# Athena Developer’s Guide

Athena S&RO Simulation, V6.3

October, 2015

William H. Duquette

Jet Propulsion Laboratory

*Copyright 2008-2015, by the California Institute of Technology. ALL RIGHTS RESERVED.*

*United States Government Sponsorship acknowledged.*

Table of Contents

Athena Developer’s Guide 1

1. Introduction 3

1.1 Supported Platforms 3

1.2 Required Tools 3

2. System Components 6

2.1 Kite 6

2.2 Mars 7

2.3 Athena 8

3. Setting Up The Development Environment 9

3.1 Install Third-Party Tools 9

3.2 Building Kite 9

3.3 Building Mars 10

3.4 Building Athena 10

4. Changing Athena Code 12

4.1 Required Background 12

4.2 Changes to Kite 12

4.2.1 Fixing a bug in kiteutils(n) or kitedocs(n) 12

4.2.2 Fixing a bug in kite(1) 13

4.3 Changes to a Mars library package 13

4.4 Changes to Athena 13

4.4.1 Fixing a bug in a library module 13

4.4.2 Fixing a bug in an application 13

5. Writing Software Mods 15

## Introduction

This document attempts to be a top-level developer’s guide for the Athena system. It covers:

* Supported platforms
* Required tools
* System components
* Setting up the development environment
* Building Athena and its components
* Development documentation roadmap

It does not attempt to describe Athena or its models, or how the Athena applications are structured internally.

### Supported Platforms

The Athena software is known to build and run on Windows, Linux, and OSX. We have personally built and run it with the following configurations:

**Windows 7**. We use Windows 7 as our primary development environment. Our systems are 64-bit; we compile Athena as 32-bit application so that it will run on any version of Windows 7. We would expect it to run adequately on later versions of Windows as well, but we have not had the opportunity to test that. In particular, we have never run Athena on Windows 10.

**64-bit Red Hat Enterprise Linux 5**. This was the standard platform for JNEM development, and we have continued to use it for Athena development. We build Athena as a 64-bit application on Linux, so it will only run on 64-bit installations. (64-bit Linux appears to be the standard these days.) It should build and run on later versions of Linux as well.

**Macintosh OS X**. We have built Athena on OSX 10.7 through OSX 10.10 (Yosemite). The Athena Workbench GUI has not been tuned for OSX, and has a number of mostly cosmetic defects and anomalies; we do not recommend OSX as a Workbench target at this time. Other parts of the system, including Arachne, function quite well on OSX.

### Required Tools

The following third-party tools are required to work with Athena in development.

**ActiveTcl 8.6**. Athena is implemented largely in the TCL/TK language, and depends on ActiveTcl, ActiveState’s distribution of TCL/TK. We are currently working with ActiveTcl 8.6.4. ActiveTcl can be downloaded and used for free from **ActiveState.com**.

**TclDevKit 5.1 or later**. TclDevKit is a set of TCL development tools sold by ActiveState; it is available from <http://activestate.com>. The most important tool in TclDevKit is “tclapp”, which we use to build standalone binaries. Each developer working with Athena will need a TclDevKit license costing about $300. The license allows the specific developer to use TclDevKit on any supported platform and any number of machines.

**Various Tcl Packages from ActiveState’s Teapot**. ActiveState builds a wide variety of open source TCL library packages and makes them available in a repository at <http://teapot.activestate.com>.[[1]](#footnote-1) ActiveTcl’s license allows us to make use of these packages; and the TclDevKit license allows us to distribute them as part of a standalone binary built using “tclapp”. Athena uses two dozen or so of these packages; if we could not get them from the “teapot” we would need to build them ourselves, which in some cases would be quite difficult (and indeed, we’ve never done so).

**Git**. Git is a distributed version control system (DVCS). We use GitHub’s software; JPL has its own GitHub4Enterprise server. We especially use GitHub’s issue tracker. If we manage to open source Athena, it will go on up GitHub.com. However, the basic Git command line tool can also be used.

**A Programmer’s Text Editor**. There is no good IDE for TCL/TK development. (We have tried and abandoned ActiveState’s Komodo IDE.) A good programmer’s text editor is thus a must, ideally one that displays the project tree in a sidebar. We’ve been using SublimeText 3.

On Windows, two further tools are required:

**MinGW (Minimalist GNU for Windows)**. This is a Linux-like tool chain for Windows, providing GNU make and the GNU C compiler, both of which are required for Athena development. In addition it provides a development environment called MSys, which provides a **bash** command shell. It is available from <http://www.mingw.org>.

Note: There are other Linux-like environments for Windows, notably Cygwin; these will not work for Athena development. First, executables and libraries built using Cygwin will only run in the Cygwin environment. Second, we are dependent on the standard TCL/TK build environment for Windows, which is MinGW.

Note: by default, MinGW/MSys creates an MSys home directory for you in a special place in its directory tree. You’ll probably be happier if MSys uses your Windows home directory instead. To do this, go into the User Accounts control panel, and select “Change my environment variables.” In that dialog, define the environment variable “HOME” to be “C:\Users\yourname” **before** you install MinGW. In my case, for example, it’s “C:\Users\will”.

**InnoSetup 5**. This is an installer compiler; we use it to build Athena’s Windows installer. It is available from <http://www.jrsoftware.org>.

## System Components

The Athena system has three major components (from a developer’s point of view): Kite, Mars, and Athena proper. Building Athena requires building each of these packages.

**Note on Man Page Notation**: Athena documentation makes considerable use of the Unix notion of a “man page”. Each man page documents one thing, and resides in a specific section of the “manual”. Man pages are referred to using man page notation: “*pagename*(*section*)”.

Section 1 documents command-line commands (i.e., applications); Section 5 documents file formats. In the TCL community, TCL commands and APIs are documented in Section “n”; and for Athena, TCL interfaces implemented by multiple TCL objects are documented in Section “i”.

Thus, the documentation for the Athena command line tool is in the **athena(1)** man page, and the documentation for the Athena scenario library is in the **athena(n)** man page. A component’s man pages can be found by opening the component’s docs/index.html file in your web browser.

### Kite

Kite is a project automation tool for TCL/TK projects written using ActiveTcl and TclDevKit. It also defines some infrastructure packages.[[2]](#footnote-2) It is written in TCL/TK and provides the following:

**kite(1)**. This is the Kite command-line tool, which is used to build Mars and Athena (and also Kite itself). It provides a number of services to the developer, including:

* Create a project tree for a new TCL/TK project
* Add an application or package to an existing TCL/TK project
* Format the project’s HTML documentation, including man pages
* Run the project’s test suite
* Build the project and create distribution sets

Many of the functions that would be provided by an IDE are provided by **kite(1)** instead.

**kiteutils(n)**. This is a package containing a number of modules of basic TCL code used primarily by Kite but also by Mars and Athena.

**kitedocs(n)**. This is a package of code that supports kite(1)’s documentation tools. It is used primarily by kite(1), but Athena’s on-line help system also makes use of it.

To learn more about the Kite system and how to use it to manage TCL/TK projects like Kite, Mars, and Athena, point your browser at **docs/index.html** in the Kite directory, and then click the link for the *Kite Developer’s Guide*. (You will need to build Kite first; see Section 3.)

You will also find the *Kite White Paper*, which explains the problems Kite is trying to solve and how it solves them. Kite has evolved since the paper was written, but it still provides useful background. You will also find the Kite man pages.

### Mars

Mars is a collection of TCL/TK packages; originally, it was the infrastructure shared between Athena and JNEM. It provides the following tools:

**mars(1)**. This the Mars command-line tool; it provides a number of subcommands that can be useful when working with the Mars code base. For example, the command “**mars uram**” brings up the URAM Workbench, an Athena-like application for interacting directly with the URAM model.

**marsutil(n)**. This is a package of basic infrastructure code, augmenting that found in **kiteutils(n)**. It includes a wide variety of important code, from basic utilities and data types to the infrastructure for Athena’s order processing code.

**Marsbin(n)**. This is a subpackage of marsutil(n) consisting of C code. It includes geometry, coordinate conversion, and image processing code, and provides TCL bindings for these libraries, augmenting the pure-TCL code in marsutil(n). In particular, it provides the conversions between latitude/longitude and MGRS grid references, and the ability to load maps in GeoTIFF format.

**marsgui(n)**. This is a package of basic GUI code; it is used by Athena workbench and other Athena GUIs.

**simlib(n)**. This is a package of simulation code, including URAM, the Mars Affinity Model, and the cell model infrastructure used to solve Athena’s economic model’s CGE.

To learn more about the packages provided by Mars, point your browser at **docs/index.html** in the Mars directory. (You will need to build Mars first; see Section 3.) In that directory you will find:

* The *Mars Analyst’s Guide*, which describes the models implemented in Mars.
* The *Mars CM Plan*, which explains the CM procedures for Mars, including the official build procedures.
* The Mars man pages.

### Athena

Athena is the Athena simulation proper; it provides the following tools and packages.

**athenawb(1)**. This is the Athena Workbench GUI, i.e., “Classic Athena”.

**athena(1)**. This is a command line tool; it primarily exists to drive Athena code in an *ad hoc* way during development, but it also provides the **athena build** command, which allows Athena scenarios to be run on “headless” nodes in the cloud.

**arachne(1)**. This is Athena’s web back-end, allowing another application to load, run, and query scenarios over HTTP. Arachne accepts connections only from processes on the same physical host.

**athena\_log(1)**. An external browser for Athena’s log files. Occasionally useful.

**cellide(1)**. An IDE for editing cellmodel(5) files like Athena’s CGE.

**helptool(1)**. A tool used at build time to build Athena’s help databases.

**ahttpd(n)**. A TCL package containing a Pure-TCL web server, used by Arachne.

**athena(n)**. A TCL package containing the Athena models, algorithms, and scenario management code. This is the heart of Athena.

**projectlib(n)**. A TCL package containing infrastructures modules used by the various Athena packages and applications.

**projectgui(n)**. A TCL package containing GUI code used by the Athena Workbench and other GUIs.

**app\_*application*(n)**. TCL packages containing the application code for the Athena applications listed above.

To learn more about the packages provided by Athena, point your browser at **docs/developer.html** in the Athena directory. (You will need to build Athena first; see Section 3.) In that directory you will find:

* The *Athena Analyst’s Guide*.
* The *Athena User’s Guide*.
* The *Athena Rules* document.
* The *Athena CM Plan*, which explains how to build Athena for testing and release.
* The Athena man pages.

## Setting Up The Development Environment

This section explains how to set up the development environment to build Athena.

**NOTE:** These instructions presume that your development machine is connected to the Internet. If it is not, setting up the development environment will be much more difficult (and, in fact, we’ve never tried to do so).

### Install Third-Party Tools

First, install the third-party tools listed in Section 1.2. When you’ve done so, you should able to do the following:

* Open a terminal window and verify that you can navigate your file system using the **bash** shell.[[3]](#footnote-3)
* Invoke a TCL shell using the **tclsh** command, and query its patch level. The patch level should match that of the version of ActiveTcl you installed.

$ tclsh

% info patchlevel

8.6.4.1

% exit

$

* Verify that you can invoke **tclapp** from the command line.

$ tclapp -help

…

$

### Building Kite

* First, use Git to clone the Kite repository to your ~/github/kite directory.
* Next, follow the directions in ~/github/kite/docs/BUILD.md to build Kite and install it for use on your system.
* You should now be able to use Kite normally from the command-line:

$ kite version

Kite 0.5.0

$

### Building Mars

* First, use Git to clone the Mars repository to your ~/github/mars directory.
* Next switch to that directory:

$ cd ~/github/mars

$

* Next, use Kite to update Mars’ dependencies.

$ kite deps update

…

$

* Next, use Kite to build Mars.

$ kite build

…

$

* Finally, use Kite to install the Mars library packages and command line tool into your environment.

$ kite install

…

$

The procedures for various CM-related activities, including builds, are documented in the *Mars CM Plan*; open the file ~/github/mars/docs/index.html in your browser, and follow the link.

### Building Athena

* First, use Git to clone the Athena repository to your ~/github/athena directory.
* Next switch to that directory:

$ cd ~/github/athena

$

* Next, use Kite to update Athena’s dependencies.

$ kite deps update

…

$

* Next, verify that you can run Athena as a Tcl script.

$ ./bin/athenawb.tcl

The procedures for various CM-related activities, including builds, are documented in the *Athena CM Plan*; open the file ~/github/athena/docs/developer.html in your browser, and follow the link.

## Changing Athena Code

Eventually you will want to modify some aspect of Athena’s behavior; this will require changing source code. This section explains some things you’ll need to know.

### Required Background

First, you will need a working knowledge of these fundamentals:

* The TCL/TK language, including
  + The TCL/TK package system
  + TCL/TK’s TclOO object system
  + The **tcltest(n)** Test Framework
* The Snit object system (start with the Snit FAQ)
  + Man page: <http://core.tcl.tk/tcllib/doc/trunk/embedded/www/tcllib/files/modules/snit/snit.html>.
  + Snit FAQ: <http://core.tcl.tk/tcllib/doc/trunk/embedded/www/tcllib/files/modules/snit/snitfaq.html>.

Next, read the *Kite Developer’s Guide*. It explains how the Kite, Mars, and Athena project trees are laid out, where files are found, how the documentation is written, and in general how things are done. If you learn how Kite works and what it does for you and what it expects, everything will go more smoothly.

### Changes to Kite

It might be necessary to fix a bug in one of Kite’s modules, whether in the Kite application or in one of its library packages.

#### Fixing a bug in kiteutils(n) or kitedocs(n)

* First, identify the code to be changed in lib/*libname*.
* Next, identify the matching test script in test/*libname*.
* Modify the code as needed.
* Use kite test to run the test suite.
* Add tests to the test script as needed.
* Update the given module’s man page in docs/mann as needed.
* Use kite docs to convert the updated man page to HTML, and check the result.
* When all looks good, build and install Kite using kite.tcl, as described in Section 3.2.
* The changes are now available to Mars and Athena.

#### Fixing a bug in kite(1)

* First, if the bug is in a library package then see Section 4.1.1.
* Next, identify the module in lib/app\_kite.
* Modify the code as needed.
* Verify that it works by running kite.tcl.
* When all looks good, build and install kite using kite.tcl, as described in Section 3.2.
* The changes are now available for general use.

### Changes to a Mars library package

* First, identify the code to be changed in lib/*libname*.
* Next, identify the matching test script in test/*libname*.
* Modify the code as needed.
* Use kite test to run the test suite.
* Add tests to the test script as needed.
* Update the given module’s man page in docs/mann as needed.
* Use kite docs to convert the updated man page to HTML, and check the result.
* When all looks good, build and install Mars using kite.tcl, as described in Section 3.3.
* The changes are now available to Athena.

### Changes to Athena

#### Fixing a bug in a library module

* First, identify the code to be changed in lib/*libname*.
* Next, identify the matching test script in test/*libname*.
* Modify the code as needed.
* Use kite test to run the test suite.
* Add tests to the test script as needed.
* Update the given module’s man page in docs/mann as needed.
* Use kite docs to convert the updated man page to HTML, and check the result.
* Do an official build, per the *Athena CM Plan*, as needed.

#### Fixing a bug in an application

* First, if the bug is in a library package then see Section 4.3.1.
* Next, identify the module in lib/app\_*appname*.
* Modify the code as needed.
* Verify that it works by running the application. Also, note that many of the applications have test suites that are run by kite test.
* Do an official build, per the *Athena CM Plan*, as needed.

## Writing Software Mods

A “mod” is a TCL file that can be loaded into a fielded version of Athena to fix or test a bug before a new official release is built and distributed. The Athena **mod(n)** man page explains how to write a software mod file, and where to put mods so that they will be found by the installed Athena software.

Mod files target particular TCL packages. Thus, if a bug is found in Athena’s **projectlib(n)** package the mod must target package “projectlib”. Then, the mod will be automatically picked up by any Athena application (e.g., the Workbench or Arachne) that uses **projectlib(n)**.

The content of a mod file looks like this (see **mod(n)** for details):

**mod** *package version number title* {

*body*

}

The *body* is a TCL script that modifies the software. The process works like this:

* All of the application’s code is loaded.
* All mods are read from disk and applied.
* Application execution begins.

A mod can define new procedures and other code objects, and can redefine existing procedures, object methods, and so forth:

* Locate the routine that needs to be changed. It might be a TCL proc, Snit method or typemethod, or TclOO method.
* Copy it into the body of the mod as is.
* Methods and typemethods require more work; see below.
* Edit the routine to make the fix.

Object methods and typemethods are defined in the context of a particular Snit type or TclOO class definition; the definition a method in the source code usually looks like this:

method somecode {a b c} { … }

If you copy this into the mod as is, you will get an error when the mod is loaded: TCL doesn’t know the type or class to which the method belongs. Instead, you need to use the appropriate syntax for defining a single method outside of a type or class definition, per the Snit and TclOO man pages. For example, to define a single Snit method outside of its type, you write code like this:

snit::method mytype somecode {a b c} { … }

1. Don’t bother pointing your browser at this address; it is accessed using tools included with ActiveTcl. [↑](#footnote-ref-1)
2. Reusable Tcl libraries are called “packages”. [↑](#footnote-ref-2)
3. Usage of the **bash** shell and of the Git version control system are beyond the scope of this document. [↑](#footnote-ref-3)