUCTION

Atlas [1] is a multi-purpose high energy physics experiment planned for the Large Hadron Collider [2] (LHC), which is scheduled for startup after 2007. The data processing needs of Atlas, its variety in subsystems, and its expected longevity pose a big challenge for the software that will be needed for extracting physics out of the future Atlas data. This challenge has attracted a large software community, with developers working today, still a few year before the experiment starts running, on every aspect of the system. Most of the huge Atlas code base, from the software that the end-user runs to the configuration of the infrastructure tools, is currently under heavy development. This has resulted in a state of flux, as a consequence of which all software usage becomes low-level, and all users end up fixing the same problems multiple times. The threshold for new users is high, and casual users are, time and again, forced to relearn most of what they thought they knew.

A large fraction of the Atlas software runs within the ATHENA framework, and in order to alleviate several of the chores that developers and end-users alike have to perform in order to work with that system, the Athena Startup Kit (ASK) has been developed. Its main features are:

- *Automation* of end-user tasks, to save time.

- *Integration* of the framework, the releases, and the release tools to improve perceived coherency.

- *Encapsulation* of volatile wisdom (from web-pages and peoples' heads) of known problems and their workarounds, to make it more accessible.

There are several different categories of ATHENA users and ASK addresses the needs of all of them.

ASK performs the steps needed to setup tools, locate resources, work around known problems, and execute the ATHENA program. In addition, ASK has ATHENA specific functionality: it can generate and update user code meant for ATHENA execution ("algorithms"), generate default job options scripts, and update the dependencies for different releases and compile modes.

A simplistic implementation of ASK would mean that its maintenance would turn into a full-time job, which is unacceptable. Thus, ASK is coded in such a way that all expected changes are properly taken into account when they occur, that a solution that has been successfully applied before is applied again if the problem resurfaces, and that it has a "Plan B" for most tasks. This paper describes in detail how this robustness is achieved.

2. WORK MODEL

There are three main aspects to everything that ASK performs: first, setup a clean-room to work in; then, decompose the task at hand; and finally, always be ready to handle failure.

2.1. The Clean-Room

There is a whole score of problems related to improper configuration of the working environment by the end-user. In order to keep full control of the environment, ASK does not require any actions outside its own process (or any of its subprocesses). Although the user needs to have full access to the environment in case all else fails, in general she is shielded. ASK accomplishes this by executing a working shell in the background, to which it establishes open pipes for input and output, and it then works on this shell as if

it was itself just another end-user. This shell is its "clean-room," and Ask has full control: all communication proceeds through an API, which allows Ask to verify and validate any commands, their progress, and their results. It is not uncommon to allow commands that may cause havoc, and simply correct settings afterwards.

A shell that is started from the current process inherits the environment and configuration of the current process, which can be invalid to begin with. Note however, that what may seem like an invalid configuration to Ask may in fact be an experimental setup from an expert user, who really wants to have the environment setup this way in order to see what happens. Therefore, Ask can not simply override any existing configuration. Instead, it accepts the settings from the user and only provides what appears to be missing. This is a very productive and pleasant way of working: as an expert user, you configure only that part of the system that you are interested in, and Ask takes care of the rest.

In practice, it turns out that for beginning and casual users a different approach is needed: many of them have residual configuration in their login files. This usually does not cause any problems at first, but sooner or later these settings will go out of date, leaving the user puzzled when things start to mysteriously fail. The user will, typically, blame[1] this on Ask and may decide to drop the tool in disappointment. Consequently, Ask comes in two modes: expert and non-expert (the latter is the default). In non-expert mode, Ask removes any known Atlas software configuration from the environment before executing any of its own code. This way, Ask will operate as if it started from a clean environment and perform the whole setup and configuration itself.

Regardless of whether the user or Ask has done the configuration, every task is executed from a known state: Ask enters a base directory, from where the task can move into its working directory, and it locates the required resources and verifies their configuration, repairing the setup as needed (taking guidelines from experts, overriding non-experts). Each task can then be implemented in a straightforward way, since it will, by design, not get executed until it has a very good chance of succeeding.

If an environment setup is irreparably broken, or if a critical resource is unlocatable, Ask has no choice but to stop. However, since such a problem occurs in very specific cases, it is often (but not always) possible to present a clear and concise error report to the user. See Section 2.3 for a more detailed discussion of error handling by Ask.

---

[1]In fact, it used to be that *every* Ask bug report was resolved by simply having the user clean up his environment.

2.2. Task Decomposition

Many of the tasks performed by Ask use the same tools and require similar resources. Further, steps within individual tasks sometimes overlap and can consequently be shared. In fact, most tasks share the same initialization and finalization step. Thus, it makes sense to implement the tasks in a layered execution model:

- **Encapsulate tools**
  Each task has access to the execution shell, but only standard UNIX commands are executed directly. Tools that are part of Atlas software, however, are normally[2] used directly. Instead, tools are encapsulated in modules, which enables Ask to locate tools and to verify their setup and possibly update or repair it, before any attempt is made to perform actions with the tool. This is the lowest layer.

- **Refactorize steps**
  It is a good programming practice to minimize the amount of duplicate code and in Ask this means that all steps that are shared between tasks are refactored. Further, the order of execution of steps is also often the same. By design many actions need the same set of parameters, and it is therefore straightforward to use meta-programming to create templates that are parametrized on the core action of a task. This guarantees that the execution order is proper, which greatly simplifies the implementation of individual steps. This is the programmatic layer.

- **Construct recipes**
  The end-user typically does not think about tasks in the same granularity as tool authors do. Thus, recipes, that are end-user tasks, are constructed from the programmatic tasks. Since programmatic tasks are supposed to be self-contained, they can be freely mixed and matched in the recipes: it is only the goal of the recipes that puts constraints on their order. This is the user layer.

- **User interfaces**
  The recipes typically take input and have output. The former needs to be collected from the user, and the latter needs to be presented back to the user. The user interfaces connect input/output elements with recipes to present the

---

[2]Similar to the setup of the environment, the user can access the tools directly if all else fails

ASK functionality to the user. There is a graphical interface and a command line interface. This is the application layer.

Third party programs or scripts, can access ASK functionality at any layer, but do so most commonly at the programmatic and/or user layer. These programs are typically similar in setup as the user interfaces (in effect, they are also located in the application layer). For example, the ASK distribution includes a program that performs full Atlas software builds and is targeted at software librarians. But in principle, it is just a batch version of the command line interface with a few different defaults and a few extended recipes.

2.3. Error Handling

There are several different kinds of errors that ASK can encounter:

- **Random.**
  Errors due to (temporarily) missing resources such as disk access, KERBEROS tokens, the tag collector, etc. These errors come and go and, unfortunately, the solution is usually "come back later, and try again" or requires some user intervention. Reporting is often, but not always, straightforward. For example, if a KERBEROS ticket is missing and ASK attempts to access a file in the code repository, it will be CVS that fails and ASK will only have an error from that tool to report, thus masking the real error.

- **User actions.**
  Errors because of a file that does not compile, an executable that does not link, an option script that the user requested but that does not exist, etc. Reporting these errors is simple: ASK needs to retrieve the error code, stop the task, and report the error message from the tool that failed.

- **Broken system.**
  Errors as a result of misconfiguration of the Atlas software distribution that is used. There is not much point in reporting this kind of error, unless the user is testing the distribution. Instead, ASK attempts to work around the problem. If that fails, then the requested task can not be performed.

It is the last item that is the most interesting, because in handling that category of errors there is the most to gain in terms of productivity for the end-user, especially if he or she is a novice or casual user. In order to deal with broken systems, ASK has several options. First of all, it may come prepared: for several releases and configurations, it comes with specific workarounds that are simply programmed into ASK. That way, the problem will not appear in the first place, because the workaround is the default.

It gets more interesting when a task does fail. In that case, ASK will try a few more things that might work, such as locating an alternative resource, querying an alternative source, or simply accepting a default and hope for the best. The ticket here is that this is precisely how an end-user would attempt to tackle the problem herself, with the exception that ASK can perform the trials much, much faster, and it has therefore a good chance of arriving at a solution with the end-user never knowing that there was a problem.

As a final resort, ASK sometimes explicitly fixes a workspace. Since this usually involves moving or replacing files and/or directories, this is done only when the case is very clear.

Note that as before, when all else fails, ASK does allow expert access at all levels, at any time.

3. IMPLEMENTATION

One of the stated goals of ASK (see section 1) is the integration of the framework, the releases, and the release tools. The natural choice is then to implement ASK in a scripting language, because that is the only way to effectively and productively integrate such a variety of tools and environments. The prevalent scripting language in Atlas software is PYTHON [3], which is e.g. used for the scripting facilities of the ATHENA framework. PYTHON is often praised for its effective use as a "glue" language, because of its extending and embedding features, and the choice for PYTHON is therefore natural.

PYTHON is portable, interpreted, and gives you a lot of functionality per line of code because of its extensive set of ready-to-use modules that come with every standard installation. That, and the fact that most (if not all) UNIX distributions actually ship with a version of PYTHON, means that all the user needs to do to install ASK, is to make sure that the scripts are located in a directory that is in included in their PATH environment variable. This is the case for all Atlas accounts at CERN, because ASK has been installed in the official scripts directory. Consequently, all Atlas users have immediate access to ASK without the need of any further setup.

4. USER INTERFACE

This section describes how the ASK functionality is presented to the user. The details of the individual commands, buttons etc. are explained in the manual [4], which is included with the distribution, and are not further elaborated here.

There are two user interfaces in ASK: a graphical user interface (GUI) and a command line interface (CLI). The former is meant for beginning and casual users, as well as for tutorials; whereas the latter is more attuned to advanced users, developers, and librarians.

4.1. Graphical User Interface

Historically, the first usage of ASK was to generate an ATHENA skeleton package. In order to be able to capture all user-configurable parameters at once, a small graphical interface interface was developed. This proved especially practical for package updates, since the current information could be displayed, ready for the user to modify. For educational purposes, a pseudo-shell, displaying the underlying commands that ASK executes, was added to the GUI. A sample view of the full GUI is shown in Figure 1. On the left, the user can find all tasks, grouped by category in notebook tabs; on the right is the pseudo-shell, where the user can see the commands that ASK executes and their the results. In principle, if the user's shell environment is clean, these commands can be repeated on a real login shell to achieve the same results as with ASK.

The categorization of tasks is chosen in such a way as to make sure that a user only rarely needs to leave the current screen. In fact, the screen selected on startup is different for an empty work directory (where the user will most likely want to create or checkout a new package), than for a directory with existing packages (where the user will most likely want to work with any of these packages). There are no dialog boxes in ASK: all input, which is minimal and completely optional by design, is collected directly in the current screen, and all output messages are in a log window that is always visible.

Integration of interactive ATHENA is via the pseudo-shell. The entry box on the lower right automatically activates when the ATHENA prompt is detected, and the user can enter input to ATHENA there.

4.2. Command Line Interface

The feature set available through the CLI is the same as the one available through the GUI. But there are two main advantages, for proficient users, to the CLI over the GUI:

- The CLI is really just the standard PYTHON interpreter. Thus, if the user is well versed in PYTHON, she can use any and all functionality available in PYTHON. Furthermore, rather than typing all commands, sets of commands can be collected in a script which is subsequently executed in the interpreter.

- Access to underlying system internals, the shell environment, build and release tools, etc. is more natural than in the GUI. One of the options available to the expert is starting a login shell as a subprocess of ASK. This shell will inherit a properly setup environment, but otherwise acts the same as if ASK wasn't there.

An additional advantage, for all users, is that the CLI can be used over a poor network connection[3] where the GUI is prohibitively slow. Finally, the CLI includes a few "über-recipes," e.g.:

```
[lxplus] mkdir work; cd work
[lxplus] ask
==>      : welcome to Athena Startup Kit v1.1
==> NOTE: enter "help()" to receive help
>>> run( 'AtlfastOptions.txt' )
```

will perform all steps necessary to run the Atlas fast simulation from the latest official release (for any other package, all the user needs to do is specify the appropriate, released options file). It absolutely does not get any easier than this!

Integration with interactive ATHENA is natural as long as the user realizes that ATHENA is run from a separate PYTHON process that does not share any functions or data with the ASK process: the user simply sees a prompt switch and can just continue typing.

5. APPLICABILITY

Naturally, ASK is used by the author himself, both to run Atlas software as well as when working on the development of the ATHENA framework. The typical ASK user, however, is a beginning user, because of the exposure of ASK to the Atlas collaborators through the software tutorials. The reconstruction tutorial [5] is used by many as their first starting point with Atlas software: the tutorial shows how to roll your own ATHENA algorithm and how to access reconstruction objects in the transient data store. That is the typical for how most end-users actually work, thus the reconstruction tutorial is popular.

ASK has been in use for some time for the builds of the Atlas software release at LBNL. This use exploits the ASK error handling in a creative way: when resources are missing (which they by definition are, since the build has yet to take place), ASK will fall back on CERN, where the release has already been built, since no build is attempted at LBNL before

————

[3]With the new version of ASK, which can be installed locally while issuing commands on a secure shell to a remote host, this is less of an issue.
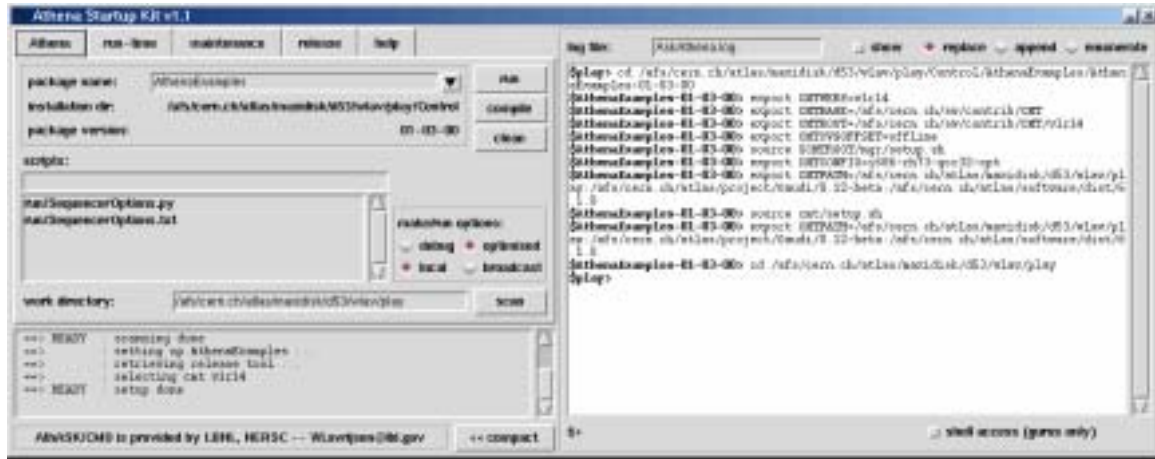
Figure 1: The main Athena Startup Kit graphical user interface.

that. Thus, ASK can fully automatically retrieve the latest settings, versions of tools, and targets from the CERN release and apply them as appropriate when building the LBNL release.

6. OUTLOOK

It is clear that ASK is currently the best way to get started with Atlas software. Getting something to run successfully, and being able to look at some output files with data in them, is a great motivator for users new to Atlas. When its abundant documentation, its use in tutorials, and its ready availability are taken into account, then ASK's value is obvious.

As Atlas software stabilizes, and as parts of it get replaced by higher quality software, the number of needed workarounds can be trimmed: at some point this will render most of ASK obsolete. Elements of it, however, will remain relevant: creating new packages, updating algorithm and requirements files, its tutorial aspect, etc., all have a long life expectancy. It is foreseen, that these parts will be packaged as software components and transferred to the GANGA [6] project, which has already benefitted from the design ideas from ASK.

There will be no major additions of functionality to ASK, because of this expected integration with GANGA. For example, ASK does not currently have the capability to submit jobs to batch systems, even though it is relatively simple to add such a feature based on the generated standalone execution scripts.

Instead, ASK will be maintained to handle future distributions of the Atlas software and to improve on its existent functionality. It will keep serving its current audience until it gets replaced by a version of GANGA that has equivalent or better functionality.

Acknowledgments

References

[1] Atlas Collaboration, "Atlas - Technical Proposal", CERN/LHCC94-43, CERN, December 1994.
[2] LHC Study Group, "The LHC conceptual design report", CERN/AC/95-05, CERN, October 1995.
[3] G. van Rossum and F. L. Drake, Jr. (eds.), "Python Reference Manual", Release 2.2.2, PythonLabs, October 2002; http://www.python.org
[4] http://cern.ch/wlav/athena/athask/index.html
[5] http://cern.ch/wlav/athena/athask/apb.html
[6] Ganga Project Group, "GANGA: a user-Grid interface for Atlas and LHCb", these proceedings. http://ganga.web.cern.ch/ganga