**Red Team**

**Stargazer**

**Automatic Telescope Control System**

**Configuration Management**

**Team Members:**

Rob Grmek

Robert Smith

**Instructor:**

**Youry Khmelevsky**

**Course:**

**COSC 471**

# ****Configuration Management****

The purpose of this configuration management document is to mention versioning control methods, naming conventions, coding conventions, and methods for system backup and recovery.

#### System Versions:

So far, there have been four releases of Stargazer. To see application code for any release of the web system, see: [http://github.com/RedTeamCOSC470/Stargazer](http://github.com/RedTeamCOSC470/StargazerReleases)

In the link, there is also a download section where different versions may be downloaded in zip or tar format. Releases have been tagged starting with *Release #5: Libraries and GUI Enhancements*.

Documentation and the telescope-controlling application code is also available in much the same way at: <http://github.com/RedTeamCOSC470/Documentation>

*Release #1: Authentication and Authorization*

The basic elements of the web application have been created. There is authentication and authorization mechanisms; user’s need to login to access any functionality of the web system and there is a hierarchy of users (either admins or regular users) each with separate privileges. Also, the basic scheduling and user management abilities are available.

*Release #2: Image Gallery, Presentation and Mobile Devices*

The web application has been improved. Apache is setup for our production web server and supports HTTPS access (although using a self-signed certificate). An image gallery is maintained per each schedule; a schedule can have many images associated with it – each with different sized resolution pictures such as a thumbnail version which can be enlarged in order to see the full resolution image. CSS has been added for mobile users so they may more easily access the web system and the CSS for PC users has also been improved. The basic groundwork has been laid for online help and about us information.

*Release #3: Refactoring*

This is the refactored version of release #2. Changes to the web application also have been made to input forms for scheduling tasks as well as to the CSS for mobile and PC users. Also, a test telescope-controlling application has been built to communicate with the telescope by writing to the serial port. Commands such as slewing the telescope North, South, West, and East have been tested and work.

*Release #4: Automated Scheduling and Telescope Positioning*

Job scheduling and interfacing with the telescope drivers was the priority here. The telescope-controlling application now allows for slewing the telescope to a specific object’s position given the proper coordinates and for parking the telescope, among other commands. Oracle 10g XE database is now being used (instead of SQLite which was not a very viable production database, anyways) so as to overcome issues with job scheduling. Triggers exist on the schedules table so when a user creates a schedule through the web application, the trigger will create a dbms\_schedule job which will call a batch file which will run the telescope-controlling application at the specified time. Also, some minor additions to the system include up-to-date online help and some minor refactoring.

#### Versioning Control:

All project files will be protected through the versioning control system ‘Git’.

All web application project files will be uploaded to the ‘Stargazer’ directory.

All documentation files will be uploaded to the ‘Documentation’ directory.

By using Git we avoid several problems such as team members simultaneously updating files as well as errors resulting in loss of data or system failure. In this case an older version would need to be retrieved and used.

Tutorials for using Git can also be found in Appendix C in the Developer’s Guide.

#### Github Account:

Login at: <http://github.com/>

Username: RedTeamCOSC470

Password: stargazer09

#### Web Application Repository Information:

Name: Stargazer

Public Clone URL: <git://github.com/RedTeamCOSC470/Stargazer.git>

My Clone URL: [git@github.com:RedTeamCOSC470/Stargazer.git](http://github.com/RedTeamCOSC470/git@github.com:RedTeamCOSC470/Stargazer.git)

#### Documentation Repository Information:

Name: Documentation

Public Clone URL: [git://github.com/RedTeamCOSC470/Documentation.git](git://github.com/RedTeamCOSC470/Stargazer.git)

My Clone URL: [git@github.com:RedTeamCOSC470/Documentation.git](mailto:git@github.com:RedTeamCOSC470/Documentation.git)

#### Archived Releases Repository Information:

Name: StargazerReleases

Public Clone URL: [git://github.com/RedTeamCOSC470/StargazerReleases.git](git://github.com/RedTeamCOSC470/Stargazer.git)

My Clone URL: [git@github.com:RedTeamCOSC470/StargazerReleases.git](mailto:git@github.com:RedTeamCOSC470/StargazerReleases.git)

#### Naming Conventions:

Each document that is created is named:

1. Prefix starting with ‘Stargazer’.
2. No spaces are used, underscores are used instead.
3. Only alphabetic characters and underscores are used.
4. Book title capitalization is used.

An example of a document name: Stargazer\_Vision.doc

#### Ruby Coding Standards:

The following Ruby code conventions are used are used for the web system. Here, some of the traditional Ruby conventions (such as from this website <http://pub.cozmixng.org/~the-rwiki/rw-cgi.rb?cmd=view;name=RubyCodingConvention>) are used. For example:

* Don’t use tab indenting, instead use 2 spaces.
* Line length: maximum 80 characters.
* No spaces after (, [ and before ], ).
* Use spaces around operators, after commas, colons and semicolons, around { and before }.
* Indent when as deep as case.
* Use def with parentheses when there are arguments.
* Use &&/|| for boolean expressions, and/or for control flow.
* Avoid multiline ?:, use if.
* Avoid return where not required.
* Use snake\_case for methods.
* Use CamelCase for classes and modules. (Keep acronyms like HTTP, RFC, XML uppercase.)
* Use SCREAMING\_SNAKE\_CASE for other constants.
* Avoid long methods.

#### C# Coding Standards:

The following C# code conventions are used for the telescope-controlling application.

When writing C# code the programmer should attempt to follow a set of rules to ensure the code is clear, readable and maintainable. To this end a set of coding standards is used. Please follow these if you make any changes to any C# code.

All methods or functions should include a comment explaining what they do. This should be in a multi line comment block located just inside the braces for the method. Even if it seems obvious from the code, it is easier to look at a comment to see what the code should be doing.

static void Main(string[] args)

{

/\*

\* The main part of the program! Does what it is told

\*/

//Your code here

}

Constant values should be specified in a variable close to the start of a function or method they are used in. If they are used in multiple functions they should be declared as static variables at the start of the class block. These should have a meaningful name that reflects what the constant represents. No one likes magic numbers.

static double aMethod(double Hour, double Minutre, double Second)

{

/\*

\* an example of constants

\*/

double MinutesInHour = 60;

double SecondsInHour = 3600;

return (MinutesInHour \* SecondsInHour);

}

Opening braces should be placed on a new line and should be indented to the same depth as the code they are part of (this is worded badly… see example). Code inside the braces should be indented one tab length further in then the braces. This keeps all the braces lined up and keeps the code easier to read.

namespace NameSpace

{

class Program

{

static double aMethod(double Hour, double Minutre, double Second)

{

//Your code here

}

}

}

Parameters passed into functions should be capitalized if they are a single word. If the parameter is more then one word the first word should be lower case and all other words are upper case. I have no idea why I do this but I do.

static void Main(double Value)

static void Main(string someOtherValue)

Method names should follow a similar standard as parameter names. Capitalized if the name is only one word and lower case first word and capitalized for every word after that if more then one word.

static void aLongMethodName()

static void Short()

All operators such as assignment, addition or equality operators should be surrounded by spaces if possible. This will keep the code clean and readable.

double aVariable = 60;

if (Args.Length == 2);

There may be more added later if I think of any.

#### Setting up Git and using Git:

This set up takes place after a Github account has been created and a repository has been created. This information is also available in Appendix C of the Developer’s Guide. Currently the information for the account and repository are:

#### Github Account:

Login at: <http://github.com/>

Username: RedTeamCOSC470

Password: stargazer09

#### Repository Information:

Name: Stargazer

Public Clone URL: <git://github.com/RedTeamCOSC470/Stargazer.git>

My Clone URL: [git@github.com:RedTeamCOSC470/Stargazer.git](http://github.com/RedTeamCOSC470/git@github.com:RedTeamCOSC470/Stargazer.git)

Information for setting up Git: (taken from Github)

#### Global setup:

See below for how to initially setup Git on a computer.

|  |
| --- |
| # First, download and install Git (see installation guide)  # Add configuration information:  git config --global user.name "Your Name"  git config --global user.email "Your Email"  # Then, add your public key (see below) |

#### Adding a public key:

A public key needs to be added to validate the computer as the owner of the repository so commits can be made.

|  |
| --- |
| # Generate a public key:  ssh-keygen  # Use the ‘cat’ command on the file that was created in  # directory such as:  cat ~/.ssh/id\_rsa.pub  # Then, add the public key to the Github account under:  # Account Settings -> SSH Public Keys |

#### Cloning the Repository:

Do this to recreate the directory structure with all project files.

Do this if the local Git repository has not already been made.

Can be used after the development environment is setup and freshly installed.

|  |
| --- |
| git clone [git@github.com:RedTeamCOSC470/Stargazer.git](mailto:git@github.com:RedTeamCOSC470/Stargazer.git) |

#### Other commands:

|  |
| --- |
| # create a new local repository:  git init    # add a file to the staging area:  # in other words, to setup file(s) before a local commit  git add [filename]  # to add all files use this:  git add .  # check status of the staging area files:  git status  # commit the staging area files to the local repository  # using the –m switch includes a message:  git commit –m “This is a commit message”  # show commits:  git log  # change username for only the local git repository:  git config user.name “[User Name]”  # adding a new remote destination called “origin”:  # a remote destination in this case is our  # Github repository  git remote add origin [git@github.com:RedTeamCOSC470/Stargazer.git](mailto:git@github.com:RedTeamCOSC470/Stargazer.git)  # list remote destinations:  git remote  # pushing the local committed files to the remote destination  # from the master branch “origin”:  # in other words, this is to commit the files to the  # Github repository.  git push origin master  # check if the local commit is not already pushed to  # the remote destination:  git log --pretty=oneline master...origin/master  # update the local Github origin master branch:  git fetch origin  # merge GitHub’s remote changes into the local master branch:  git pull origin master  # in the project root, create a file for Git to use  # to ignore certain files:  vim .gitinore  # then write the filename into the file  [filename]\*  :wq!  #then add the file  git add .gitignore  # see changes since files have last been stages:  # (file is added, but not committed)  git diff  # remove any changes in the working directory:  git checkout -- .  # unstage the file, but still have changes to the file:  git reset HEAD [filename]  # start from the last commit; remove everything:  git reset -hard    # create a branch:  git branch [branchname]  # list branches:  git branch  # checkout 1 commit back  git checkout HEAD^  git checkout HEAD~1  # 2 commit backs, etc  git checkout HEAD^^  git checkout HEAD~2  # go back to master  git checkout master  # use gitignore command to ignore some files  touch .gitignore  # add files you wish git to ignore (i.e. doc/api)  doc/api  # create a branch and checkout that branch  git checkout –b manage\_users  # or  git branch manage\_users  git checkout manage\_users    # merge the changes to master branch and delete old branch  git checkout master  git merge manage\_users  git branch –D manage\_users |