

Astrolab

v0.965

User Guide

Copyright

Copyright in this document is the property of C. J. Gaudion.

© Copyright 2011

Contents

1. Introduction.....	4
1.1. Current Position.....	4
2. Current Version 0.965.....	6
3. Installation.....	6
4. Starting the Application.....	6
5. Application Contents.....	7
5.1.1. Folders.....	7
5.1.2. Framework.....	7
5.1.3. Applications.....	8
5.1.4. Data Pages.....	8
5.1.5. Equation Pages.....	8
5.1.6. Information Pages.....	9
5.1.7. Utility Pages.....	9
5.1.8. Data Sets.....	9
5.1.9. Demo.....	9
5.2. Stage – Preliminary Assessment.....	14
5.2.1. Basic requirements of this project.....	14
5.2.2. Why re-invent the wheel?.....	15
5.2.3. How difficult will this be?.....	15
5.2.4. Why develop in Open source + Cross Platform Development.....	16
5.2.5. Preliminary Assessment Summary.....	16
6. Phase – Planning.....	17
6.1. Stage – Scope.....	17
7. Phase – Requirements.....	18
7.1. Stage – Specification for Platforms used in Development.....	18
7.2. Stage – Detailed Specification – Supporting Web Site.....	19
7.3. Stage – Detailed Specification – Astrolab Application.....	20
8. Phase - Analysis and Design	22
8.1. Astrolab Application Simple Design Specification.....	22
8.2. AstroEquations.xls.....	22
9. Phase – Implementation.....	22
10. Phase - Deployment.....	22
11. Phase – Testing.....	23

12. Phase – Evaluation.....	23
13. Appendix A - JavaScript Style Guide.....	24
13.1.1. Code Style.....	24
13.1.2. Comments	25
13.1.3. Commenting Scripts in JavaScript.....	25
13.1.4. Form Fields.....	25
13.1.5. Function Declarations	26
13.1.6. Indentation	28
13.1.7. JavaScript Files	29
13.1.8. Names	29
13.1.9. Quotes.....	30
13.1.10. Statements	30
13.1.11. Simple Statements	30
13.1.12. Compound Statements	30
13.1.13. Labels	30
13.1.14. Symbols.....	32
13.1.15. Variable Declarations	33
13.1.16. Variable Scope and Lifetime.....	34
13.1.17. Whitespace.....	34
14. Appendix B - Recommended Development Tools.....	36
15. Original Specification for Astrolab.....	37

1. Introduction

The idea creation stage of this project can now be considered complete. The project was started to combine years of experience supporting and developing computer applications with the study of Astronomy and Astrophysics in an effort to document the development and support of a related open source application from start to finish, with the **requirement to produce a useful end result, while keeping costs to an absolute minimum.**

The end result of the process should be a web site to show how a useful working application could be developed by anyone with little knowledge (but at least an interest) of setting up a PC in detail, selecting the development tools, or setting up a supporting website and maintaining the whole system.

The supporting web site will also attempt to document the best resources for the study of Astronomy / Astrophysics generally as well as those that compliment the final application and its development.

This document contains a description of the whole Conquest of Space resource project as well as requirements and specifications for the website and the Astrolab application development, this cuts down on the number of different documents that can often be generated in a large formal project, and we have no wish to bore readers if it can be avoided.

Astronomy, Astrophysics and Cosmology have differing definitions. Astronomy can be described as the observational subject, Astrophysics attempts to explain the observations and Cosmology is the study of the Universe as a whole. As there is much overlap between in the subjects in books and courses we will usually describe everything under 'Astrophysics' for the purposes of this project.

1.1. Current Position

Phase	Stage	
1) Initial Software Requirements Spec	Idea Creation	
	Scope	
	Preliminary Assessment	
2) Define / Refine Detail design and spec	Prototyping as required	
	3) Build v0.965	<<
3) Build v0.965	Code / Construct	
	Unit Test	
	Integration Test	
4) Function, systems, performance	System Test	
5) User Test	Package	
	Deploy	
	Test	
6) Update software requirements Spec		
7) Update solutions architecture	Fixes or new features required	
Return to phase 2 for	Version 0.966	

Colour coding

Complete	In Progress	Not Started
----------	-------------	-------------

Formal project management can get tied up with a lot of documentation and the above is extracted from a document on a framework for application development. However we have no intention of getting bogged down with that framework and a lot of boring documents – 'keep it simple!'

Once it gets to the Execution stage and development actually starts then the same table format can be used to track progress on individual parts of the whole project if necessary.

The above table can be used to track the progress of a project overall and follows the Iterative or Incremental development model shown in the diagram below.

2. Current Version 0.965

Astrolab is currently at version 0.965 and still in the 'Beta' stage of development.

It is web browser based and designed to be Web Standards compliant and work with latest versions of Firefox, Safari and Chrome. All pages HTML code is validated to the XHTML 1.0 Transitional standard.

IE is no longer actively supported (in that we do not believe in wasting effort and complicating coding with browser specific commands – the browser must be web standards compliant) as it does not support vector graphics (SVG) files as a default – only with an unsupported plugin, however 99.9% of the application may work.

Browsers - current Web standard compliant ratings – using the Acid 3 test.

Browser	Platform	Version	Acid 3 Rating
IE 7	Windows	7.0.5730.13CO	
Firefox	Windows		
	OSX		
Safari	Windows		
	OSX		
Chrome	Windows		
	OSX	(G4 platform not supported)	

3. Installation

For the Windows version - Unzip the complete folder and place anywhere, the application is self contained and web browser based.

The OSX version uses Apple's own tools to package the application ready to copy to a disc image type file that can be downloaded and installed via their own install process into the usual Applications folder.

4. Starting the Application

mMain.html starts 'mobile device version' - reduced graphics, simple menus, each page displayed in full, no frames

main.html starts full version in frames

auto.html attempts to detect 'device' to load appropriate version

fDefault.css and fEnvironment.js contain parameters to set font size and layout, mobile device version usually being set to have a smaller default font.

5. Application Contents

Artefacts

Below is a list breaking down the current functions in Astrolab. Colour code shows state, matching the individual page header, and as shown on the web site.

Being coded / Prototype	User Test Version	User Version
-------------------------	-------------------	--------------

5.1.1. Folders

Folder	Description
/css	Style sheets controlling page layout and font size
/datasets	Contains data sets for reference and study
/imageLib	Images used in the application
/imageLibMoon	Specific Moon images related to Phase calculations
/documents	Supporting documents
/utility	Misc utilities related to project

5.1.2.

Framework

Page / File	Description
auto.html	Start application and attempt to detect OS and version to start
cConstants.js	Constant values used in the JavaScript
fEnvironment.js	Environment control variables
fAppWindow.html	Main application window
main.html	Start frame view
fMenu001.html	Mobile device version menu 1
fMenu002.html	Mobile device version menu 2
fMenu003.html	Mobile device version menu 3
fMenu004.html	Mobile device version menu 4
fMenuWindow1.html	Frame version menu 1
fMenuWindow2.html	Frame version menu 2
fMenuWindow3.html	Frame version menu 3
fMenuWindow4.html	Frame version menu 4
fTitleWindow.html	Frame version main menu
mMain.html	Start mobile device view
fDefault.css	Default page formatting
cMathLibNew.js	Math lib

5.1.3. Applications

Page	Description
cCalc001Eclipse.html	Solar Eclipse Calculations – past and future
cCalc002Lunar.html	Lunar Phase Calculations
cCalc003Crater.html	Impact Cratering – Crater Size
cCalc003CraterHlp.html	Impact Cratering – Help
cCalc005AstroCalc.html	Astrophysics Calculator
cCalc006JavaEx.html	Java Test
cCalc009Kepler.html	Kepler's Laws of Planetary Motion
cConv001Dist.html	Distance Conversions
cConv002Temp.html	Temperature Conversions Between Scales
cCalc004ConvDeg.html	Convert Degrees/Minutes/Seconds to/from Decimal and Radians

5.1.4. Data Pages

Page	Description
cData001Mercury.html	Planetary Data – Mercury
cData002Venus.html	Planetary Data – Venus
cData003Earth.html	Planetary Data – Earth
cData004Mars.html	Planetary Data – Mars
cData005Jupiter.html	Planetary Data – Jupiter
cData006Saturn.html	Planetary Data – Saturn
cData007Uranus.html	Planetary Data – Uranus
cData008Neptune.html	Planetary Data – Neptune
cData009Pluto.html	Planetary Data – Pluto
cData010Sun.html	Star data – The Sun
cData011StarClass.html	Star Classification
cData012AstroConst.html	Astrophysical Constants
cData013MathConst.html	Mathematical Constants
cData014PhyConst.html	Physical constants
cData015PT.html	Periodic Table of the Elements

5.1.5. Equation Pages

Page	Description
------	-------------

cEqu001Drake.html	Drake Equation
cEqu002Parallax.html	Stellar Parallax
cEqu003Luminosity.html	Luminosity
cEqu004Vol.html	Volume and Area of a Sphere

5.1.6. Information Pages

Page	Description
cCredits.html	Acknowledgements for code and other contributions.
cInfo001EHubble.html	Edwin Hubble original paper
cInfo002SIUnits.html	SI Units – explained as used in the application
cInfo003Parsec	Parsec help
cInfo004HRDiagram	Example HR Diagram
cInfo005Gloss.html	Glossary
cInfo006Equ.html	Equation List
cInfo007Clock.html	
cInfo009SDS.html	Standard Data Sets
cInfo010GNUL.html	GNU Licence
cInfo012Geom.html	Geometry Help

5.1.7.

Utility Pages

Page	Description
cUtil001WebBr.html	Current Web Browser Standard Check
cUtil002WebBr.html	Current Web Browser Information

5.1.8. Data Sets

Folder	Description
300BS	300 Brightest Stars
HRDiagData	
PeriodicTable	Periodic Table data
SolarSystem	Solar System Data
SunSpots	Average Monthly Sunspot number since 1749
YaleBrightStarCatalogue	Yale BSC

5.1.9. Demo

Page	Description
cVectorG001.svg	Under development
cCalc006JavaEx.html	Link to java applet
appletexample.jar	Applet used in the above

AstroEquations.xls

Equations, constants and values used in the Astrolab application. Spreadsheet AstroEquations.xls contains a list and 'test harness examples' of equations to be used in Astrolab. It also lists values for constants and measurements used in Astrophysics, using National Physical Laboratory and CODATA recommended values.

Astrolab Run Time Structure

The principle is that the core of the application will be identical in both the PDA and online / standalone application. How options are selected is controlled by a simple menu system in the PDA version, and by surrounding with a simple html 'frame' with menu options in the 'full' online / standalone version.

main.html starts the 'full' version mMain.html starts the mobile device version.

Auto.html is a page which contains code to try and determine the best version to run based on operating system and browser detected when run from a web server.

		Astrolab Application auto.html start page	
	mMain.html start page		main.html start page
Application	Mobile Online version		Standalone / Online version
Functions selected / controlled by	Simple menu list Framework		Menu driven 'GUI' Framework
Identical Calculations and data	Core application	=	Core application

Changes made in the core will be made in the mobile version and simply copied into the standalone version. There will be just enough code in the core to remove links back to the supporting framework if not applicable to that version, or remove graphics not suitable for the PDA version.

All core applications common to all versions have file names starting with 'c'. Application framework files are specific to their own framework have file names starting with 'f'. HTML file names follow the Java function naming convention of starting with a lower case letter e.g. planetData.html

Application 'windows', menus and calculations are driven by HTML and Javascript.

Development Work Flow

Below is a simple list showing how development and files should flow through the various stages of this project and the development, test and live (user) environments.

```

Ideas / Enhancements  >>  sourceforge tracker list  >>  web site specification
                                                                >>  application specification
                                                                >>  blog news

web site specification  >>  web site dev  >>  web site test  >>  web site live env

application specification >>  app dev env  >>  app test  >>  app live env
                            >>  app user guide  }
                            >>  app readme.txt  }
                            >>  app change log  } >>  app test  >>  app live env

```

Bugs

```

>> sourceforge tracker list  >> web site dev  >> website test  >> web site live env  >> sf tracker
                            >> app dev env  }
                            >> app change log } >> app test  >> app live env  >> sf tracker

```

News

```

>> sourceforge.net  >> web site as appropriate
                    >> blogger.com
                    >> twitter.com

```

Astrolab Build Process

There are no programs to compile at present as the whole application is based on HTML and JavaScript.

All that is required to copy a development release from the development folder to a test or live folder is to:

Clear the destination folder if appropriate, i.e. some files are no longer required and any link to these redundant files may be lost or the files will simply take up space.

If all the files in the new release are either existing files or new files then they will simply overwrite the old versions / create new files when copied.

Check / update the fEnvironment.js file with the version number of the current release to be distributed.

Update the documents\AstroReadme.txt file with changes and version number for this release.

Also add list of main objects to end of .txt file in date modified order. This can be done by pasting list of nightly build objects from web server listing once cleaned up.

Copy the contents of the application dev folder into the application folder under the test web site

Once tested copy the contents of test site application folders into the live web site folders.

Upload all folders to the web server site

Zip the content of the main application into a file to be uploaded to sourceforge.net as a 'Windows' version

Zip data files – if amended – to upload as a separate package as they take up space and are not currently linked directly into project.

Copy main application folder to Mac platform after deleting previous version

Build package on Mac with Package builder and then add to a disc image for upload to sourceforge.net

Update sourceforge.net project news and main blog to indicate new release is now available.

5.2. Stage – Preliminary Assessment

The preliminary assessment stage involves a certain amount of research to basically find out how feasible the idea is, there is no point in jotting down load of ideas for the application to do everything you ever wanted and then get to the programming stage to then find either it just cannot be done because you don't have the knowledge or the development tools are not available or it will be too costly.

We already know we want the application to be web based, open source and run on multiple platforms. The fact that that the web site, and blog, exist mean that some development has already started – this brings us to the subject of Prototyping.

Prototyping is the process of quickly attempting to put together a working model (the prototype) in order to test various aspects of a concept or design, illustrate ideas or features and thus get some idea of how easy the full development will be.

Prototyping is often treated as an integral part of the system design process, where it is believed to reduce project risk and cost. Often one or more prototypes are made in a process of iterative and incremental development where each prototype is influenced by the performance of previous designs, in this way problems, or deficiencies in design, can be corrected.

When the prototype is sufficiently refined and meets the required functionality, robustness, costs, and development tools available are known to be able to cope, then full development can start.

For more information see the Wikipedia entry on Prototyping - <http://en.wikipedia.org/wiki/Prototyping>.

We have also started putting together a specification document (which will be available shortly), again with project management there can be a large number of documents before development even starts, which we are not going to go into here.

In this case we are going to have one sensible document which will keep a note of everything we need as we go along, so once we get to the real specification stage a lot of work will have already been done. At that point we can review the story so far before proceeding to start the final development of the application and we only have to deal with one document.

Prototyping Note: Articles from both the blog and the web site may be included in the document if relevant (although they may be re-edited). Blog entries will also make up the 'latest news' entry on the main web site page, so the blog will act as a news archive. Blog entries may also expand into a more detailed web page on the web site if the author feels they are relevant.

We will now cover more of the subject of preliminary assessment in the following paragraphs.

5.2.1. Basic requirements of this project

As part of the preliminary assessment we need to expand the idea creation and list the basic requirements of the project i.e. what do we want to achieve, here done as bullet points:

- A web site supporting study of Astronomy and Astrophysics and summarising the best web sites that also support this study.
- A web based application to store basic data on astronomical objects and describe and calculate common equations used in Astronomy and Astrophysics.
- Application to be open source so others can contribute and not be restricted to running on one platform (here platform can be hardware **or** operating system **or** both)
- Web site to also detail and support the hardware and software used in the project and application development
- Document the whole project as far as possible to assist those interested in the subjects covered (including written documentation and a blog).
- Minimise costs

(The more detailed requirements for the project come under the planning phase of the project.)

5.2.2. Why re-invent the wheel?

So we have our basic requirements, but before we go too far down the road of developing a new application from scratch (and ignoring the learning side of the project for a moment), is there any application which already does the same thing that would save re-inventing the wheel ?

There are astronomy programs out there that are free, but research so far has not found one which provides what I want, most are star locaters / databases. If there are any web sites or applications for calculations they only do 'bits and pieces'. I don't want to be jumping from one web site to another or installing too many different applications on my PC.

There is also a lot of repetition on the web (hence the silly numbers in Google hits ...) in both information and applications, the idea here is to combine just the best snippets from the web and past astronomy courses in one application and web site.

If an application can be found which complements this one, then it might be used in conjunction with / interfaced to or listed on the website as a recommended resource.

There may be some repetition necessary, but the whole project is a learning experience as well for the developer and the audience.

5.2.3. How difficult will this be?

The web site and documentation should not be difficult; the main problem is it will be time consuming.

The application itself – making it platform independent and open source will mean it is not tied to one platform and open to a wider audience, however it will mean that it has to be developed with tools that are available on multiple platforms.

The growth of the internet and web based applications show that an internet browser based application is the way forward.

We have no requirement here to store large databases, only perform calculations on small amounts of data. Information on the main objects in our Solar system will be sufficient to cover database development, most of the information should fit in the individual web pages / javascript.

At present HTML and Javascript will be the main 'programming' language, so the project could almost be completed with any text editor.

For most purposes the platforms are Windows, Apple OS X and Linux. However the Palm PDA has been added as a platform as it contains its own web browser and if initial development is done for a small screen this will cut the screen detail and code down to the minimum necessary to do the job, so the core application can be developed quickly. Embellishments for a normal PC screen can be added later, but even then, only if they will add to the functionality, so hopefully this will help speed up overall development.

5.2.4. Why develop in Open source + Cross Platform Development

With open source the fact that the original text of the program code is available to all means it is easier to translate from say PC to Apple even if this is not done by the original author. The fact that the applications are mostly free or very low cost can attract users. There is a wide open source community who may contribute further development or fixes for free.

Support may be intermittent depending on how much time the originator has available and how much community support the project has, but generally if there are a number of open source applications doing the same thing, you would pick the most popular and well supported. Testing an application before using or recommending it is important and we will probably cover this subject in more detail later.

There is unlikely to be a vast amount of money made from this project, but then the tools selected to develop it can be open source so there is little cost apart from time as and when available. **As stated in the Idea Creation (and Basic Requirements) costs are to be kept to a minimum.**

Open source does not mean that an application is available on multiple platforms – this is known as cross platform development.

This project will cover both open source and cross platform development, the advantages of cross platform development being that you are not then tied to using one particular machine or even operating system, so you can swap when you choose or as trends change. For example you may be using a PC which may run Windows or it may run Linux, alternatively you may have a recent Apple running Windows or OS X.

For more information on the reasons for cross platform development check out the Guernsey branch of the British Computer Society and their developers forum <http://www.developers.org.gg/> and read Steve Streeting's presentation on the subject.

5.2.5. Preliminary Assessment Summary

As stated in 2.2.2, from current trends and previous experience the actual application should be web based – Web 2 is on its way and developers such as Apple and Google are encouraging complete web based applications. Apple are promoting the development of 'droplets', small applications that can run on the new iPod Touch and iPhone, being browser based they should also run on any platform.

Web pages can be built with simple tools down to a text editor like Notepad. Each web browser itself provides most of the interface. There are cross platform programming language tools which can develop code to run in the web pages, but JavaScript is also a powerful tool already built into the browser and does not need any other 'servers' running in the background.

Not having to use complex expensive development tools cuts cost and we do not want to be tied to using other products that may be required to run on the web server, or locally, (other than perhaps Java) at the same time to support the application; it should be self contained as far as possible.

Many tools required for this project are available as other open source projects on the web either free and / or supported by donations.

Also we don't want to become too reliant on always having a web connection, but again if the application is self contained then web browser based applications can be run from a local drive on a PC or PDA.

On the subject of low cost / free tools check out <http://sourceforge.net/index.php> where there is a vast array of applications available.

6. Phase – Planning

6.1. Stage – Scope

The project will cover the entire process from:

The COS Project can be split into two main sections

- The supporting website – which will cover
 - i) An associated blog which acts as an initial notepad which may then have items expanded on the main web site and also mirrors the main web site links to Astro resources
 - ii) Link to a sourceforge.net web page to explore sourceforge.net facilities in supporting the project
 - iii) Astrolab application to support calculations & development of that application
 - iv) creating the hardware platform
 - v) installing the operating system
 - vi) selecting the software to support the hardware, the website, the application development, and maintain the system
 - vii) developing the website and application
 - viii) documenting of the whole process
 - ix) on going maintenance and development

- The application - Astrolab

The application is intended to provide a tool to cover common astrophysics related data, laws and calculations which can be used on the web site or independently of the web site on any platform. As described in 2.1 Astronomy, Astrophysics and Cosmology are referred to under one heading of 'Astrophysics' for the purposes of this project.

In order to keep the initial amount of work down and keep the project start up under control it will initially only cover the items listed below which will help prove the concept:

- i) Data on planets and the Solar system
- ii) Astrophysics calculator
- iii) Distance and temperature conversions
- iv) A Glossary of common terms used in Astrophysics
- v) Initial development will be for PDA with a desktop / standalone / online version derived from this

Using the Iterative development model further functions can be added later once the initial development is proved to work.

7. Phase – Requirements

7.1. Stage – Specification for Platforms used in Development

Development platform – hardware, basic

Hard drive 80 GB

MSI motherboard / Athlon processor

1Gb memory

Development platform – software

O/S Windows XP SP2

IE 7

Notepad

Open Office

Gimp 2

Dreamweaver 8

Learn XHTML

7Zip

FileZila

Hosting for

Website – UK2.net – conquestofspace.com

Blog – blogger.com - <http://conquestofspace.blogspot.com/>

Project – website and sourceforge.net

Email – alchymylab@gmail.com

Test Platforms

MSI / Athlon

O/S Windows XP SP2

IE 7 + Javascript

Apple Power PC G4

OS X 10.4

Safari + Javascript

7.2. Stage – Detailed Specification – Supporting Web Site

Web support for the application will split into 3 parts

- 1) A blog which acts as a scratchpad for news, views etc which may then be expanded upon in the main web site pages if relevant. All links to relevant web sites and news / notes appear on the blog first. Links are then added to the main web page(s). There is a latest news item link on the main web page which points to the blog.
- 2) A sourceforge.net account to promote the application, supporting web site, and act as a store for downloads, source files and supporting documentation.
- 3) The main web site with more detailed information on the application and how it and the other components are developed and supported, and expanding on the blog records if relevant:

The standards below should be used for the main web site:

- XHTML is now a widely used standard and all pages should meet the W3C XHTML 1.0 Transitional design standard as a minimum.
- Website should be based around a template page so all the pages are standardised for easy maintenance.
- Basically pages should have a header section, menu bar, content section and footer section which can be created as 4 separate tables spaced apart vertically.
- The Header and footer should rarely change, the footer should indicate when each page was last updated.
- The content section is where most changes should occur.
- To speed up navigation from each page a simple common menu bar should be included under each page header. It should only contain the most frequently used links and a home page link.
- Infrequently used links should only be shown on the home page or on a relevant content page.
- No pop up menus, windows or advertisements. No animated graphics. Standard text box Google content related advertisement is allowed.
- A sitemap should be included, this will be one page where the content lists every other page and its links other than the standard menu bar links, which should be shown for the home page only.
- The sitemap should be one of the first pages created from the template and should act as a simple database to list all other pages on the web site. It can be used to sketch the basic layout of the site by subject / title and the actual pages themselves and their content created later.

7.3. Stage – Detailed Specification – Astrolab Application

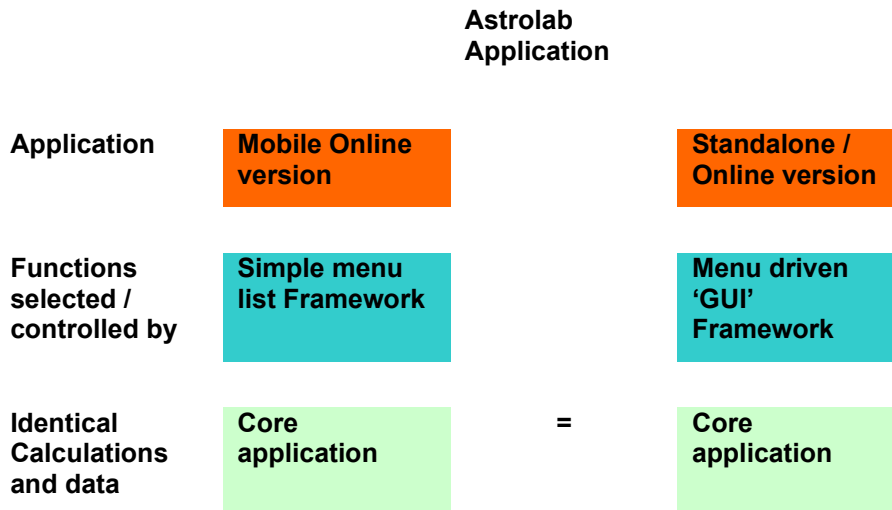
Item	In Version	Date last amended
Items Independent of Application Functionality		
		12/05/2008
Must be platform independent - so automatic support under MS Windows, Mac OSX and Linux		
Core is based on HTML standard and meets W3C standard		
Must meet accessibility standards		
Can be run from a Web server or standalone		
Standalone installation must be a simple process – drag and drop application folder		
Uninstall must be a simple process – delete application folder		
Programming, code must be understandable, well documented and platform independent – (Javascript only at present)		
All new releases to be cumulative		
Meaningful version identification to be used in all application versions and supporting website conquestofspace.org		
Application Interface:		12/05/2008
Web Browser		
Possible W3C compliant		
Simple frames to avoid duplication on individual pages		
Content:		08/01/2009
Initial development will be for PDA with a desktop / standalone / online version derived from this	0.8	
Third party resources used as part of the app:		08/01/2009
Some public domain sets of data on stars, YBSC and nearest and brightest stars	0.89 Cinfo009.html	

Facilities:		12/05/2008
There will be a download facility for a packaged version of the main application under the supporting web site		

8. Phase - Analysis and Design

8.1. Astrolab Application Simple Design Specification

The principle is that the core of the application will be identical in both the PDA and online / standalone application. How options are selected is controlled by a simple menu system in the PDA version, and by surrounding with a simple html 'frame' with menu options in the 'full' online / standalone version.



Changes made in the core will be made in the mobile version and simply copied into the standalone version. There will be just enough code in the core to remove links back to the supporting framework if not applicable to that version, or remove graphics not suitable for the PDA version.

All core applications common to all versions have file names starting with 'C'. Application framework files are specific to each version and have file names starting with 'F'.

Application 'windows', menus and calculations will be driven by HTML and Javascript

8.2. AstroEquations.xls

Equations, constants and values used in the Astrolab application. Spreadsheet AstroEquations.xls contains a list and 'test harness examples' of equations to be used in Astrolab. It also lists values for constants and measurements used in Astrophysics, using National Physical Laboratory and CODATA recommended values.

9. Phase – Implementation

The whole project is based around HTML pages and Javascript – all text files, designed using various HTML tools. No programs need to be compiled in current versions. Zipped folders are produced for deployment on platforms other than the development machines.

10. Phase - Deployment

To install website files they can simply be run from any folder they are stored in, or for Live releases they are simply uploaded to the web server.

The application files are also self contained, the zipped application is simply downloaded from the Sourceforge.net project page and copied and unzipped to the desired platform and the application started by loading the main.html page in a web browser.

11. Phase – Testing

Final releases will be tested and by the developer(s) and users. Any bugs will be recorded on the Sourceforge.net project page. Any bug fix will be recorded on Sourceforge.net list and may be included in the AstroChgLog.txt file with the next version made available for download on sourceforge.net

12. Phase – Evaluation

Any issues raised from testing or enhancement ideas should be logged on the Sourceforge.net project page and may be incorporated in a specification for the next iterative / incremental development and release, if a large development, or simply listed in AstroChgLog.txt file if changes are included with the next version made available for download on sourceforge.net.

13. Appendix A - JavaScript Style Guide

The following sections are still under revision.

Their main function is to list recognised standards used in JavaScript code for this project, which in turn should make the code easy to read and understand how it works. The lessons learned can then be used to write further JavaScript code in this or any other project or even move on to code in Java itself.

Sections cover:

- Code Style
- Comments
- Commenting Scripts in JavaScript
- Function Declarations
- Indentation
- JavaScript Files
- Names
- Quotes
- Statements
- Symbols
- Variable Declarations
- Whitespace

These sections are based on the [Sun](#) documents under [Code Conventions for the Java Programming Language](#), but it is modified for JavaScript, as this language is not exactly the same as the Java language, but we try to stick to the same naming conventions and formatting which are mainly designed to make the code readable and easier to maintain.

When storing files on the development platform all directory names are in interCaps convention. Web pages in the application follow Java interCaps function name convention and may have a lower case prefix of 'framework or 'core. Core functions are intended to be independent of the framework and can be run under mobile or standalone versions of the application. These are the parts of the application that could be ported to another language / platform.

Supporting documents follow the Java ClassName convention.

There is no real connection with Java, this naming is simply chosen for convenience and consistency.

13.1.1.Code Style

Always put else on its own line, as shown above.

Don't use else after return, i.e.

```
if (x < y)
    return -1;
if (x > y)
    return 1;
return 0;
```

Both `i++` and `++i` are acceptable.

Name inline functions, this makes it easier to debug them. Just assigning a function to a property doesn't name the function, you should do this:

```
var offlineObserver = {
```

```
observe: function offlineObserve(aSubject, aTopic, aState)
{
  if (aTopic == "network:offline-status-changed")
    setOfflineUI(aState == "offline");
}
}
```

13.1.2. Comments

Be generous with comments. It is useful to leave information that may be read after by people (possibly yourself) who will need to understand what you have done. The comments should be well-written and clear, just like the code they are annotating.

It is important that comments be kept up-to-date. Erroneous comments can make programs even harder to read and understand.

Make comments meaningful. Focus on what is not immediately visible. Don't waste the reader's time with stuff like

```
i = 0; // Set i to zero.
```

Generally use line comments. Save block comments (/* */) for formal documentation, comments over more than one line, and for commenting out.

13.1.3. Commenting Scripts in JavaScript

For older browsers which may not support JavaScript it is possible to hide the script as follows - the JavaScript engine allows the string "`<!--`" to occur at the start of a `<script>` element, and ignores further characters until the end of the line. JavaScript interprets `"/"` as starting a comment extending to the end of the current line. This is needed to hide the string `-->` from the JavaScript parser in older browsers at the end of a script before `</script>`. We use this standard to enclose all code within JavaScript and style tags, example below:

```
<script type="text/javascript">
<!-- to hide script contents from old browsers
function square(i){
  document.write("Value of I is", i , "<BR>")
  return i * i
}
// end hiding contents from old browsers -->
</script>
```

As applications using JavaScript require a browser that does support it then these comments may be irrelevant but we are leaving them in all code for now, as some code may be in the `<body>` of a html page as well as the `<head>`.

13.1.4. Form Fields

ID vs. Name in Form Fields

When creating a form and form field, you should give them both an id and a name and make them the same, e.g.

```
<form id="form1" name="form1">
<input type="text" id="form1Field1" name="form1Field1" />
```

```
<input type="text" id="form1Field2" name="form1Field2" />
</form>
```

This is because browsers use the name field when they submit to a server, but when working with JavaScript it's best to use id (and the associated `document.getElementById()`) to identify elements. You could use something like `document.getElementsByName()`, but it is a lot harder to work with, because names aren't guaranteed to be unique and thus that function returns a collection of elements, not just one.

You should also always name your name and id the same thing (and not reuse that identifier anywhere else), because of IE's poor implementation of `getElementById()` which can sometimes return an element whose "name" matches the "id" you asked for.

Always use both, make them the same string, and make that string unique in the document.

13.1.5.Function Declarations

All functions should be declared before they are used. Inner functions should follow the `var` statement. This helps make it clear what variables are included in its scope.

There should be no space between the name of a function and the `(` (left parenthesis) of its parameter list. There should be one space between the `)` (right parenthesis) and the `{` (left curly brace) that begins the statement body. The body itself is indented four spaces. The `}` (right curly brace) is aligned with the line containing the beginning of the declaration of the function.

```
function outer(c, d) {
    var e = c * d;

    function inner(a, b) {
        return (e * a) + b;
    }

    return inner(0, 1);
}
```

This convention works well with JavaScript because in JavaScript, functions and object literals can be placed anywhere that an expression is allowed. It provides the best readability with inline functions and complex structures.

```
function getElementsByClassName(className) {
    var results = [];
    walkTheDOM(document.body, function (node) {
        var a; // array of class names
        var c = node.className; // the node's classname
        var i; // loop counter

        // If the node has a class name, then split it into a list of simple
        // names.
```

```
// If any of them match the requested name, then append the node to
the set of results.
```

```
    if (c) {
        a = c.split(' ');
        for (i = 0; i < a.length; i += 1) {
            if (a[i] === className) {
                results.push(node);
                break;
            }
        }
    }
});
return results;
}
```

If a function literal is anonymous, there should be one space between the word `function` and the `(` (left parenthesis). If the space is omitted, then it can appear that the function's name is `function`, which is an incorrect reading.

```
div.onclick = function (e) {
    return false;
};

that = {
    method: function () {
        return this.datum;
    },
    datum: 0
};
```

Use of global functions should be minimized.

When a function is to be invoked immediately, the entire invocation expression should be wrapped in parens so that it is clear that the value being produced is the result of the function and not the function itself.

```
var collection = (function () {
    var keys = [], values = [];

    return {
        get: function (key) {
            var at = keys.indexOf(key);
```

```
        if (at >= 0) {
            return value[at];
        }
    },
    set: function (key, value) {
        var at = keys.indexOf(key);
        if (at < 0) {
            at = keys.length;
        }
        keys[at] = key;
        value[at] = value;
    },
    remove: function (key) {
        var at = keys.indexOf(key);
        if (at >= 0) {
            keys.splice(at, 1);
            value.splice(at, 1);
        }
    }
};
}());
```

13.1.6.Indentation

The basic unit of indentation is four spaces.

Any further indents required are in blocks of four spaces.

Tabs are not to be used at all because there still is not a standard for the placement of tabstops. The use of spaces can produce a larger filesize, but the size is not significant if the application is split into simple pages. Line Length

Avoid lines longer than 80 characters. When a statement will not fit on a single line, it may be necessary to break it. Place the break after an operator, ideally after a comma. A break after an operator decreases the likelihood that a copy-paste error will be masked by semicolon insertion. The next line should be indented 8 spaces.

A blank line is left between sections in JavaScript and HTML code.

```
var result = prompt(aMessage,
                    aInitialValue,
                    aCaption);

var IOService = Components.classes["@mozilla.org/network/io-
service;1"]
```

```
.getService(Components.interfaces.nsIIOService);
```

13.1.7. JavaScript Files

JavaScript programs should be stored in and delivered as `.js` files.

JavaScript code should not be embedded in HTML files unless the code is specific to a single session, otherwise the code in HTML adds significantly to pageweight.

13.1.8. Names

Names should be formed from the 26 upper and lower case letters (`A .. Z`, `a .. z`), the 10 digits (`0 .. 9`), and `_` (underbar). Avoid use of international characters because they may not read well or be understood everywhere. Do not use `$` (dollar sign) or `\` (backslash) in names.

Do not use `_` (underbar) as the first character of a name. It is sometimes used to indicate privacy, but it does not actually provide privacy. If privacy is important, use the forms that provide [private members](#). Avoid conventions that demonstrate a lack of competence.

Most variables and functions should start with a lower case letter.

Constructor functions which must be used with the [new prefix](#) should start with a capital letter. JavaScript issues neither a compile-time warning nor a run-time warning if a required `new` is omitted. Bad things can happen if `new` is not used, so the capitalization convention is the only defense we have.

- Use interCaps for names and enumeration values;
- JavaScript does not have macros or constants, however consistency is important, so constants are all capitals with the `"_"` between words as in Java, e.g. `UPPER_CASE`.
- Convenience constants for interface names should be prefixed with `nsI`, e.g.

```
const nsISupports = Components.interfaces.nsISupports;  
const nsIWBN =  
Components.interfaces.nsIWebBrowserNavigation;
```
- Enumeration values should be prefixed with the letter `k`, e.g.

```
const  
kDisplayModeNormal = 0;
```
- Global variables should be prefixed with the letter `g`, e.g.

```
var  
gFormatToolbar;
```
- Arguments (parameter names) should be prefixed with the letter `a`.
- Event handler functions should be prefixed with the word `on`, in particular try to use the names `onLoad`, `onDialogAccept`, `onDialogCancel` etc. where this is unambiguous.
- Function names, local variables and object members have no prefix.
- Try to declare local variables as near to their use as possible; try to initialize every variable.

13.1.9.Quotes

Always Quote Attribute Values

For HTML Attribute values should always be enclosed in quotes.

Double style quotes are the most common, single style quotes are also allowed, but we will always try to use double.

In JavaScript we will always use double quotes around strings, but in some rare situations, like when the attribute value itself contains quotes, it is necessary to use single quotes:

```
var name = 'John "ShotGun" Nelson'
```

OR

```
document.write('<a title="Main Page" href="mmain.html" >Main  
Page</a> &gt; Drake Equation');
```

13.1.10. Statements

13.1.11. Simple Statements

Each line should contain at most one statement. Put a ; (semicolon) at the end of every simple statement. Note that an assignment statement which is assigning a function literal or object literal is still an assignment statement and must end with a semicolon.

JavaScript allows any expression to be used as a statement. This can mask some errors, particularly in the presence of semicolon insertion. The only expressions that should be used as statements are assignments and invocations.

13.1.12. Compound Statements

Compound statements are statements that contain lists of statements enclosed in { } (curly braces).

- The enclosed statements should be indented four more spaces.
- The { (left curly brace) should be at the end of the line that begins the compound statement.
- The } (right curly brace) should begin a line and be indented to align with the beginning of the line containing the matching { (left curly brace).
- Braces should be used around all statements, even single statements, when they are part of a control structure, such as an `if` or `for` statement. This makes it easier to add statements without accidentally introducing bugs.

13.1.13. Labels

Statement labels are optional. Only these statements should be labeled: `while`, `do`, `for`, `switch`.

return Statement

A `return` statement with a value should not use () (parentheses) around the value. The return value expression must start on the same line as the `return` keyword in order to avoid semicolon insertion.

if Statement

The `if` class of statements should have the following form:

```
if (condition) {
    statements
}

if (condition) {
    statements
} else {
    statements
}

if (condition) {
    statements
} else if (condition) {
    statements
} else {
    statements
}
```

for Statement

A `for` class of statements should have the following form:

```
for (initialization; condition; update) {
    statements
}

for (variable in object) {
    if (filter) {
        statements
    }
}
```

The first form should be used with arrays and with loops of a predeterminable number of iterations.

The second form should be used with objects. Be aware that members that are added to the prototype of the *object* will be included in the enumeration. It is wise to program defensively by using the `hasOwnProperty` method to distinguish the true members of the *object*:

```
for (variable in object) {
    if (object.hasOwnProperty(variable)) {
        statements
    }
}
```

while Statement

A `while` statement should have the following form:

```
while (condition) {
    statements
}
```

do Statement

A `do` statement should have the following form:

```
do {  
    statements  
} while (condition);
```

Unlike the other compound statements, the `do` statement always ends with a `;` (semicolon).

switch Statement

A `switch` statement should have the following form:

```
switch (expression) {  
    case expression:  
        statements  
    default:  
        statements  
}
```

Each `case` is aligned with the `switch`. This avoids over-indentation.

Each group of *statements* (except the `default`) should end with `break`, `return`, or `throw`. Do not fall through.

try Statement

The `try` class of statements should have the following form:

```
try {  
    statements  
} catch (variable) {  
    statements  
}  
  
try {  
    statements  
} catch (variable) {  
    statements  
} finally {  
    statements  
}
```

continue Statement

Avoid use of the `continue` statement. It tends to obscure the control flow of the function.

with Statement

The `with` statement should not be used.

13.1.14. Symbols

- Spaces around braces used for in-line functions or objects, except before commas or semicolons, e.g.

```
function valueObject(aValue) { return { value: aValue }; };
```

- Otherwise function braces must always be on their own line, i.e.

```
function toOpenWindow(aWindow)  
{
```

```
aWindow.document.commandDispatcher.focusedWindow.focus();
}
```

- Otherwise spaces are not necessary inside brackets e.g. parameter lists, array subscripts. This includes wrapping an in-line JavaScript object in parentheses, or the `for (;;)` construct - the space normally required after the first semicolon is inhibited by the second semicolon, the space after the second semicolon is inhibited by the close parenthesis.

- Prefer double quotes, except in in-line event handlers or when quoting double quotes.

- Braces are not indented relative to their parent statement. Stick to the style used in existing files, but when creating new files you may choose your favourite of the following acceptable constructs:

```
if (dLmgrWindow)
    dLmgrWindow.focus();
else
    dLmgr.open(window, null);

if (dLmgrWindow) {
    dLmgrWindow.focus();
} else {
    dLmgr.open(window, null);
}

if (dLmgrWindow) {
    dLmgrWindow.focus();
}
else {
    dLmgr.open(window, null);
}

if (dLmgrWindow)
{
    dLmgrWindow.focus();
}
else
{
    dLmgr.open(window, null);
}
```

- Use `\uXXXX` unicode constants for non-ASCII characters. The character set for XUL, DTD, script and properties files is UTF-8 which is not easily readable.

13.1.15. Variable Declarations

All variables should be declared before used. JavaScript does not require this, but doing so makes the program easier to read and makes it easier to detect undeclared variables that may become implied globals. Implied global variables should never be used.

- x) JavaScript variables must start with a letter or underscore "_" and follow the interCaps convention
- xi) JavaScript is case sensitive.

The `var` statements should be the first statements in the function body.

It is preferred that each variable be given its own line and comment. They should be listed in alphabetical order.

```
var currentEntry; // currently selected table entry
var level;        // indentation level
var size;         // size of table
```

JavaScript does not have block scope, so defining variables in blocks can confuse programmers who are experienced with other C family languages. Define all variables at the top of the function.

Use of global variables should be minimized.

13.1.16. Variable Scope and Lifetime

The scope is the region of the program for which the variable is declared. Variables may be either Global, or Local. Local variables are available only within the section of code in which they were defined. Changes to Local variables are not reflected 'outside' their area of definition. When you exit this area, the variable is destroyed. Global variables are available to other JavaScript code. Generally, variables declared as "var" are Global variables. Variables declared inside a function as "var" are local to the function. Variables defined inside a function without the "var" are Global variables. The lifetime of Global variables starts when they are declared, and ends when the page is closed.

```
<script type="text/javascript">
var altitude = 5; //GLOBAL

function square( ) {
    base = 17; //GLOBAL
    sqr = 0.5*(base + altitude);
    return sqr;
}

function perimeter() {
    var side = 7.5; //LOCAL
    prm = 2*side + base;
    return prm;
}
</script>
```

13.1.17. Whitespace

Blank lines improve readability by setting off sections of code that are logically related.

- Lines should not contain trailing spaces, even after binary operators, commas or semicolons.

- A keyword followed by ((left parenthesis) should be separated by a space.

```
while (true) {
```

- A blank space should not be used between a function value and its ((left parenthesis). This helps to distinguish between keywords and function invocations.
- All binary operators except . (period) and ((left parenthesis) and [(left bracket) should be separated from their operands by a space.
- No space should separate a unary operator and its operand except when the operator is a word such as `typeof`.
- Separate binary operators with spaces.
- Spaces after commas and semicolons, but not before.
- Each ; (semicolon) in the control part of a `for` statement should be followed with a space.
- Spaces after keywords, e.g. `if (x > 0)`.
- One (or two) blank lines between block definitions. Also consider breaking up large code blocks with blank lines.
- End the file with a newline. (This applies mainly to emacs users.)

14. Appendix B - Recommended Development Tools

Recommended free and open source applications and their current versions used under OS X to develop this project. The development is now based on an iMac / OSX platform, only applications we have actually tested make it to this list, which is subject to change. Some testing may be performed under Windows XP running under VirtualBox (Sun Microsystems).

Application	Windows	OS X	Last Checked
Adobe Reader	V 9.3.1	V 9.3.0	23/02/10
ADSL Test Utility	-		
Chrome http://googlechromereleases.blogspot.com/	V 4.0.249.89	-	23/02/10
FileZilla	V 3.3.2	V 3.3.1	23/02/10
Firefox	V 3.6	V 3.6	23/02/10
GIMP		V 2.6.6	23/02/10
Learn HTML @ w3schools	-		-
NeoOffice	-	3.0.2	23/02/10
Safari	V 4.0.4 (531.21.10)	V 4.0.4 (5531.21.10)	23/02/10
Sourceforge.net	-		-
W3C Markup Validator	V 1.2		25/02/11

15. Original Specification for Astrolab

Items Independent of Application Functionality
Must be platform independent - so automatic support under MS Windows, Mac OSX and Linux
Core is based on HTML standard and meets W3C standard
Must meet accessibility standards
Can be run from a Web server or standalone
Standalone installation must be a simple process – drag and drop application folder
Uninstall must be a simple process – delete application folder
Programming, code must be understandable, well documented and platform independent – (Javascript only at present)
All new releases to be cumulative
Meaningful version identification to be used in all application versions and supporting website conquestofspace.org
Application Interface:
Web Browser
Possible W3C compliant
Simple frames to avoid duplication on individual pages
Content:
Initial development will be for PDA with a desktop / standalone / online version derived from this
Astrophysical Constant List
Astrophysics calculator
Convert Distances
Temperature conversions
Equations - Drake Equation
Glossary of common terms used in Astrophysics
Mathematical Constants
Periodic Table – List only
Physical Constants
Solar system data

Units of Measurement
Equations - Sphere volume and area calculation
Third party resources used as part of the app:
None at present
Facilities:
There will be a download facility for a packaged version of the main application under the supporting web site