CCS Visualization: Java Applet

Documentation of Progress

Written by: Michael Mascaro

Edited by: Sean Li

Summer 2011

**Table of Contents**

1. Usages and Goals: CCS Visualization Introduction…………………………………page 3
2. JSatTrak/World Wind: Introduction………………………………………………....page 4
3. Summer 2010: JSatTrak…………………………………………………………......page 5
4. Summer 2010: JSatTrak Reduced Functionality Prototype…………………….…...page 5
5. Summer 2010: BeanShell Scripting………………………………………………....page 6
6. Summer 2010: Total Progress……………………………………………………….page 7
7. Summer 2011: Goals………………………………………………………………...page 7
8. CCS Visualization: WebStart Introduction………………………………………….page 8
9. WebStart: Code Modifications……………………………………………………....page 9
10. WebStart: Capabilities……………………………………………………………….page 9
11. WebStart: Known Issues………………………………………………………..….page 10
12. WebStart: Progress Report…………………………………………………………page 11
13. CCS Visualization: Applet Introduction…………………………………………...page 11
14. Applet: Coding Process………………………………………………………..…...page 12
15. Applet: Code Description……………………………………………………..…....page 13
16. Applet: Library Changes...........................................................................................page 15
17. Applet: Current Capabilities………………………………………………………..page 17
18. Applet: Known Issues...…...…………………………………………………..…...page 19
19. Applet: Progress Report…………………………………………………..………..page 19
20. Applet: Future Plans……………………………………………………..…………page 20
21. Applet: Integration Plans/Requirements……...………………………………..…..page 21
22. Comparison: Advantages/Disadvantages..................................................................page 22
23. Conclusion.................................................................................................................page 23
24. Appendix...................................................................................................................page 24

**Usages and Goals: CCS Visualization Introduction**

The CCS website provides satellite information and data to its users as they require. Ideally, one of the tools available to users should be a visualization tool capable of displaying satellites in orbit and their current, past, and future locations. The current system in use by the CCS website involves a user requesting a specific visualization, which is generated in a movie format by FreeFlyer. This movie is then transferred over the internet to the user’s computer, where the visualization can be watched. Unfortunately, this is a slow process that provides a rigid, non-interactive result. It is the primary purpose of this project to develop a visualization alternative that provides a quicker, more interactive visualization. Part of the problem with using a visualization tool is the demands on the system and browser to display it. Advanced visualization programs can display high resolution images and smooth interactive features, but may not be functional for a user using an older internet browser or computer. An embeddable program that can be displayed within the CCS web pages is ideal; however other solutions were researched as well. Java applets were found to be the most useful and common programs for internet visualization, therefore this project is using Java as the language for visualization in order to ensure that it is available to the maximum possible number of users.

*Please note: The sections of this documentation pertaining to work done during the summer of 2010 are excerpts from the more detailed documentation done during that year. Please reference the documentation from that year,* ***Using JSatTrak for Visualization*** *and* ***Reduced Functionality Prototype Documentation*** *for more detailed information.*

**JSatTrak/World Wind Introduction**

During the primary research for the project, two programs were discovered that provided the beginning of the visualization tool. JSatTrak and NASA World Wind are both written in java and provide the foundation for a visualization tool suitable for the CCS website.

JSatTrak is an open source satellite tracking program. The application is created through NASA’s World Wind software and can calculate and display satellite orbits at any time in the past or future. It contains orbital elements for over 3000 satellites, has over 800 built in ground stations with customization available. It has the capability to display both a 3D globe and 2D ground tracking display with fully interactive features such as zoom, camera angle, and more. It allows for you to load custom 3D models (.3ds files) and includes a model-centered view mode. Available for download from the JSatTrak website (<http://www.gano.name/shawn/JSatTrak/>) is a web-start version, the entire downloadable program with an .exe and .jar file, and the source code.

NASA World Wind is a virtual globe developed in Java for use as a desktop application, web-start distributed application, or a java applet. This versatility made it ideal for our project, and it is also used as the base visualization for JSatTrak. World Wind comes with easy-to-use interactive features, and is available as an open-source download from the NASA World Wind website (<http://worldwind.arc.nasa.gov/java/>). This includes examples for making a World Wind desktop application, web-start application, and java applet.

**Summer 2010: JSatTrak**

*(Excerpt taken from* ***Using JSatTrak for Visualization****)*

JSatTrak does not have the capabilities to compete with an advanced program such as FreeFlyer, but it could have a role in simple, effective distributed visualization for satellite applications. Since JSatTrak has the capability to read satellite data from ephemeris files, and displays 3D visualizations with globes from NASA WorldWind, JSatTrak can be used as an effective visualization tool. Also, because JSatTrak is developed and written in Java it can be distributed over the web using Java web-start.

The biggest advantage to using JSatTrak as a visualization tool comes from the built-in BeanShell scripting capabilities. This enables JSatTrak to display a desired scenario on start-up, without a user inputting commands. Also, since scripts are written in Java and can be dynamic, a single script can create multiple different scenarios. This means that when distributed over the web for visualization purposes, a script can be distributed as well with the desired visualization, so the user opens the program and the visualization is already loaded.

**Summer 2010: JSatTrak Reduced Functionality Prototype**

*(Excerpt from* ***Reduced Functionality Prototype Documentation****)*

Using JSatTrak for visualization purposes in the CCS project is an effective tool, but JSatTrak contains many features that are unnecessary for the visualization goals required by CCS. In order to create a more user-friendly, embeddable version of JSatTrak that serves strictly as a viewer tool required a reduction of functionality from the original JSatTrak program. This documentation covers the development of the JSatTrak Reduced Functionality Prototype (RFP), which was developed to provide a prototype of a “view-only” mode of JSatTrak. It will cover the development of the RFP, as well as both the removed and current capabilities of the prototype. Also included within this documentation are the exact changes made to the source code and to the graphical user interface to create the RFP. Development has lasted for four days to the moment this documentation was written, so this documentation will also cover future goals for further development of the RFP. *(This is for 2010, not the current time)*

The Reduced Functionality Prototype is developed from Shawn Gano’s JSatTrak. The RFP was created by editing and modifying his code. The changes described in “Using JSatTrak for Visualization” were applied to the original source code, and then further modifications were made to remove features and create the RFP. Primary development of the RFP during the summer of 2010 was by Michael Mascaro, with support from Stefan Novak of a.i. solutions. Development of both the original JSatTrak and the RFP are located on github ([www.github.com](http://www.github.com)) and can be found under <http://github.com/mdmzero0/JSatTrak>. All code changes and work was done in NetBeans and tracked using Git.

**Summer 2010: BeanShell Scripting**

*(Exceprt from* ***Using JSatTrak for Visualization****)*

BeanShell is used in JSatTrak for two things, start-up mission creation, and plug-in capabilities. When JSatTrak first opens, it searches for a BeanShell script called “startup.bsh” within the directory that contains the .exe or .jar file for JSatTrak. This script is run immediately when the program is started. Also, JSatTrak is programmed to read BeanShell scripts located within the “Plugins” folder of the program files. These scripts are located under the “Plugins” window in JSatTrak. These scripts can be reloaded and opened once JSatTrak is already running and there is no file name requirement for these scripts as long as they have .bsh extensions.

**Summer 2010: Total Progress**

*(Except from* ***Using JSatTrak for Visualization****)*

Using JSatTrak as a visualization tool is not a far-fetched plan, but a feasible and easily implemented plan that would allow a simple, accessible way to view 3D visualizations of satellites in orbit to accompany mission analysis from CCS. The use of BeanShell scripts creates a simple yet powerful tool to display the different visualizations a user may require. The flexibility of being able to combine a single script with a reduced functionality version of JSatTrak opens up the possibility of an effective, interactive applet displaying visualizations for CCS users within moments of their analysis requests. Therefore, integrating JSatTrak into CCS would be an advantageous solution to a search for an interactive 3D visualization for CCS.

**Summer 2011: Goals**

It is the goal of this project to provide a working visualization tool for use in the CCS website before the summer’s end. Two main products are to be developed during this process; a web-start version of JSatTrak based off the Reduced Functionality Prototype developed last summer, and an embeddable applet based off of a combination of NASA World Wind and JSatTrak code. The web-start version has other possible applications, including one as a replacement for STKViewer for FDS purposes, but the end result of this project will be a fully embedded Java applet using code from JSatTrak to display satellites on NASA World Wind. As the code is further developed, more functionality from JSatTrak can be incorporated without much difficulty. The minimum requirements for our project to be deemed “completed” would be as follows: having an embeddable virtual globe that displays multiple satellites in orbit, can “play” the visualization (show the satellites orbiting around the globe as it rotates, etc), and performs the previous actions by reading the satellite information off a web server location.

**CCS Visualization: WebStart Introduction**

The web-start application designed for CCS would serve as a temporary solution to the visualization problem while other, more ideal solutions are being developed (ie: embeddable visualizations). A Java web-start program exists as a go-between for desktop applications and embedded applications. When using a web-start application, the user clicks on a hyperlink on the webpage, which serves as a download for the .jnlp file, a set of instructions for the web-start program. Java reads this downloaded file, and follows the instructions to run a program without installing any components. Therefore it is more accessible than a desktop application, as no installation is required and only one file is downloaded to the user’s computer. For CCS purposes, a web-start application is being developed with an included script to read parameters and ephemeris data from a remote file location. The web-start application would then load those instructions into the application for visualization.

**WebStart: Code Modifications**

*The following code modifications are additional modifications to the ones described in the development of the Reduced Functionality Prototype from 2010. For more information on those modifications, please reference the aforementioned documentation.*

The code from JSatTrak has been modified in order to reduce functionality as well as fix bugs in the program. One of the initial bugs discovered caused the 2D view to zoom in the opposite direction that the zoom button was pressed. This was fixed quite simply in line 890 of the file “J2Dearthpanel.java”. Another vital modification made involved the capability to read ephemeris files from a remote server location using a URL instead of a filepath. This capability was added in the file “StkEphemerisReader.java” and be found on line 102. In addition to this capability, the java file that handles user inputs was modified to read off a remote server using a URL (“onlineInput.java”). These modifications make the web-start application independent of files on the local machine.

**WebStart: Capabilities**

The main reason behind using the web-start application is the current capabilities provided by the application. These include:

* Visualization of 2D groundtracks as well as 3D globes
* Opening of both xml and zip formatted scenarios
* Saving scenarios
* Reading both a start-up script and reading scripts as plugins
* Model-centered and Earth-centered view modes
* Fixed and inertial 3D Earth-centered views
* Interactive 3D and playback features
* Screenshots and movies created from scenarios
* Real time and non-real time display modes with variable step sizes

These capabilities, combined with the accessability of the web-start application (any computer with Java installed can run it), make the web-start application a versatile, ready-to-use visualization tool.

**WebStart: Known Issues**

There are a few known issues currently unresolved with the web-start application, one of which is critical to the deployment of the program. The first issue is the lack of propagation past the ephemeris dates for the satellites. Currently, once a satellite reaches the end of the ephemeris file, it simply disappears from the program. Ideally, the satellite should continue to propagate after the ephemeris file ends, using the last data from the ephemeris file to predict the satellite location. Another issue occurs when switching from a full-screen to a smaller window for the 3D views. When switching out of full-screen, the view does not repaint correctly at first, waiting until the user mouses-over the display. Both of these issues, however, are not critical to the deployment of the web-start application.

The major issue directly affects the deployment of the web-start application. This has to do with including the necessary files to run the web-start application, causing it to be unable to load any 3D visualization. The application requires two native libraries additional to the standard libraries included on any computer, which should be loaded by the .jnlp file for use in the application. However, the file is currently failing to load these libraries, causing the web-start application to fail to meet the requirement of a 3D visualization tool. These required libraries have been used before in other web-start applications (including JSatTraks own web-start), however, so the problem can be resolved.

**WebStart: Progress Report** *(Summer 2011)*

The web-start application is ready for deployment on a server for testing, pending resolution of the required libraries error. Barring further errors being discovered, the web-start application will be capable of providing a 3D visualization, as well as its other previously listed capabilities, from the CCS website, using files and information located on the servers and requested by the user.

**CCS Visualization: Applet Introduction**

While distributing visualizations through a web-start application is an acceptable method, a more ideal method would involve using an embedded visualization tool in the webpage. Creating a Java applet, however, is somewhat more difficult than a web-start application. Java applets require the main class to be a subclass of the Java applet class, meaning that JSatTrak would have to be significantly modified to fulfill this requirement. The sheer size and complexity of the JSatTrak code would make this a time demanding and challenging process. In light of this, it was decided to approach the creation of the applet in a different way. An example applet that embeds NASA World Wind into a browser was used as a base code, and segments of JSatTrak’s code were added into the program to add the functionality of a satellite visualization tool. This method, while difficult in its own right, has proven to be a far more feasible strategy for developing an effective, embeddable visualization tool.

**Applet: Coding Process**

The initial process of modifying the World Wind applet into a useable visualization tool required the addition of many files and hundreds of lines of code. Code located in the main class of JSatTrak, as well as the J3DEarthPanel, was added to the main applet file. This code included the code for the 3D layers used by JSatTrak, and code for the addition of one satellite. Additional code was added as needed to resolve the dependencies caused by the addition of the code, until all the necessary code was added to display a satellite in the applet. A total of 39 additional files were added to the applet code.

While all the dependency errors were resolved with these additional files, there were still major problems. The World Wind applet code we had found was designed using a version of NASA World Wind that was newer than the version used for JSatTrak. Unfortunately, the two versions were not compatible, and this was causing errors throughout the code. In order to resolve these problems, we either had to change the JSatTrak or World Wind applet code to be compatible with a specific version of NASA World Wind, or we had to create our own library for use with our code. It was decided, due to the complexity of the methods and processes within both of the sets of code, that we would merge together the two versions of NASA World Wind to create a library compatible with the entire code.

The process of creating a library compatible with both versions of NASA World Wind required the source code for each version of the library. The latest version was readily available for download, but the previous versions were much more difficult to find. A Java decompiler was used to recreate the source code for the older version, but the decompiler failed to recover the source code for certain segments of the code. A process of trial and error, as well as in-depth research, eventually led to the discovery of the remaining source code, and the newly merged library was imported successfully into the applet, resolving the previous errors.

After all the syntax errors were resolved, the process switched to resolving run-time errors, which required only minor adjustments to be made to the code. Once these errors were resolved, all that remained was debugging and adding functionality to the applet, which was done through a process of trial and error.

**Applet: Code Description**

The primary code and processes for the applet are located in the **Main** package, in the file “WWJApplet.java”. This file creates the World Wind virtual globe, as well as adds the additional layers needed for the JSatTrak visualization. These additional layers are primarily the **eciLayer** (Earth Centered Inertial) and the **ecefLayer** (Earth Centered Earth Fixed). Contained within these two layers are the orbit models for the satellite display. When a satellite is rendered, it is rendered within these layers. An important thing to remember about the two layers is that they contradict one another- you cannot have a view be both inertial and fixed, so therefore one layer must always be **setEnabled(false)**. Failure to do so will cause two satellites to be rendered. The layers are set up within the **setUpLayers()** command.

Included after the addition of these layers is the code for adding a satellite using data read from an ephemeris file stored on a remote server location. The satellite is created using a hard-coded name and ephemeris location. The ephemeris is read into the applet using the StkEphemerisReader **readStkEphemeris** method. The time is retrieved from the ephemeris file time string and this is set to the current scenario time using **setTime(time)**. Setting the scenario time changes the position of the sun and the sun shading to their corresponding positions at that time. The method **propogate2JulDate** is another important method, which takes the ephemeris data and converts it into a format that can be rendered by the ECI and ECEF layers. Lastly, the most important thing in the code involving the addition of the satellite is the **satHash.put()** command. This command adds the satellite and its information into the satellite hashtable, which stores the information of every satellite in the applet. The methods that use the satellite information and render the satellites all read the satellites from the **satHash**, so unless you put the satellite into the **satHash**, it will not be displayed in the applet. The code for adding the satellites is found in the **inputSatellites()** method.

The next important method is the **updateTime** method. This method is used to update satellites and the 3D graphics as the time in the scenario changes. The **setupView** method modifies the view settings for either an earth-centered view or a model-centered view. The update method adjusts the lens flare layer based on the position of the sun. The code just described is all contained within the **Main** package in the “WWJApplet.java” file.

The **Bodies** package contains the java files for the sun and moon. Currently, the code to display the moon is not working, but the sun displays correctly and its position updates correctly within the applet. The Layers package contains all the layers that are added to World Wind to display the satellites. These include the **eciLayer** and **ecefLayer**, as well as the orbit model layers, sun position provider, and the view setup layer (“AutoClipBasicOrbitView.java”). The Satellite package intuitively contains the files need for the addition of satellites to the applet, namely “AbstractSatellite.java,” “CustomSatellite.java,” and “StkEphemerisReader.java”. In the Shapes package is “Cone.java” and “SphereObject.java.” The cone is used for displaying the satellite sensor footprint, which is currently disabled. The sphere object is more important however, whenever a satellite is not assigned a 3D model, it is represented by a sphere created in “SphereObject.java.”

The largest package in the applet is the **Utilities** package, where the majority of the computational code is located. Most of these files simply contain methods for solving the math involved in the orbits and calculations, but one of the most important codes contained in this package is “Time.java.” This file contains all the methods relating to the scenario time and its manipulation. The final package is the **View** package, which contains all the additional files used for viewing the satellites. Most of these files deal with the model centered view code, which is not currently in use.

**Applet: Library Changes**

One of the problems found in the development of the applet occurred as a result of JSatTrak’s age. JSatTrak, and consequently the code we were using from JSatTrak, was built using NASA World Wind version 0.5. The applet code, and the World Wind library we were using for the applet, used version 0.6 however. The two versions are very different from each other, with some methods not existing in the other version, and classes being moved into different packages and with different names. This causes considerable difficulty in coding the applet, and has been the source of the majority of errors and bugs within the applet. In order to counteract this, a process has been developed to combine various parts of the two libraries so that these errors are resolved.

In order to combine these libraries, the source code was downloaded for both version 0.5 and version 0.6. A java decompiler was also used to attempt to determine the correct methods and classes that needed to be added to the 0.6 version from the 0.5 version. Changes were made to the 0.6 version source code in NetBeans, then compiled and built into a new jar file for testing within the applet code. Through a process of trial and error, the syntax and runtime errors were resolved. Minor changes made to the library included adding methods for **setNearClipDistance** and **setFarClipDistance** in “BasicView.java” and adding **getOrbitViewModel** in “BasicOrbitView.java”.

The biggest changes to the library involved the addition of a package used to control the sun shading within the applet. Downloaded from an external source, this package is added to the examples folder in the World Wind library. Inside the sunlight package, the “RectangularNormalTessellator.java” class had to be modified to be compatible with the 0.6 library. Several additional override functions had to be added, and their code was derived from finding other override usages of the same function within the 0.6 version of the World Wind library. However, it was later found that the “RectangularNormalTessellator.java” class was causing an incompatibility error with Windows 7, and the class is no longer used within the code. The primary purpose of the class was for the sun shading effect, which has been removed as a result.

Another important change was made to the “BasicOrbitView.java” file, in the **computeFarClipDistance()** method. The return value has been changed to a multiple of ten times the value. This resolved the rendering issue where the satellite visualization clipping plane was causing the satellites to only render half an orbit.

Another change, albeit not as important, was made to the **starsLayer()** method. There was a problem with the stars rendering and if you zoomed too far out you could see a globe made of stars. To fix this, we added a \*10.0 to setRadius which affects the radius of the stars globe. An example of the code looks like this: starsLayer.setRadius(starsLayer.getRadius()\*10.0);

**Applet: Current Capabilities**

As of July, 2011 the applet has the capability to access a parameters file from an http URL that contains the information for satellite name, color, ephemeris location, and a scenario time string. The applet can read and parse this file into the corresponding commands and information to add each satellite requested into the scenario. Each satellite is displayed as a sphere and their orbits are plotted around the Earth in the color requested, and each satellite is labeled with its corresponding name given in the parameters. The scenario time string is read into the applet, and if the given time is in the correct format and within the times detailed in the ephemeris files, the applet time is set to the requested time.

Once the scenario is loaded, the user can use the play, pause, and reset buttons to control the visualization. The play and pause buttons work play and pause the scenario, as expected, and the reset button returns the scenario to the original time (typically the requested time in the parameters file, unless it was incorrect) and stops the scenario from playing. The user can change the time steps in increments using increase and decrease buttons, which raise and lower the time step to pre-programmed sizes. The user can also input a custom step size, which is accepted into the program if it is of the correct format (a double).

There are several view options available to the user, including the option to choose between an Earth-Centered Inertial (ECI) and an Earth-Centered Earth-Fixed view (ECEF). These options are both available for the 3D view, but the user can also select a 2D view that shows the groundtrack of the satellites. The user can also choose to view the orbit trace or to just view the satellites by themselves (default shows the orbit trace). These view options can be changed at any time, even while the scenario is running.

The final options available to the user are the real-time mode and the automatic update mode. The real-time mode sets the scenario time to the current system (computer) time, changes the step size and refresh rate so that the scenario steps one second every second, and plays the scenario. When the real-time mode is off, the scenario time returns to whatever time the scenario was previously on (not necessarily reset to the beginning of the scenario), and is reset to whatever the previous step size was, and the refresh rate for the scenario is set back to the default value (50 milliseconds). The automatic update mode, when selected, uses a timer to re-input the parameters file and essentially reload the scenario. This allows any changes to the satellite information or ephemeris to be changed without reloading the webpage, simply by changing the values in either the parameters or ephemeris files. When reloading the scenario, the user input time string is ignored as long as it had been input correctly originally, meaning that the scenario will not reset to the input time on each automatic update (approx. 6 seconds currently).

**Applet: Known Issues**

There are a few known issues or desired improvements that still remain for the applet itself. The first issue involves the place names layer (when set visible) and the satellite names. When there are labels involved in the scenario (place names or satellite names), they are visible through the Earth, causing a rather confusing visualization. In short, the Earth does not obstruct the view of the names, causing them to appear to be in places that they are not. We suspect this is a layering issue, but so far we have been unable to resolve it. As a result, the place names layer has been set to a default of false or not visible, and cannot be made visible unless the code is modified to true.

The next error is a more confusing problem, involving the animation of the scenario. When the scenario is animating, there is a noticeable pause in the scenario that occurs regularly every one and a half seconds (rough estimate). This pause remains no matter what step size is chosen, and if a faster refresh rate is used, the pause still remains at roughly the same intervals. This problem was not evident in JSatTrak, so we’re not sure what could be causing the issue or how to resolve it. As it is, the only time the pause is not noticeable is when using real-time mode.

**Applet: Progress Report**

The applet has been tested extensively for bugs within the functionality that has been previously described. It is currently distributed locally, but the files it reads (the parameters and ephemeris files) are all hosted over a local sever using a program called Mongoose. The applet has been tested and confirmed to work on Windows XP and Windows 7, as well as Internet Explorer 8, Internet Explorer 9, Google Chrome, and Mozilla Firefox 4. Testing has yet to be completed on an Apple operating system, Windows Vista, and the Safari web browser. Each aspect of the functionality described has been tested multiple times, and in conjunction with other aspects of the applet code, and all errors encountered have been resolved.

**Applet: Future Plans**

There are several options and opportunities for future expansion of the capability of the applet and improvements for the visualizations. Currently the applet can only read ephemeris files that are in STK format, which limits the usefulness of the program. In the future we’d like the applet to have the capability to read any ephemeris format, making it much more versatile. While this improvement would be the highest priority, it is also rather difficult to achieve due to the number of different ephemeris formats and the math inherently involved in reading orbit data and plotting it. Another idea for a future capability is to have a user be able to choose which satellites they want to visualize within the applet. Currently, the parameters file controls which satellites are visible: if the satellite is input into the parameters file, it is visible within the scenario at some point. However it could be advantageous to load many satellites, and then allow the user to select which satellites they wanted to visualize at any particular time.

Further future plans could include the addition of a model-centered view mode, where a 3D model of a satellite is loaded and used as the center of the visualization. Currently there are segments required for the model-centered view mode in the applet, but they are not implemented and the capability is not currently available. The model-centered view would be an option like the ECI and ECEF view options, but it would be a single model-centered view, with the option of selecting which satellite to view. Since a large majority of the model-centered code from JSatTrak is in the applet code, this should not be too much of a challenge to complete.

Other minor visual improvements could be made, such as the option to display a “tail” orbit trace, in which only a lagging segment of the orbit is shown behind the satellite, in order to cut down on visual “clutter” when viewing large numbers of satellites. This could be very difficult, however, because of the way that the orbit trace is created using arrays in the code. Previous attempts to create this option have created multiple errors.

**Applet: Integration Plans/Requirements**

In order to integrate the applet into the CCS website or any webpage, a few requirements have to be completed. First, the proper files have to be created to call the Java applet. These include Java’s .jnlp web-start launcher file, and an html file that serves as the webpage (see the appendix for examples of these codes). Once these two files are created, and the required applet Jar files are built and hosted on a server, the applet is ready to use. The one additional piece of information needed is the user inputs. Currently, the inputs are created into a parameters file that is also hosted on the internet, but has to follow a specific format in order to be read into the scenario. An example of this format is located in the appendix. For more specific instructions on the applet integration, please see the documentation entitled “User Guide for Visualization Applet.”

**Comparison: Advantages/Disadvantages**

When comparing the advantages and disadvantages of the web-start distributable visualization tool, and the applet visualization tool, there is one tool that is clearly superior to the other. The web-start has the advantage of being a distributable version of JSatTrak, meaning that it has nearly all the functionality of the original JSatTrak, including the ability to create multiple views at the same time, make movies, create screenshots, and change a variety of visualization options. However, these advantages are limited by the fact that the web-start visualization requires a download to the user’s computer, as well as extensive scripting to create a beanshell file capable of displaying the requested scenario. The larger size of the program also means that the user will experience larger load times, and possibly slower computer performance as well.

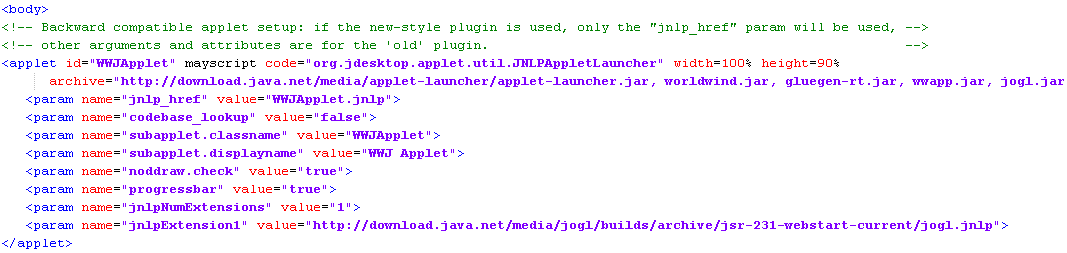
On the other hand, the applet visualization, while lacking in the ability to display multiple views at once, retains a large amount of the visualization options available to the web-start version. The applet also requires less scripting since it is hard-coded to read user inputs and ephemeris location (no beanshell scripting required). Since the applet is smaller and embedded in a webpage, there are no downloads required and the wait time is shorter. The biggest advantage to the applet is the ability for it to be displayed within the webpages of CCS, making it much more accessible. Clearly, using the applet as the preferred visualization tool is the best choice, and it has been developed as such.

**Conclusion**

The visualization tool for CCS is an important upgrade to the current process of using a video created by Free Flyer to display satellites in orbit. It has been developed from a Java satellite tracking program (JSatTrak), into a web-start distributed version with reduced functionality, into an embeddable applet built from NASA World Wind software. The applet solution that has been developed is more than capable of meeting and exceeding the current capabilities of the visualizations within CCS. The applet has been extensively tested, and the known issues have been documented, as well as areas for future progress and improvements. It is our hope that this program will be of great use for the CCS website and other future visualization endeavors.

**Appendix**

*Code for embedding an applet within html (Example)*



*Code for the .jnlp applet launcher (Example)*

