CCR

(Computer Controlled Railroad)



**Name**: Brice Hilgemann

**Professor**: Dr. Pankratz

**Class**: CSCI460 - Senior Capstone Experience

**Date**: April 28, 2010

Index

**General About Me**

* Introduction 3
* Resume 4
* Philosophy 5
* Career Plans 5

**CCR Project**

* Project Item Locations 6
* Project Definition 7
* Project Requirements 7
* Previous Versions 8
* Initial Strategy 9
* Client-Server Design 9
* Server Interface
  + Design Progression 10
  + Layout and main job 11
  + Structures and ownership 11
  + Server Diagram 12
* Client Interface
  + Design Progression 13
  + Layout and main job 14
  + Structures and ownership 14
  + Client Diagram 15
* Communication – The Protocol 16
* Structures Explained 17
* Issues Resolved 20

**Documentation**

* Hardware Documentation 21
* Quick User’s Guide 22
* Presentation Documentation 23
* CCR Code – Client Functions 32
* CCR Code – Server Functions 34

**Coding**

* CCR Code – Client 38
* CCR Code – Server 44

Introduction

This is Brice Hilgemann's Senior Capstone Project for the Computer Controlled Railroad.  
  
Although this project has gone through many different iterations in the past, hopefully this will be a new look that expands on other missing areas. This project focuses on multiple interfaces to interact with the railroad and allow anyone (even those without previous model railroad experience) to control train/track hardware. Below is an abstract of this project:

**Computer Controlled Railroad**  
  
Model railroads have always been a hobbyists delight with complexities and technical hardware that takes hours to fully understand and utilize.  Being able to move a train and control the railroad is the ultimate goal - but is this something that anyone can do?  This project looks at taking the complexities out of the way with easy to understand computer interfaces that can control railroad objects and prevent collisions.  Multiple users will be able to select a train to command, manipulate railroad hardware, and receive visual feedback on their individual train progress.

The following pages will give a brief introduction about the author followed by information regarding this project. Documentation is included throughout this document as well as the project website at <http://compsci.snc.edu/cs460_archive/2010/hilgbs/>.

If there are any further questions or implications about this document, feel free to contact me at [brice.hilgemann@gmail.com](mailto:brice.hilgemann@gmail.com). Thanks and enjoy!

Philosophy

I never intended to become a Computer Science major but upon taking an introductory course, I became hooked by the possibilities CS offered.  There are so many great things that Computer Science offers a person and when attempting to derive what my CS philosophy involved, three words stuck out to me.  It is by coincidence, they also make a nice acronym – CCC.

**Creation** – this is my favorite part of CS that is about creating something useful.  CS allows a person to take an idea and create something from it that others can use.  Creating applications that make life easier, perform tasks that would take too long for humans to do, or simply entertain.

**Collaboration** – this is a great component of CS that brings people of all backgrounds together.  Programming languages translate across many different languages and communities throughout the Internet can be found to assist with problems. Groups of people come together for the common interest of working together to solve a problem.  Documentation can also be found in multiple locations to help a novice become a professional. Companies and groups also work on giant projects to create a single product that thousands of people can benefit from.

**Change** – since Computer Science is a fairly new discipline and technology is constantly evolving, it is up to us to continually change.  To prevent obsolescence, we must continually learn and adapt to situations around us.  Computer Science has taught me to always be open-minded and to always be looking for ways to improve upon things.

All in all, Computer Science has shown me to take a different approach and outlook on life in general.  So much of what Computer Science does is show how you can derive value from different actions.  By taking something that challenges you to think logically and become a life-long learner, I know that I will find fulfillment in all that I do.  This is by far the best career choice I have made and it is something that constantly keeps me moving.

Career Plans

After I graduate from St. Norbert College in May of 2010, I plan to transition into the workforce.  Although the job market is not very friendly right now, I was able to use my knowledge from St. Norbert college and internship experience to gain an entry level position at Georgia-Pacific, LLC in Green Bay as a Programmer/Analyst.  In this position, I will be supporting and developing applications to assist with a very large package known as PassPort.  I will be working on the same team from my internship and am very excited!!  
  
I will also be getting married in July of 2011 and will continue to stay around the Green Bay area.  Although there are unknowns in my future, I am confident that with the skills I have acquired from St. Norbert College that I will succeed.

Project Item Locations

Documentation documents/CCR\_Documentation\_hilgbs.docx

Images/Graphics images/ or documents/psd

Program Executables client/ or server/

Instructions essentials.txt

Diagrams diagrams/

Website http://compsci.snc.edu/cs460/hilgbs/

Code client/ or server/

Support Downloads other\_downloads/

**All other essential files can be founder under my website under the documents area.**

**CS460: Capstone Experience Project Definition**

**2010 DC Pankratz**

**COMPUTER CONTROLLED RAILROAD**

**Name**: Brice Hilgemann

*Project Definition*: Design and implement interfaces for the CCR that campus visitors can use to

control trains.

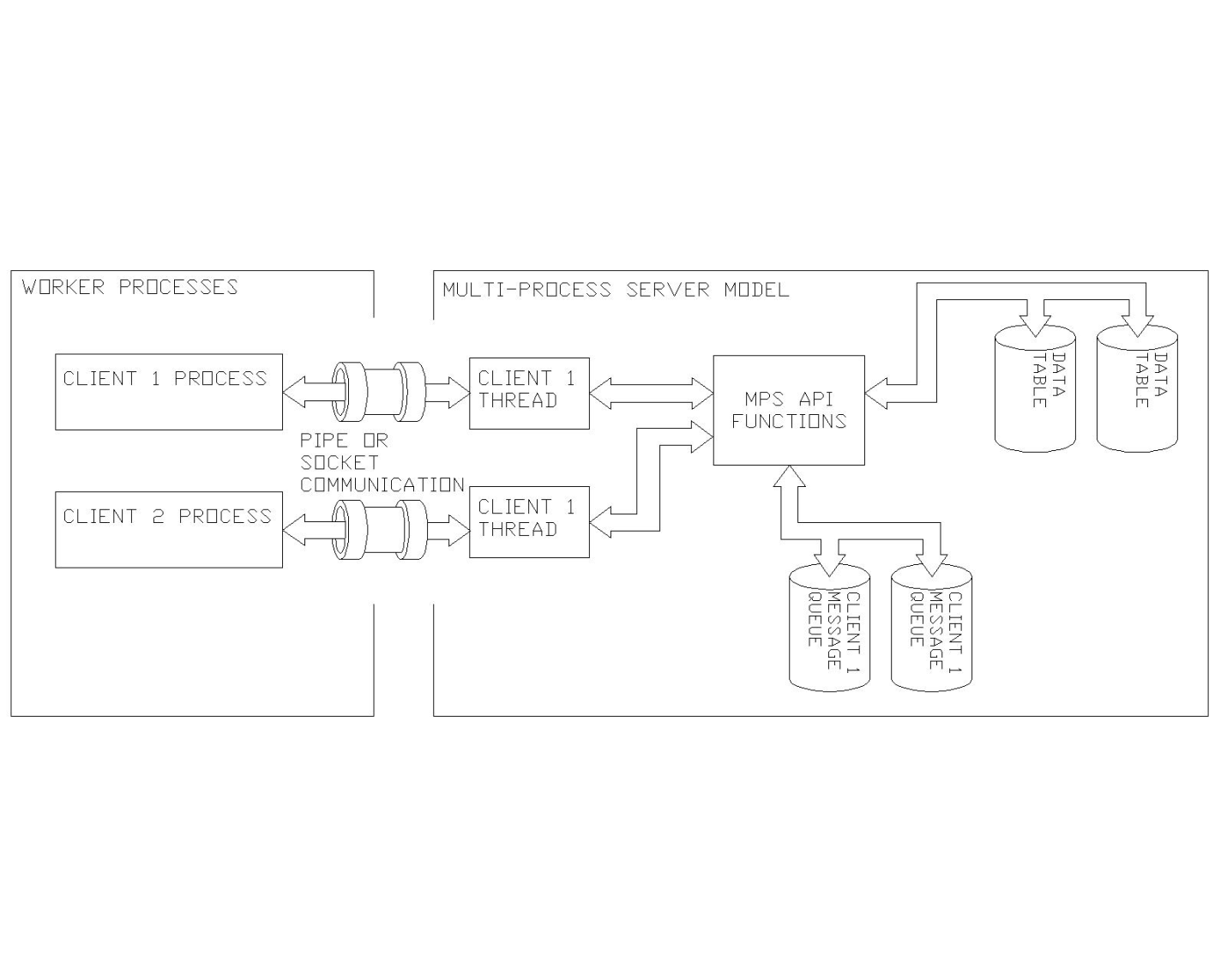
**General Description and Requirements:**  
    1. Ensure PC control of the layout (motors, turnouts, sensors)  
    2. Consider multiple interfaces including:  
           a. PC (mouse/keyboard)  
           b. Mobile devices  
           c. Wireless devices   
    3. Interfaces can be easily used by operators w/o DCC or computer experience  
    4. Provide visual feedback and a log of the layout states  
    5. Develop a simplified operating system that handles multiple trains and prevents collisions  
  
**Resources:**  
    1. Senior Project 2009: Steve Vandenbush  
    2. CSCI370 2009: TOS Design  
    3. WiThrottle for WiFi mobile app interfaces

Previous Versions

**Chris Kratz’s Project**

Description – Chris did a lot of work on not only building the track that we are currently using for the railroad, but the communication and messaging needed to be done. His work shows this client-server processing and direct communication with the track.

Preview:



Location – [G:\PANKDC\ccr\vandenbush\CCR\CCR\DCCManager](file:///D:\bshilgem_ccr_2010\CCR_Documentation_hilgbs.docx)

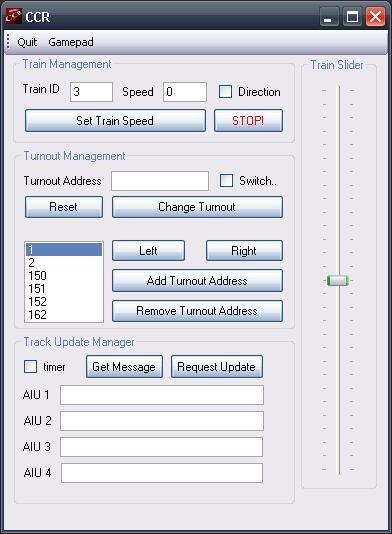
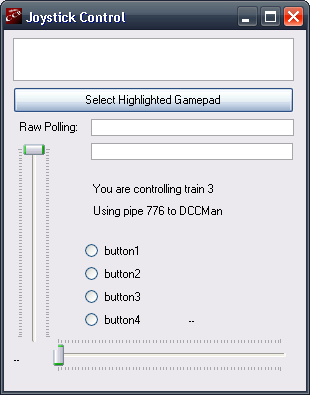
[G:\PANKDC\ccr\vandenbush\CCR\CCR\tos\_api](file:///D:\bshilgem_ccr_2010\CCR_Documentation_hilgbs.docx)

<http://compsci.snc.edu/cs460_archive/2007/kratcl/>

**Steve Vandenbush’s Project**

Description – Steve expanded on Chris’s work to bring a graphical interface to control the trains. He adapted Chris’s work to be used in C# and also allowed for the manipulation of trains using a game controller.

Preview:

Location – [G:\PANKDC\ccr\vandenbush](file:///D:\bshilgem_ccr_2010\CCR_Documentation_hilgbs.docx)

<http://compsci.snc.edu/cs460_archive/2009/vandsh/>

Initial Strategy

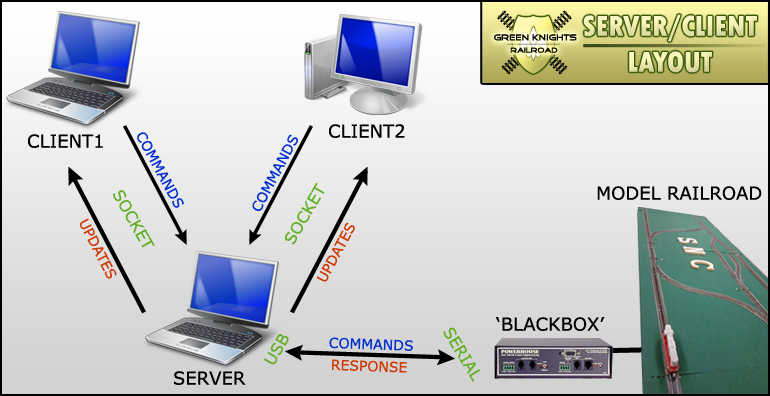
Initial Thoughts – When I first received this project, I had some previous experience looking at the complexities and problems that came with controlling multiple trains on a model railroad. From this, I knew that I had to keep a strong focus on communication to allow multiple trains to function with track hardware and prevent ownership issues/collisions. By taking concepts that I had assisted in creating for CS370, I began planning and looking at what language would work best for this project.

Since this project is geared towards anyone being able to control and manage trains, I knew that I needed a very intuitive visual project. My immediate thought was to use C# which offers form programming with visual items to allow images and events. Could this language offer the communication and structures needed to allow this to be dynamic? I began researching and found that C# had a class that assisted with asynchronous communication and callbacks that utilize threads and assist with the complexities of this task.

Initial Goals/Concepts:

* Get familiar with hardware
* Understand previous code and methods
  + Why did the other users decide to do their projects that way and what were the benefits and issues that they encountered?
* Utilize as much previous work as possible
* Implement OS principles/strategy from CS370
* Look at the benefits of asynchronous processing/sockets over named pipes
* Look at how to prevent ownership issues - monitors in C#
* Configure project to be dynamic and allow multi-user access

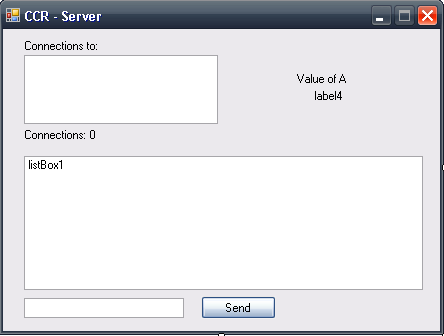
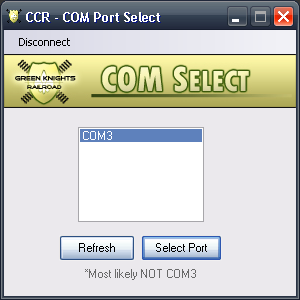
Client-Server Design



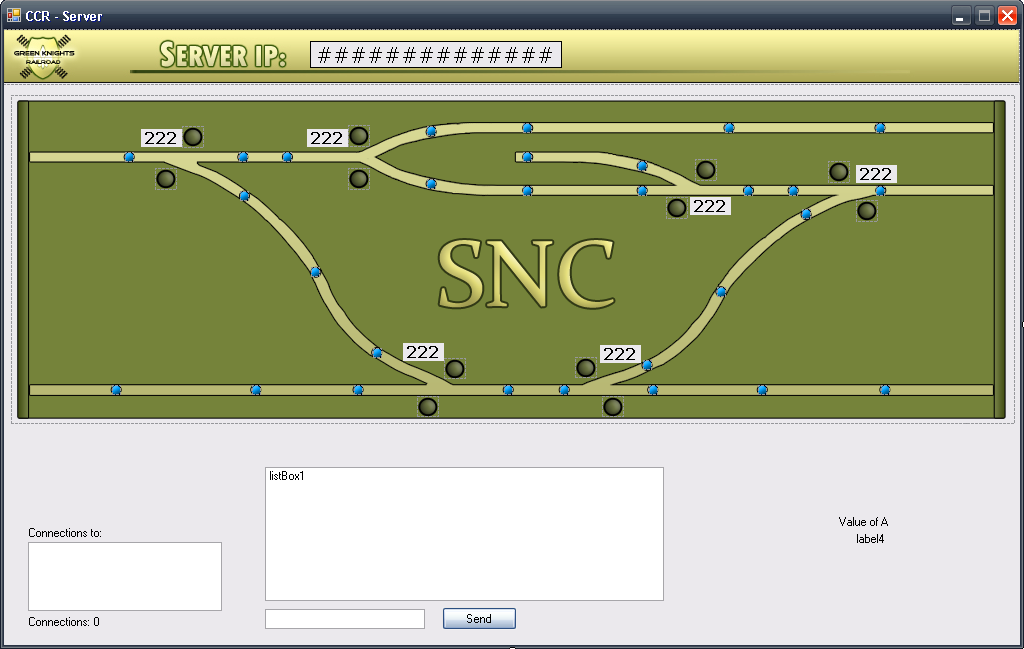
Server Interface

**Server Design Progression**

Phase1:

Phase2:



Final Layout:



**Main Job**

The main job of the server is to hold the ownership of structures and make sure that requests from clients to do conflict with each other. The server tracks train progress on the screen with visuals, handles connections from users and disconnects, allows trains to be added and removed, and communicates directly with the train station.

**Structures in Server**

Connections Structure (listView1) – holds connections

Turnout Structure (listView2) – holds turnout information

Track Layout Structure (listView4) – holds track information (lookup table)

Resource Structure (listView5) – holds currently owned track information

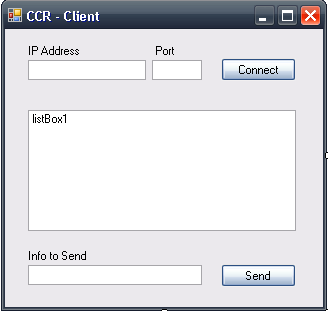
Train Structure (listView7) – holds current train information

AIU Structure (listView9) – holds default AIU information to base retrieved status

Client Interface

**Client Design Progression**

Phase1:



Phase2:



Final Layout:

**Main Job**

The main job of the client is to be a visual tool for users to send requests to the server to move their train. Through indicators represented by buttons, they are able to click these to toggle or change the train’s attributes. The user is able to move a train forward, stop the train, move a train backward, toggle a turnout, or toggle the train’s direction.

**Structures in Client**

There are no structures on the client as all information about the train is retrieved from the server. To make sure that there are no conflicts and to see that a request can actually be approved, all processing needs to be done on server-side and the status of items is just shown visually through the client.

Communication – The Protocol

Below is the standard format for which commands or messages are sent from the server to the client and vice versa. These commands are sent across sockets where callbacks are then set up once again to handle more communication.

**Protocol to Server**

|  |  |
| --- | --- |
| **CMD FORMAT** | **DESCRIPTION** |
| [ 1 ] ; [ TRAIN ID ] | Increase speed or up arrow issued |
| [ 2 ] ; [ TRAIN ID ] | Decrease speed or down arrow issued |
| [ 3 ] ; [ TRAIN ID ] | Stop train has been issued |
| [ 4 ] ; [ TRAIN ID ] | Direction forward has been issued |
| [ 5 ] ; [ TRAIN ID ] | Direction backward has been issued |
| [ 6 ] ; [ TO # ] | Toggle turnout has been issued |
| [ 7 ] ; [ TRAIN ID ] ; [ DIRECTION ] ;  [ ORIENTATION ] ; [ COLOR ] ; [ TRACK ] | User wants to Add a train from the server – get basic information from form and call with protocol to add |
| [ 9 ] ; [ TRAIN ID ] | Toggle light has been issued |
| [ 10 ] ; [ TRAIN ID ] | Toggle horn has been issued |

**Protocol to Client**

|  |  |
| --- | --- |
| **CMD FORMAT** | **DESCRIPTION** |
| [ 0 ] | There are no trains available – disconnect |
| [ 1 ] ; [ TRAIN ID ] ; [ SPEED ] ; [ DIRECTION ] ; [ LIGHT ] ; [ HORN ] ; [ BLOCKED ] ;  [ TO\_NEAREST ] ; [ NBR TOS ] ; [ TO# ST ]... | General update of train information sent from the server to the client to update images and status indicators |
| [ 2 ] | Train has been deleted or client has been disconnected from server |
| [ 3 ] ; [ TRAIN ID ] ; [ NBR TOS] ; [ TO # ]... | Beginning train information for setting labels |

Structures Explained

**Train Structure** (listView7)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Train ID | Owner | Track ID | Rqst Track | Blocked | Speed | Direction | Orientation | Light | Horn | Next Sensor | Speed Lmt | Color |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Train ID | Train ID associated with train | # |
| Owner | Client connection number in control of train | # or 'none' |
| Track ID | Current track that the train is on/owns | # or 'none' |
| Rqst Track | Next track that the train is moving towards/owns | # or 'none' |
| Blocked | Indicates whether the train is blocked | bool |
| Speed | Current speed that train is going | # |
| Direction | Current direction that train is going | Forward or Reverse |
| Orientation | Current orientation that the train is facing on track | Left or Right |
| Light | Light indicator of train | bool |
| Horn | Horn indicator of train | bool |
| Next Sensor | Next sensor that train should run into | # |
| Speed Lmt | Speed limit on current track that train cannot go over | # |
| Color | Color associated with this train | text |

**Connections Structure** (listView1)

|  |  |  |
| --- | --- | --- |
| Train ID | Count | IP and Port |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Train ID | Train ID associated with train | # |
| Count | Client connection number in control of train | # or 'none' |
| IP and Port | The current connection address | IP or 'Available' |

**AIU Structure** (listView9)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| # | Holds current status of that AIU ID | # |

**Resource Structure** (listView5)

|  |  |  |  |
| --- | --- | --- | --- |
| Train ID | Turnout ID | Turnout St | Nearest TO |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Train ID | Train ID associated with train that owns track | # |
| Turnout ID | Turnout ID associated with track | # or 'none' |
| Turnout St | Turnout State needed for this track | ‘top’, ‘bottom’, or ‘none’ |
| Nearest TO | Nearest Turnout based off of track | # |

**Track Structure** (listView4)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Left Trk | Left Trk2 | Right Trk | Right Trk2 | Turnout ID | Speed Lmt | Trk Lgth | Turnout St | Left Sens | Right Sens | Nearest TO |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Left Trk | Track to the left of current track # | # or ‘dead’ |
| Left Trk2 | 2nd set of track to left of current track # - TO involved | # or 'none' |
| Right Trk | Track to the right of current track # | # or ‘dead’ |
| Right Trk2 | 2nd set of track to right of current track # - TO involved | # or 'none' |
| Turnout ID | Turnout ID associated with track | # or ‘none’ |
| Speed Lmt | Speed limit on current track that train cannot go over | # |
| Trk Lgth | Track length associated with track (currently not used) | # |
| Turnout St | Turnout State needed for this track | ‘top’, ‘bottom’, or ‘none’ |
| Left Sens | Sensor to left of track | # or ‘none’ |
| Right Sens | Sensor to right of track | # or ‘none’ |
| Nearest TO | Nearest Turnout based off of track | # |

**Turnout Structure** (listView2)

|  |  |  |
| --- | --- | --- |
| Turnout ID | Turnout St | Owner |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Turnout ID | Turnout ID | # or 'none' |
| Turnout St | Current Turnout State | ‘top’, ‘bottom’, or ‘none’ |
| Owner | Signifies if train is on track that where Turnout is | # or ‘none’ |

**These are the main notable structures that are used – other objects are used throughout but these are the main utilized to ensure ownership.**

Issues Resolved

**Technical Resolutions**

* AIU processing
* New command protocol for new box
* Multiple train handling
* Collision control
* Log information
  + Printing logs
  + Saving logs
* Serial port communication
* Asynchronous threads and client/server communication
* Client and server disconnects
* Visuals to show positioning
* Ownership issues
* Dynamic addressing issues
* Speed limits

**Other Resolutions**

* Easy-to-use interfaces
* WiFi connectivity
* Simplified OS
* Event-driven

Hardware Documentation

**AIU Setup**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 50 | |  | 54 | |  | 58 | |  | 60 | |
| Dipswitch Setup |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| NULL > |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | Left Track | | | | |  | Right Track | | | | |

**Switch-It Setup**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 12 | 24 | 36 |  | 4 | 8 | 16 |
|  |  |  |  |  |  |  |
| Left Track | | |  | Right Track | | |

**NCE CAB Address**

|  |
| --- |
| 02 |
|  |

**Drivers for RS232-USB Device** – located under documents/other\_downloads/serial\_to\_usb

Other NCE hardware documentation can be found on Google, through the NCE links on my web-page, or from the manual that DCP has.

Quick User’s Guide

Follow the below instructions to utilize this program to control the track. If there are any issues in using the program, please verify that you have the correct drivers to utilize the RS232-USB device and also have .NET Framework 3.5 installed on your computer.

Files that are needed: Server.exe with associated .txt files that hold default track information

Client.exe

**Setting up the Server**

1. Connect all of the track pieces together and plug in both of the cords to give the track power.
2. Switch both power adapters for the ‘Blackbox’ and ensure that all three green lights are lit on the track. Turnouts and other indicators should also be up.
3. Plug USB for direct connection into the computer/laptop that is going to be used as the server
4. Right-click on ‘My Computer’ on the desktop and select ‘Manage’ to access Computer Management. Here you can verify that a COM is associated with the USB
   1. Click ‘Device Manager’ and then expand the Communications tab to verify that a COM is set up with the USB. It should be listed as a Prolific USB device.
5. Launch Server.exe from the associated folder where the .txt files for track defaults are located. It is crucial that these are in the same folder for setup purposes.
6. Locate the server IP and port listed at the top for client connection

The server can now add trains through the ‘Add Trains Dialog’, remove trains once added through the ‘Remove Trains Dialog’, and see all connections. If there are any questions about operations that the server can perform, please view my videos on my site or review documentation.

**Setting up the Client**

1. Launch Client.exe
2. Retrieve the server IP and port that is listed on top of the computer that is running the server
3. Verify that a train has been added so that when you click ‘Connect’, you will have control of a train to then send commands
4. Click ‘Connect’

The client can now have the train move in different directions, stop the train, toggle turnouts, and perform other train-specific actions. Status is retrieved automatically from the server and some commands will not worked if the train is listed as ‘BLOCKED’ or it wouldn’t be in the user’s interest to change.

Presentation Documentation







