**SOFTWARE DESIGN SPECIFICATION DOCUMENT**

© Axon Guidance

CSc 191 Senior Project

Department of Computer Science - College of Engineering and Computer Science

California State University, Sacramento

Version 11.24.2009

# 1. INTRODUCTION

This is the Software Design Specification document for the RFID Triangulation and Inventory project sponsored by the NAND Solutions Group of Intel.

This project is being undertaken by Axon Guidance development team. The team is comprised of undergraduate students majoring in Computer Science at California State University, Sacramento. The team members are enrolled in a two-semester senior project course required of all undergraduate majors. Successful delivery of the desired software product will fulfill the senior project requirement for the student team members.

The identification of the project sponsor and the team are as follows:

PROJECT SPONSORS:

Svanhild Simonson

Validation Engineer

[svanhild.m.simonson@intel.com](mailto:svanhild.m.simonson@intel.com)

Ronald Peroni

Validation Manager

[ronald.j.peroni@intel.com](mailto:ronald.j.peroni@intel.com)

AXON GUIDANCE DEVELOPMENT TEAM:

Team Lead:

Steven Salmons

408.881.3310

[stevefett@gmail.com](mailto:stevefett@gmail.com)

Team Members:

Salil Nizar

Forrest Slater

Morgan Darke

* 1. **PURPOSE**

The Software Design Specification defines the design of the product. It specifies the architecture of the product and how the different components of the project interact to satisfy the requirements in the SRS.

* 1. **SCOPE**

This document represents the baseline design specification for the RFID Triangulation and Inventory project. It specifies the different components of the product and how they interact. In addition, it defines the interface provided to the user and the design of each component. Any changes to this document after baseline approval will be made through the submission of a Baseline Change Request form.

* 1. **DEFINITIONS AND ACRONYMS**
     1. **DEFINITIONS**

**Baseline**: A baseline is a work product that has been formally reviewed and accepted by the involved parties. A baseline is changed only through formal configuration management procedures.

**Software Requirements Specification**: Documentation of the essential requirements (functions/features/uses, performance, design constraints, and attributes) of the software and its external interfaces.

**Non-functional Requirements:** A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behavior

**Project Sponsor:** The customer. Due to the academic nature of this project, the sponsor has no financial responsibilities for the development of the product.

**1.3.2. ACRONYMS**

RFID: Radio Frequency Identification. Small antenna tags with memory storage containing identification information

PHP: PHP Hypertext Preprocessor. Web-based language to be used in the user interface.

SQL: Structured Query Language. A relational database system to be used in the project.

SSD: Solid State Drive, in this case NAND based hard disks.  
 OLE: Object Linking and Embedding, a Microsoft-specific method of distributing objects.

* 1. **OVERVIEW OF DOCUMENT CONTENTS**

Section 2: Architectural Design – Describes the hardware and software necessary for the project and how they communicate.

Section 2.1:Hardware Configuration – Describes the hardware necessary for the project.

Section 2.2: Software Design Architecture – A model of the systems behavior as it interacts with outside entities.

Section 3: Interface Design – Describes the look, feel and behavior of the components of the system visible to the user.

Section 3.1-n: Webpages– Screenshots and description of each webpage.

Section 4: Database Schema – Describes the relational database used in the project.

Section 4.1: ER Diagram – Shows the entities and relationships described by the data.

Section 4.2: Creating the database – Statements to create the database.

Section 4.3: Stored Procedures – Specifications for procedures that can be invoked from the application.

Section 5: Component Design Specifications – Detailed description of the design of the software.

Section 5.1: Sequence Diagrams – Shows the interaction between components.

Section 5.1.1 – 5.1.12

Section 5.2: Design Specifications

Section 5.2.1 – 5.2.n

Section 6: Performance Analysis – Describes any issues or constraints uncovered by the SDS.

Section 7: Feasibility and Resource Estimates – Describes the resources required to build, operate, and maintain the software.

Section 8: Software Requirements Traceability Matrix – Relates the design components to the requirements.

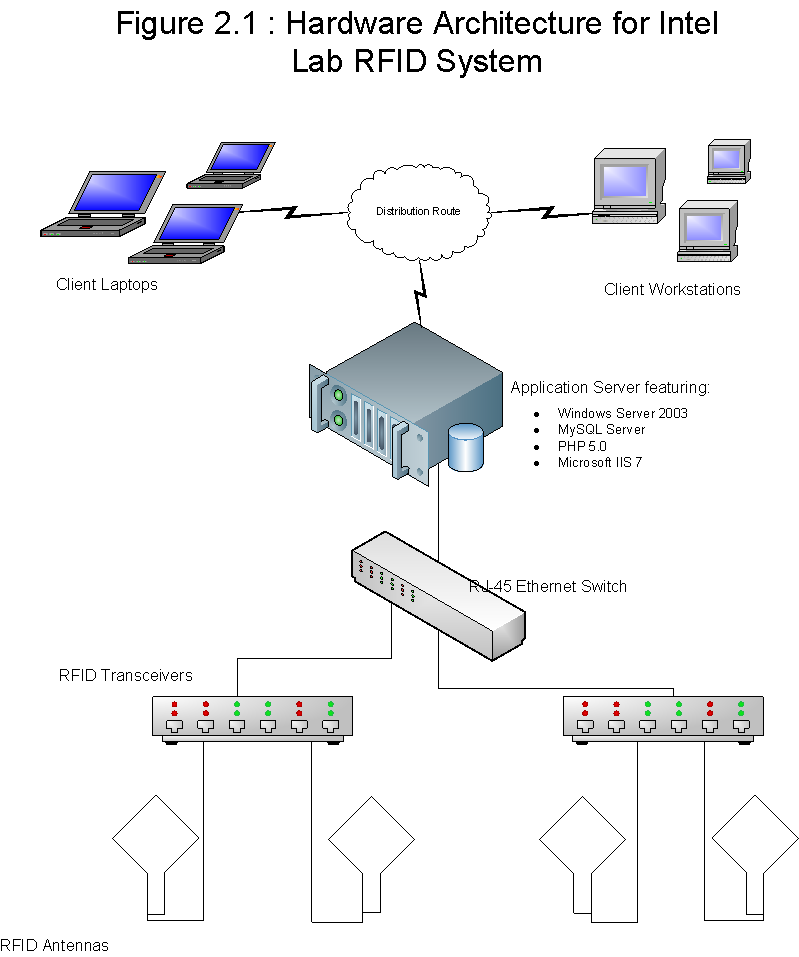
Section 9: Approvals – The list of key signatories necessary to sign-off on the SDS.

Appendix A: Data Dictionary – A listing of all the data elements, data structures, and data tables representing the system.

# 2 ARCHITECTURAL DESIGN

## 2.1 Hardware Architecture

The hardware design of the Lab RFID Tracking System is a three-tier architecture involving an RFID transceiver, Application Server, and web clients. RFID antennas, connected two per transceiver, will be directly connected via RJ-45 Cat6 Ethernet to 100mbit switches. On the same subnet, an application server will communicate with the transceivers via the network. A direct connection to the corporate network will supply access to the application via a web browser with Java accessibility. The following diagram demonstrates this architecture:



## 2.2 Software Design Architecture

### 2.2.1 Presentation Layer

The RFID Tracking software will provide its functionality via a PHP-generated browser based GUI. Location tracking queries requiring graphical output will be serviced via an integrated applet from within the browser. The functional aspects of the software can be categorized as belonging to one of these two feature sets:

* **Administrative Tasks:** These features involve maintenance, monitoring, and accessibility to the software system. User account creation, privilege and access rights settings, system environment configurations, and database sanitation functions exist within this category. System environment configurations encompass the functions needed to control antennas, lab rooms to be monitored, physical layout of the antenna arrays, and other antenna/transceiver level tasks. Database sanitation includes performing database optimizations and backups.
* **User Tasks:** These features involve the tasks that the normal user would perform with the system. Tracking objects, viewing object location histories, assigning item ownership, and adding new item tags to the database make up the main feature set for this category. Most of these features will be displayed via the embedded Java applet.

Due to the desire to modularly support the addition of future lab environments, the software solution utilizes Java for the graphical representation. This will allow the software to display a variety of floorplans based on the lab area being scanned, as well as other critical “runtime” graphical functions that a real-time tracking system requires. Text-only features such as the full suite of Administrative Tasks will be purely PHP-generated HTML to increase performance, as well as maximize accessibility for administrative functionality.

### 2.2.2 Business Logic Layer

The major components that make up the composite software system involve the following:

* **Transceiver Network Driver:** The application server will communicate with the antenna grid via a Transceiver Network Driver, which is written in C. It makes use of a vendor-provided API to communicate with the ThingMagic Transceiver hardware via its Ethernet network interface. The C network driver will handle all antenna commands that the PHP engine requests based upon user input or scheduled requests. Request results will be stored directly in the SQL database.
* **PHP Parsing Engine:** The PHP parser takes the software’s PHP-coded pages as input and outputs HTML compliant web pages based on the business logic within the input code. The PHP engine will parse business logic designed to handle Administrative Tasks, User Tasks, and background tasks generated by client requests. The PHP logic will distribute tasks to one of two sources depending on the client request issued: the Database Server to read or insert data or the IIS 7 Web Server to display HTML for non-graphical output.
* **Microsoft IIS 7.0 Web Services**: The IIS service will handle page request transactions between the client and the server. It will display PHP forms to handle input, or HTML tables to display output. It will perform middleman services between the client and the PHP parser as well as the client and the embedded Java applet.
* **Java Triangulation Applet**: The embedded java applet will directly interact with the SQL server and handle location triangulation based on the antenna data stored within. It will also rely upon the SQL server for lab room floorplan information. There will be no direct communication between the PHP generated web forms and the Java Applet, so all triangulation-based queries that require graphical output will be handled by the applet alone.

### 2.2.3 Data Management Layer

Data management services will be provided by mySQL Enterprise Server. Application drivers from within the C Network Driver, PHP 5.0 Parsing Engine, and Java will provide the connectivity APIs for each application. Data integrity, sanity, and validation will be handled at input time to reduce the validity-checking overhead by the Database Management System. An emphasis will be placed on a high performance, high concurrency configuration due to the real-time requirements of the system.

### 2.2.4 Diagram of Software System2

This diagram shows the various components of the project, how they interact, and what they are responsible for.3 INTERFACE DESIGN

## Use Cases

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Description** | **Webpage** |
| Show Item Location | The Actor specifies the items to locate. The system either looks up the last known location recorded in the database or performs a new scan on the RFID transceiver to locate the items. The system then displays the location to the actor. | locateitem.php |
| Specify Item Search Criteria | The actor provides information about the items to locate. This information can be a serial number, tag ID number, item type (e.g. SSD, Test Equipment, Protocol Analyzer, Laptop, etc.), item description, target antenna(s), or keyword. | locateitem.php |
| Specify Item Characteristics | The actor inputs the information about the item to add. This information can be a serial number, tag ID number, item type (e.g. SSD, Test Equipment, Protocol Analyzer, Laptop, etc.), item description, or keyword. | itemcontrol.php |
| Add Item | The Actor inputs the characteristics of the item to add to the program and places an RFID tag on the write station. The system writes the serial number to the RFID tag. Then the actor sticks RFID tag to item. | itemcontrol.php |
| Show All Items in Lab | The Actor requests to see all items currently in the lab. The system queries each antenna in the lab and displays a list of all items detected. | locateitem.php |
| Scan Area For Items | Each specified antenna returns the tags that it detected. | locateitem.php |
| Manage Antennas | This use case allows the Lab Manager actor to add or update the antenna array in the case of addition, removal, or transfer of antenna nodes in the lab. | antennacontrol.php |
| Manage User Accounts | This feature allows the Lab Manager actor to add, delete, or modify the user accounts for the software system. | usercontrol.php |
| Manage Item Categories | The lab manager is able to add, modify, or remove an Item Category. Examples include “hard drive,” “protocol analyzer,” or any other type of item classifications that items can be divided into. | itemcategorycontrol.php |
| Manage Lab Rooms | Lab rooms will be added, deleted, or edited via this feature. Critical elements about the room, such as a room name and description, length and width dimensions in meters (of the largest lengths and widths for non-rectangular rooms), and a floorplan image of each room are kept here. | roomcontrol.php |
| List Items that Cross Door Threshold | The user specifies a date or series of dates that ask the system to check for any items that have entered or left the lab on the specified dates. To detect if an item has left the lab, 3 scan histories will be used: A previous scan that shows the item was in the room, a previous scan showing the item picked up near the door antenna, and a scan showing the item no longer in the room. The opposite sequence of scans will determine if the item has entered the room. | locationalhistory.php |

## Navigation Diagram

This diagram shows the links between the webpages in the project. Each page is linked to all other pages.

# 4 DATABASE SCHEMA

## 4.1 ER Diagram



## 4.2 Creating the Database

Create Statements:

CREATE TABLE `antennatable` (

`AntennaName` varchar(30) DEFAULT NULL,

`AntennaID` int(11) NOT NULL AUTO\_INCREMENT,

`AntennaX` double NOT NULL,

`AntennaY` double NOT NULL,

`RoomID` int(11) NOT NULL COMMENT 'Foreign Key into RoomTable',

PRIMARY KEY (`AntennaID`),

KEY `RoomID` (`RoomID`)

);

CREATE TABLE `itemclasstable` (

`ItemClassID` int(11) NOT NULL AUTO\_INCREMENT,

`ItemClassName` varchar(20) NOT NULL,

`ItemClassDescription` varchar(500) NOT NULL,

PRIMARY KEY (`ItemClassID`),

UNIQUE KEY `ItemClassName` (`ItemClassName`)

);

CREATE TABLE `itemtable` (

`TagID` int(11) NOT NULL AUTO\_INCREMENT,

`SerialNumber` varchar(40) NOT NULL,

`ItemMake` varchar(30) NOT NULL,

`ItemModel` varchar(40) NOT NULL,

`ItemClassID` int(11) NOT NULL COMMENT 'FK Into ItemClassTable',

`KeywordList` varchar(255) NOT NULL,

`AddedBy` int(11) NOT NULL COMMENT 'FK into UserTable',

`DateAdded` date NOT NULL,

`ModifiedBy` int(11) NOT NULL COMMENT 'FK into UserTable',

`DateModified` date NOT NULL,

`CurrentOwner` int(11) DEFAULT NULL COMMENT 'FK into UserTable',

`Enabled` tinyint(4) NOT NULL,

PRIMARY KEY (`TagID`),

KEY `ItemClassID` (`ItemClassID`),

KEY `AddedBy` (`AddedBy`),

KEY `ModifiedBy` (`ModifiedBy`),

KEY `CurrentOwner` (`CurrentOwner`)

);

CREATE TABLE `locationtable` (

`LocationID` int(11) NOT NULL AUTO\_INCREMENT,

`TagID` int(11) NOT NULL COMMENT 'FK into ItemTable',

`FoundBy` varchar(50) NOT NULL COMMENT 'FK into AntennaTable',

`PowerFound` varchar(100) NOT NULL,

`ScanTime` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`RoomID` int(11) NOT NULL COMMENT 'FK into RoomTable',

PRIMARY KEY (`LocationID`),

KEY `TagID` (`TagID`),

KEY `FoundBy` (`FoundBy`),

KEY `RoomID` (`RoomID`)

);

CREATE TABLE `roomtable` (

`RoomID` int(11) NOT NULL AUTO\_INCREMENT,

`RoomName` varchar(30) NOT NULL,

`RoomDescription` varchar(255) NOT NULL,

`RoomXDimension` double NOT NULL,

`RoomYDimension` double NOT NULL,

`RoomGraphic` longblob NOT NULL,

PRIMARY KEY (`RoomID`),

UNIQUE KEY `RoomName` (`RoomName`)

);

CREATE TABLE `usertable` (

`UserID` int(11) NOT NULL,

`UserName` varchar(20) NOT NULL,

`FirstName` varchar(20) NOT NULL,

`LastName` varchar(30) NOT NULL,

`Password` varchar(20) NOT NULL,

`Permissions` int(11) NOT NULL,

PRIMARY KEY (`UserID`),

UNIQUE KEY `UserName` (`UserName`)

);

ALTER TABLE `antennatable`

ADD CONSTRAINT `antennatable\_ibfk\_1` FOREIGN KEY (`RoomID`) REFERENCES `roomtable` (`RoomID`) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE `itemtable`

ADD CONSTRAINT `itemtable\_ibfk\_1` FOREIGN KEY (`ItemClassID`) REFERENCES `itemclasstable` (`ItemClassID`),

ADD CONSTRAINT `itemtable\_ibfk\_2` FOREIGN KEY (`AddedBy`) REFERENCES `usertable` (`UserID`),

ADD CONSTRAINT `itemtable\_ibfk\_3` FOREIGN KEY (`ModifiedBy`) REFERENCES `usertable` (`UserID`),

ADD CONSTRAINT `itemtable\_ibfk\_4` FOREIGN KEY (`CurrentOwner`) REFERENCES `usertable` (`UserID`);

ALTER TABLE `locationtable`

ADD CONSTRAINT `locationtable\_ibfk\_1` FOREIGN KEY (`TagID`) REFERENCES `itemtable` (`TagID`),

ADD CONSTRAINT `locationtable\_ibfk\_2` FOREIGN KEY (`RoomID`) REFERENCES `roomtable` (`RoomID`);

# 5 Component Design Specifications

|  |  |  |
| --- | --- | --- |
|  | **Use Case Components** | |
| **Use Case Name** | **Webpage** | **Java Files** |
| Show Item Location | locateitem.php | ControllerIO.java, Database.java, Triangulate.java |
| Specify Item Search Criteria | locateitem.php | ControlPanel.java |
| Specify Item Characteristics | itemcontrol.php | None. |
| Add Item | itemcontrol.php | None. |
| Show All Items in Lab | locateitem.php | ControllerIO.java, Database.java, MapPanel.java |
| Show Item Details | locateitem.php |  |
| Manage Antennas | antennacontrol.php | None. |
| Manage User Accounts | usercontrol.php | None. |
| Manage Item Categories | itemcategorycontrol.php | None. |
| Manage Lab Rooms | roomcontrol.php | None. |
| List Items that Cross Door Threshold | locateitem.php |  |

## 5.1 Sequence Diagrams

### 5.1.1 Show Item Location Sequence Diagram



### 5.1.2 Specify Item Search Criteria Sequence Diagram



### 5.1.3 Specify Item Characteristics Sequence Diagram



### 5.1.4 Add Item Sequence Diagram



### 5.1.5 Show All Items In Lab Sequence Diagram



### 5.1.6 Show Item Details Sequence Diagram



### 5.1.7 Manage Antennas Sequence Diagram



### 5.1.8 Manage User Accounts Sequence Diagram



### 5.1.9 Manage Item Categories Sequence Diagram



### 5.1.10 Manage Lab Rooms Sequence Diagram



### 5.1.11 List Items that Cross Door Threshold Sequence Diagram



## 5.2 Design Specifications

### 5.2.1 Antenna.java

|  |  |
| --- | --- |
| **Preconditions** | Database connection must be established. |
| **Interface** | static class AntennaEntry (Antenna antenna, double powerfound)   * This static class provides an Antenna object and the power by   that antenna in decibels.  Antenna AntennaEntry.getAntenna()   * Returns the specified Antenna associated with the AntennaEntry.   The primary purpose of this is too match the antenna entries with proper antenna on the grid.  void setDecibels (double decibels)   * This sets the decibels of the current antenna, derived from the corresponding AntennaEntry.   double getDecibels ()   * Returns the decibels of the current antenna   void draw (Graphics2D g)   * Paints the antenna visually on the map panel.   MouseClicked (MouseEvent e)   * Selects the antenna |
| **Processing Specifications** | 1 – Pull Antennas from database  2 – Draw Antennas on map panel using draw() method  3 – Pull AntennaEntrys from database  4 – Match correct Antennas to AntennaEntries using AntennaEntry.getAntenna method  5 – Use getter methods on Antennas, including getDecibels() and  getLocation () , for scanning and possible triangulation |
| **Database Requirements** | Antennas are pulled from the database from one mass query and then stored sequentially in a list. The query matches all of the antennas room ID’s that match the current room.  "Select \* From antennatable Where RoomID=" + current |
| **Postconditions** | Antenna’s provide information about their location and decibels level pertaining to a particular asset, for triangulation purposes. Antenna’s also draw themselves onto the map panel. |

### 5.2.2 Asset.java

|  |  |
| --- | --- |
| **Preconditions** | Database connection must be established. |
| **Interface** | String setID (String id)   * Sets the identification of the asset   String getID ()   * Gets the identification of an asset   void ping ()   * Visually pings the asset on the map panel, this involves a call to the draw method   void draw (Graphics2D g)   * Paints the asset visually on the map panel   void mouseClicked (MouseEvent e)   * Pings the asset   void mouseEntered (MouseEvent e)   * Visually displays the assets information |
| **Processing Specifications** | 1 – Query all Antennas the detect the asset  1 – Pinpoint Asset location through Scanner class  2 – Create Asset based on location and accuracy of the location  3 – Visually display all found assets during last scan |
| **Database Requirements** | Assets positions are derived from all the antennas that spot it. AntennaEntry represents all the antenna’s that can see the asset.  "Select AntennaName, AntennaID, AntennaX, AntennaY, RoomID " + "From antennatable " + "Where AntennaID=" + antennaStrings[i]; |
| **Postconditions** | The assets is visually displayed for the user. The asset is dynamic in that the user can ping it visually anytime |

### 5.2.3 ControllerIO.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | void loadWorkspace(int roomID)   * Pulls the Room and Antenna information from the database. Creates the workspace environment and visually displays it on the map panel.   void Vector<Antenna> getHotAntennas()   * Returns the decibels readings of all Antennas pertaining to the particular asset being searched   void initSweep(int [] tagIDs)   * Scans the current workspace for the Assets matching the tagIDs contained in the parameter   void displayCurrentAssets (Vector<Asset> assets)   * Visually maps all the assets found by the sweep. |
| **Processing Specifications** | 1 – Connect to the database  2 – Build the current workspace  3 – Execute scans for the assets |
| **Database Requirements** | ControllerIO makes method calls to the database class in order initially connect to the database, and then run it’s queries.  Connection conn = Database.*connect*("root", "moocow", "stevefett.gotdns.com", "seniorproject"); |
| **Postconditions** | The ControlllerIO is responsible for connecting to the database and building the current workspace and executing scans for assets.  All database calls are handled within ControllerIO. |

### 5.2.4 ControlPanel.java

|  |  |
| --- | --- |
| **Preconditions** | GUI Frame must be instantiated. |
| **Interface** | None |
| **Processing Specifications** | 1 – Designates area for SearchPanel  2 – Creates SearchPanel and adds it to ControlPanel |
| **Database Requirements** | None |
| **Postconditions** | Creates foundation for SearchPanel. |

### 5.2.5 Database.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | static Connection connect(String user, String password, String host, String database)   * Establishes database connection   static Vector<Antenna> getAntennas(Connection conn, int room)   * Returns all the Antennas in the current room   static Vector<Antenna.AntennaEntry> getLocationInfo(Connection conn, int tagID)   * Returns all of the AntennasEntrys in the can currently see the Asset matching the parameter tagID   static Room getRoom(Connection conn, int ID)   * Returns the Room matching the ID |
| **Processing Specifications** | 1 – Connect to the Database  2 – Run Queries |
| **Database Requirements** | Multiple Database Calls are made. Here is how the original connection is made:  String url = "jdbc:mysql://" + host + "/" + database;  Class.forName("com.mysql.jdbc.Driver").newInstance();  return DriverManager.getConnection(url, user, password); |
| **Postconditions** | The Database class is responsible for connecting the database and returning the results of all possible queries. |

### 5.2.6 FieldSquare.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | Vector<Antenna> getAntennas()   * Returns the Antennas in the field square |
| **Processing Specifications** | 1 – Four Antennas are placed in FieldSquare  2 – Field Square returns antennas in same order |
| **Database Requirements** | None |
| **Postconditions** | The FieldSquare represents a virtual square of Antennas in the Room for triangulations purposes. |

### 5.2.7 MainFrame.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | void init ()   * Creates and initializes all major classes and panels   void setWorkspace ()   * Commands ControllerIO to create the current Workspace |
| **Processing Specifications** | 1 – Create GUI Frame  2 – Create and initialize all major classes  3 – Set current Workspace |
| **Database Requirements** | None |
| **Postconditions** | Initializes the GUI Frame and major components. |

### 5.2.8 MapPanel.java

|  |  |
| --- | --- |
| **Preconditions** | GUI frame must be instantiated. |
| **Interface** | void setBackgroundImage(Image i)   * Sets the background image of the panel   void setGridDimesions(Dimension d)   * Sets the dimensions of the panel, width and height   void paintComponent(Graphics g)   * Paints the grid lines, background image, assets, and antennas of the panel   void update(IObservable o, WorkspaceObject arg)   * Updates the panel visually with current WorkspaceObject informaion |
| **Processing Specifications** | 1 – Set image and grid dimensions of panel  2 – Paint and repaint panel |
| **Database Requirements** | None |
| **Postconditions** | The MapPanel represents the interactive and dynamic visual display of the current Workspace |

### 5.2.9 ObjectInfoBox.java

|  |  |
| --- | --- |
| **Preconditions** | An arbitrary WorkspaceObject must exist. |
| **Interface** | paintComponent(Graphics g)   * Paints and sets the information text of the box |
| **Processing Specifications** | 1 – Create ObjectInfoBox based on WorkspaceObject  2 – paint box with current WorkspaceObject information |
| **Database Requirements** | None |
| **Postconditions** | The ObjectInfoBox provides a visual display of information based on the moused over WorkspaceObject. |

### 5.2.10 Room.java

|  |  |
| --- | --- |
| **Preconditions** | Must be connected to Database. |
| **Interface** | BufferedImage getImageFromGraphic()   * Returns the image that represents the room   int getRoomId()   * Returns a numerical identification of the Room   String getRoomName()   * Returns the designated name of the room |
| **Processing Specifications** | 1 – Create Room  2 – Hold and return Room information |
| **Database Requirements** | GetImageFromGraphic makes a call to ControllerIO to get the current background image from the Database. |
| **Postconditions** | The abstract data type for the current room within the workspace. |

### 5.2.11 Scanner.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | void sweep(Vector<Antenna> antennas)   * Attempts to locate and create the asset through various techniques by utilizing the location and decibel levels of the passed in Antennas   Asset getCurrentAsset()   * Returns the last asset created by the Scanner class |
| **Processing Specifications** | 1 – Sweep the workspace for the desired Asset  2 – Create the Asset once located  3 – Return the Asset to the ControllerIO |
| **Database Requirements** | None |
| **Postconditions** | The Scanner class is responsible for locating and creating an Asset derived from a list of pertinent Antenna’s readings. |

### 5.2.12 SearchPanel.java

|  |  |
| --- | --- |
| **Preconditions** | ControlPanel must be initialized. |
| **Interface** | void actionPerformed(ActionEvent e)   * Sends search data from the text field to the ControllerIO |
| **Processing Specifications** | 1 – Search criteria combo box and search parameter text field are created.  2 – Information from search text field are passed to the ControllerIO for queering. |
| **Database Requirements** | None |
| **Postconditions** | Handles all search queries by the user. |

### 5.2.13 Traingulation.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | void tSharp (FieldSquare fs, int antennaCount)   * Triangulates the location of the asset based upon the passed in fieldsquare parameter and number of antennas in that square (greater than or equal to three)   Void trough (Vector<Antenna antennas, int antennaCount)   * Roughly triangulates the location of the asset based upon the passed in antennas vector and number of antennas in that vector (greater than or equal to three)   int getX ()   * Returns the x-position of the triangulated asset.   int getY ()   * Returns the y-position of the triangulated asset. |
| **Processing Specifications** | 1 – Triangulates the Asset  2 – Returns the position coordinates |
| **Database Requirements** | None |
| **Postconditions** | Triangulates the position of an Asset, based on pertinent Antenna’s readings. |

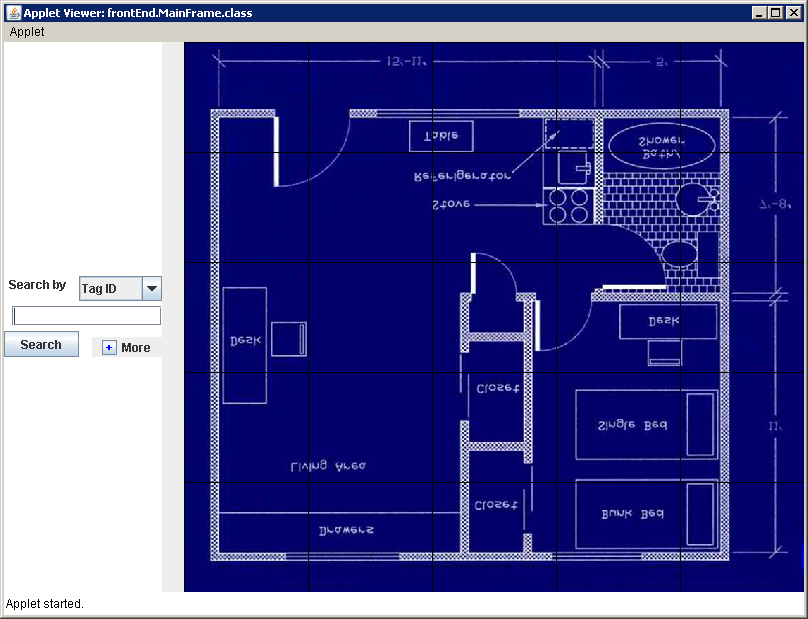
### 5.2.14 Workspace.java

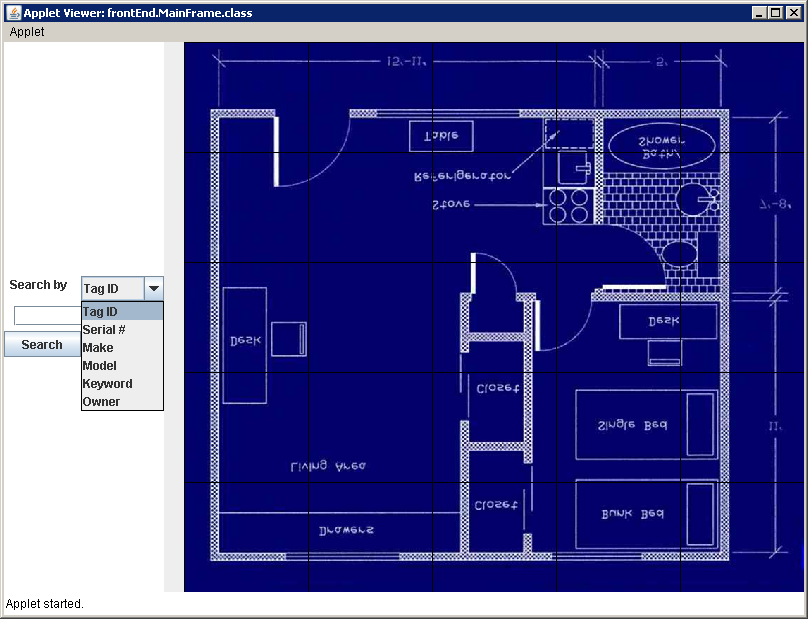
|  |  |
| --- | --- |
| **Preconditions** | Room and Antennas in room must me instantiated. |
| **Interface** | void setBackgroundImage()   * Sets the background image   Vector<Antenna> getAntennas()   * Returns antennas in workspace   Dimension getGridDimension()   * Returns dimensions of the room; width and height   Image getBackgroundImage()   * Returns the current background image   Vector<FieldSquare> getFieldSquares()   * Returns the FieldSquares in the workspace |
| **Processing Specifications** | 1 – Sets the components of the Workspace  2 – Returns information to the ControllerIO |
| **Database Requirements** | None |
| **Postconditions** | The abstract data type for the current working environment. |

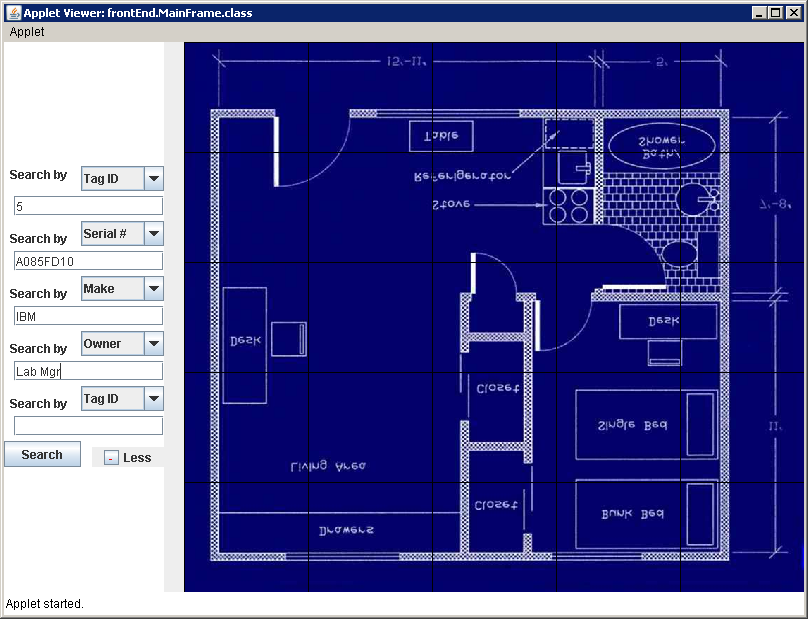
### 5.2.15 WorkspaceObject.java

|  |  |
| --- | --- |
| **Preconditions** | None |
| **Interface** | void setSelected (boolean selected)   * Sets if the WorkspaceObject has been selected by the user or not   boolean isSelected ()   * Returns if the WorkspaceObject is currently selected   void setLocation(int x, int y)   * Sets the location of the WorkspaceObject   Dimension getLocation()   * Returns the location of the WorkspaceObject   int getHeight ()   * Returns height of WorkspaceObject   int getWidth ()   * Returns width of WorkspaceObject   Boolean contains (double begX, double begY, double endX, double endY)   * Returns if WorkspaceObject is within the points of the passed in parameters |
| **Processing Specifications** | Note Applicable |
| **Database Requirements** | None |
| **Postconditions** | This is an abstract class that defines all objects within the Workspace |

Java Applet Screenshots:



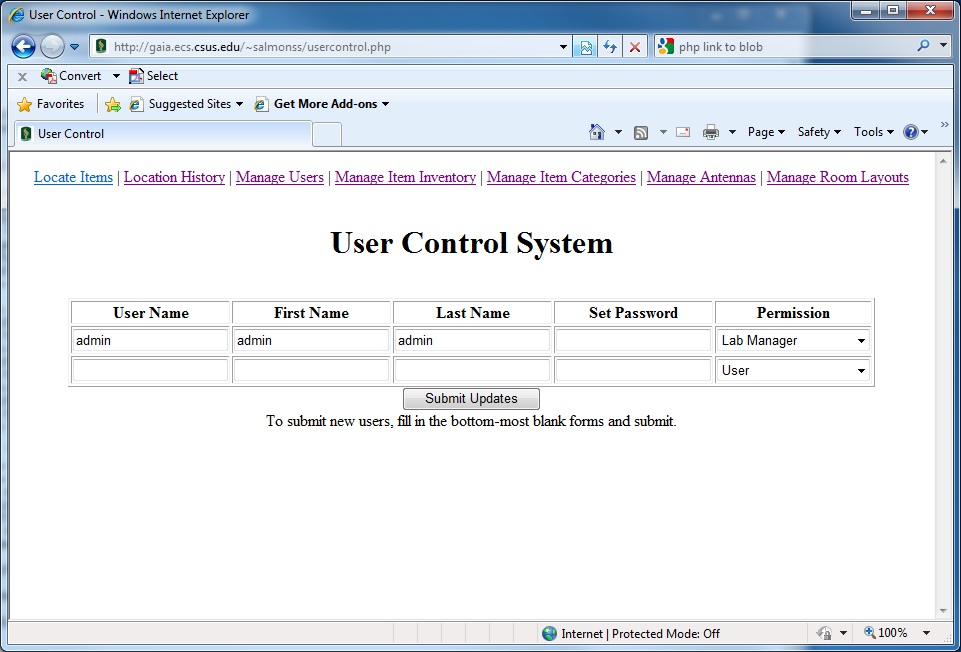




### 5.2.16 usercontrol.php

|  |  |
| --- | --- |
| **Preconditions** | Only lists yourself if logged in and not privileged account. |
| **Interface** | None. |
| **Processing Specifications** | Data is queried from db to fill text fields of all existing user accounts but leaves password field empty. Passwords can only be reset, not viewed. This screen will only show the current user if not a privileged account. Input sanitation handled by DBMS. |
| **Database Requirements** | UserTable |
| **Postconditions** | No postprocessing necessary. |

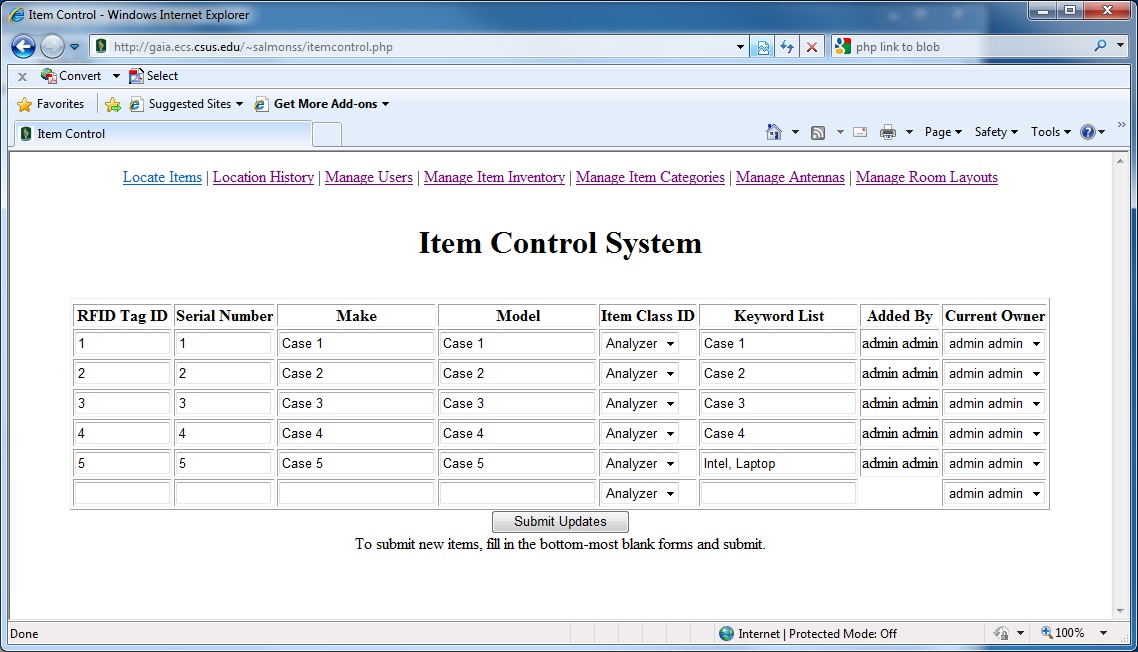
Screenshot:



### 5.2.17 itemcontrol.php

|  |  |
| --- | --- |
| **Preconditions** | Read-only if not a privileged login, only viewable if logged in. |
| **Interface** | None. |
| **Processing Specifications** | Data is queried from db to fill text fields of all existing items. Only admins can modify, anyone can add. Input sanitation handled by DBMS. |
| **Database Requirements** | ItemTable |
| **Postconditions** | No postprocessing necessary. |

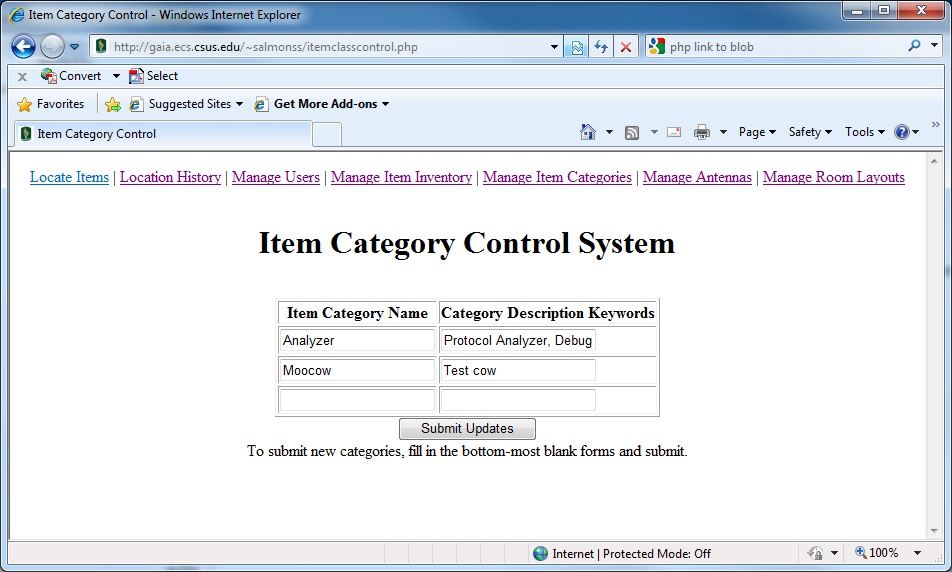
Screenshot:



### 5.2.18 itemcategorycontrol.php

|  |  |
| --- | --- |
| **Preconditions** | Read-only if not a privileged login, only viewable if logged in. |
| **Interface** | None. |
| **Processing Specifications** | data is queried from db to fill text fields of all existing items. Only admins can modify, anyone can add. Input sanitation handled by DBMS. |
| **Database Requirements** | ItemCategoryTable |
| **Postconditions** | No postprocessing necessary. |

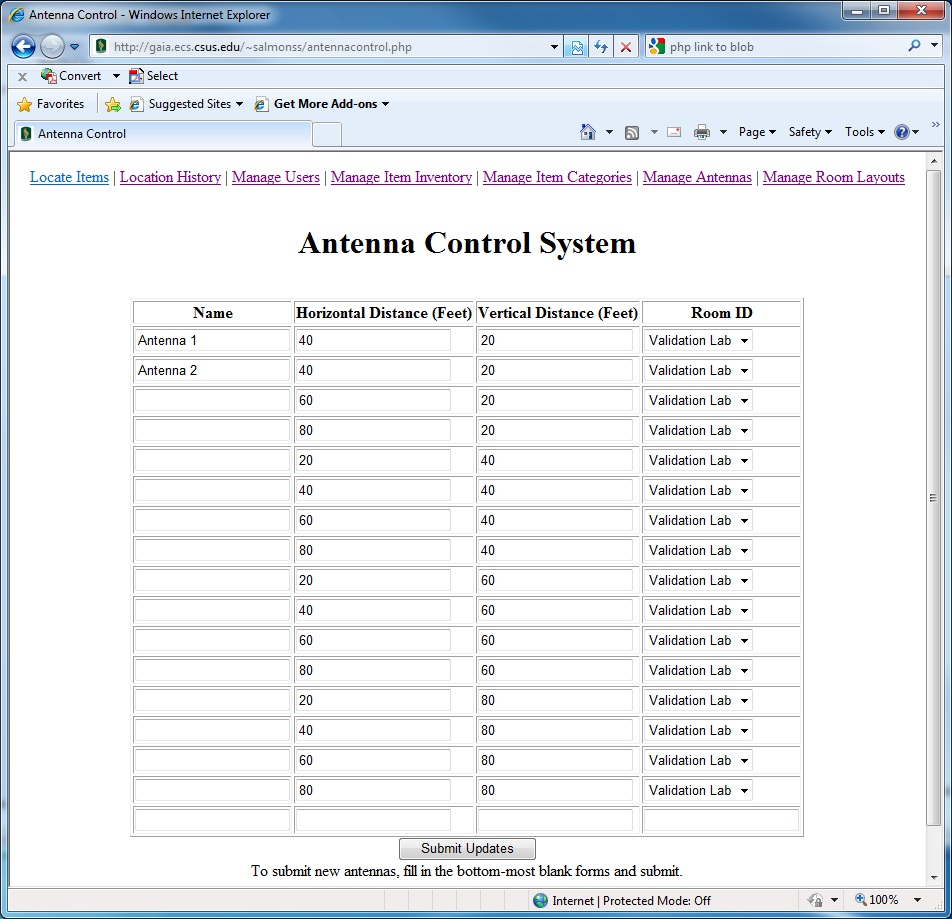
Screenshot:



### 5.2.19 antennacontrol.php

|  |  |
| --- | --- |
| **Preconditions** | Read-only if not a privileged login, only viewable if logged in. |
| **Interface** | None. |
| **Processing Specifications** | Data is queried from db to fill text fields of all existing antennas. Only admins can modify, anyone can add. Input sanitation handled by DBMS |
| **Database Requirements** | AntennaTable |
| **Postconditions** | No postprocessing necessary. |

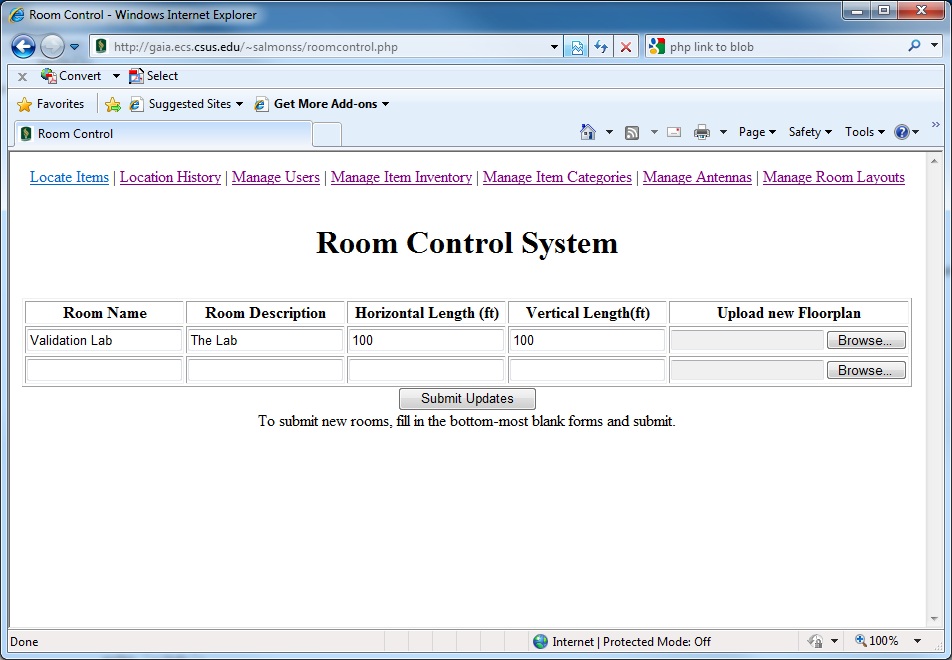
Screenshot:



### 5.2.20 roomcontrol.php

|  |  |
| --- | --- |
| **Preconditions** | Read-only if not a privileged login, only viewable if logged in. |
| **Interface** | None. |
| **Processing Specifications** | Data is queried from db to fill text fields of all existing rooms. Only admins can modify, anyone can add. Input sanitation handled by DBMS |
| **Database Requirements** | RoomTable |
| **Postconditions** | No postprocessing necessary. |

Screenshot:



### 5.2.21 main.php

|  |  |
| --- | --- |
| **Preconditions** | No preconditions. |
| **Interface** | None. |
| **Processing Specifications** | None. |
| **Database Requirements** | UserTable for login. |
| **Postconditions** | Login must be valid using user/password combination. |

### 5.2.22 locationalhistory.php

|  |  |
| --- | --- |
| **Preconditions** | Viewable/useable by all. |
| **Interface** | None |
| **Processing Specifications** | Input is in the form of multi-select boxes and date pulldowns. Guaranteed sanitization. |
| **Database Requirements** | ItemTable, AntennaTable |
| **Postconditions** | No postprocessing necessary. |

### 5.2.23 locateitem.php

|  |  |
| --- | --- |
| **Preconditions** | Viewable/useable by all. |
| **Interface** | Must properly embed java applet. |
| **Processing Specifications** | Provides an embedded applet view. |
| **Database Requirements** | All tables but UserTable. |
| **Postconditions** | No postprocessing necessary. |

# 6 PERFORMANCE ANALYSIS

The loading of the image of the room within the java applet was much slower than expected, taking 1-2 seconds. This was an issue because a user can navigate away from the applet and back to it, causing the image to be reloaded. In order to alleviate this problem, when the user tries to navigate away from the applet, the applet is moved to the background instead of closed.

# 7 FEASIBILITY AND RESOURCE ESTIMATES

The system will be implemented in the following fixed performance environment:

* 1 Application Server -- To run the Microsoft IIS 7.0 Web Server, mySQL database, and the Transceiver-to-SQL driver.
* 1 Transceiver per 4 Antennae

# 8 SOFTWARE REQUIREMENTS TRACEABILITY MATRIX

This section relates the design in this document to the requirements specified in the SRS.

|  |  |
| --- | --- |
| **SDD Paragraph** | **SRS Paragraph** |
| Section 2, Paragraph 1 | Section 3.3 Paragraph 1 |
| Section 2.2.1 | Section 2.2, Section 3.4.7 |
| Section 2.2.2 | Section 3.4.1, 3.4.2 |
| Section 2.2.3 | Section 3.4.3, Paragraph 2 |
| Section 3 | Section 2.2 |
| Section 4 | Section 3.1.13 |
| Section 6, Paragraph 1 | Section 3.4.3, Paragraph 3 |
| Section 7, Paragraph 1 | Section 3.2, Paragraph 1 |

**Axon Guidance**

**Sign-Off Sheet**

**Faculty Advisor**

Dr. Ying Jin, Faculty Advisor, CSUS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Axon Guidance**

Steve Salmons, Project Manager, Axon Guidance \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Salil Nizar, Team Member, Axon Guidance \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Morgan Darke, Team Member, Axon Guidance \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Forrest Slater, Team Member Axon Guidance \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**APPENDIX A: DATA DICTIONARY**

**A.1: DATA ELEMENTS**

|  |  |
| --- | --- |
| **Element Name** | **Description** |
| AddedBy | 1. *Description*: User ID number of user that added the item to the  system.  2. *Set By:* Software uses currently logged in user id at time of creation.  3. *Used By:* Item table to keep track of who added what items.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* None |
| AntennaID | 1. *Description*: Uniquely generated ID number for each antenna node.  2. *Set By:* Database auto-generated.  3. *Used By:* Antenna table unique identifiers  4. *Data Structures:* None  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* Unique values only. Primary key. |
| AntennaName | 1. *Description*: A descriptive name of the antenna’s location.  2. *Set By:* User input  3. *Used By:* Room and antenna configurations, antenna-specific scans.  4. *Data Structures:* None  5. *Variable Type:* Char(30)  6. *Validity Rules:* Unique, alphanumeric only (no spaces) |
| AntennaX | 1. *Description*: The “X” distance from the room’s (0,0) coordinate in  meters.  2. *Set By:* User input  3. *Used By:* Room and antenna configurations, antenna-specific scans,  calculating locations.  4. *Data Structures:* AntennaCoordinate  5. *Variable Type:* Double  6. *Validity Rules:* None |
| AntennaY | 1. *Description*: The “Y” distance from the room’s (0,0) coordinate in  meters.  2. *Set By:* User input  3. *Used By:* Room and antenna configurations, antenna-specific scans,  calculating locations.  4. *Data Structures:* AntennaCoordinate  5. *Variable Type:* Double  6. *Validity Rules:* None |
| CurrentOwner | 1. *Description*: User ID number of user that is currently assigned  responsibility of the item.  2. *Set By:* Software uses currently logged in user id at time of creation.  3. *Used By:* Item table to keep track of who added what items.  4. *Data Structures:* OptionalCharacteristics  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* None |
| DateAdded | 1. *Description*: Date timestamp of the time the item was added.  2. *Set By:* Database generated.  3. *Used By:* Item table characteristic.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* Date  6. *Validity Rules:* None |
| DateModified | 1. *Description*: Date timestamp of the time the item characteristics were  last modified.  2. *Set By:* Database generated.  3. *Used By:* Item table transaction history notekeeping.  4. *Data Structures:* None  5. *Variable Type:* Date  6. *Validity Rules:* None |
| Enabled | 1. *Description*: Bit specifying whether item is enabled or disabled.  2. *Set By:* User input.  3. *Used By:* Item maintenance.  4. *Data Structures:* Required Characteristics.  5. *Variable Type:* TinyInt (1 byte).  6. *Validity Rules:* 0 for disabled or 1 for enabled. |
| FirstName | 1. *Description*: User’s real first name.  2. *Set By:* User input.  3. *Used By:* User account maintenance.  4. *Data Structures:* FullName  5. *Variable Type:* Char(20)  6. *Validity Rules:* UNICODE |
| FoundBy | 1. *Description*: Comma separated list of AntennaID numbers that pick up  the TagID.  2. *Set By:* Software algorithm.  3. *Used By:* Location history, location triangulation algorithm.  4. *Data Structures:* None  5. *Variable Type:* Varchar  6. *Validity Rules:* Integers and comma delimiters only. |
| ItemClassDescription | 1. *Description*: Descriptive string explaining the type of item that falls  into this category..  2. *Set By:* User input.  3. *Used By:* ItemClass maintenance.  4. *Data Structures:* None  5. *Variable Type:* Varchar  6. *Validity Rules:* None |
| ItemClassID | 1. *Description*: Predefined item categories to broadly classify items into  sortable groups.  2. *Set By:* Automatically generated by Database.  3. *Used By:* Item Table queries (histories and full item details). Primary  key of ItemClass table, foreign key in Item table.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* Integer  6. *Validity Rules:* Unique |
| ItemClassID | 1. *Description*: Unique ID used for item category records in the ItemClass  table.  2. *Set By:* Automatically generated by database.  3. *Used By:* Primary key into ItemClass table, foreign key for Item table.  4. *Data Structures:* None  5. *Variable Type:* Integer  6. *Validity Rules:* Unique |
| ItemClassName | 1. *Description*: Name for an item category type, such as “Solid State  Drive”.  2. *Set By:* User input.  3. *Used By:* Item table categorization.  4. *Data Structures:* None  5. *Variable Type:* Char(20)  6. *Validity Rules:* Alphanumeric with spaces. |
| ItemMake | 1. *Description*: The item manufacturer’s name.  2. *Set By:* User input  3. *Used By:* Item Table record required characteristic.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* Char(30)  6. *Validity Rules:* All UNICODE characters (for international support). All  characters forced to capitalized forms. |
| ItemModel | 1. *Description*: The model name of the item.  2. *Set By:* User input  3. *Used By:* Item Table record required characteristic.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* Char(40)  6. *Validity Rules:* UNICODE characters. All characters forced to  capitalized forms. |
| KeywordList | 1. *Description*: Comma separated list of keywords describing the item.  2. *Set By:* User input  3. *Used By:* Optional characteristic for item table search criteria.  4. *Data Structures:* OptionalCharacteristics  5. *Variable Type:* Varchar  6. *Validity Rules:* All capitalized alphanumeric, comma delimited. |
| LastName | 1. *Description*: User’s real last name.  2. *Set By:* User input.  3. *Used By:* User account maintenance.  4. *Data Structures:* FullName  5. *Variable Type:* Char(30)  6. *Validity Rules:* UNICODE |
| LocationID | 1. *Description*: Generated ID number given to each record in the item  location history table.  2. *Set By:* Automatically generated by Database.  3. *Used By:* Item Location queries (histories and triangulation). Primary  key of Location table.  4. *Data Structures:* None  5. *Variable Type:* Integer  6. *Validity Rules:* Unique |
| ModifiedBy | 1. *Description*: User ID number of user that last edited the item in the  system.  2. *Set By:* Software uses currently logged in user id at time of editing.  3. *Used By:* Item table to keep track of who modified what items.  4. *Data Structures:* None  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* None |
| Password | 1. *Description*: User’s account password.  2. *Set By:* User input.  3. *Used By:* User account security validation.  4. *Data Structures:* None  5. *Variable Type:* Char(20)  6. *Validity Rules:* None |
| Permissions | 1. *Description*: Access rights level for a user account.  2. *Set By:* User input.  3. *Used By:* User account security.  4. *Data Structures:* None  5. *Variable Type:* Integer  6. *Validity Rules:* 0 for guest, 1 for technician, 2 for sysadmin. |
| PowerFound | 1. *Description*: Comma separated list of decibel values needed to locate  the target TagID on each antenna that found it.  2. *Set By:* Software algorithm.  3. *Used By:* Location triangulation algorithm.  4. *Data Structures:* None  5. *Variable Type:* Double  6. *Validity Rules:* None |
| RoomDescription | 1. *Description*: Descriptive string explaining the room’s purpose,  location, or any other defining factors.  2. *Set By:* User input.  3. *Used By:* Room record maintenance.  4. *Data Structures:* None  5. *Variable Type:* Varchar  6. *Validity Rules:* None |
| RoomGraphic | 1. *Description*: Storage space for room’s floorplan graphic.  2. *Set By:* User input.  3. *Used By:* Graphical user interface for an item’s triangulated location.  4. *Data Structures:* None  5. *Variable Type:* BLOB  6. *Validity Rules:* JPEG, GIF, BMP |
| RoomID | 1. *Description*: Uniquely generated ID number for lab rooms.  2. *Set By:* Database auto-generated.  3. *Used By:* Location history table foreign key, room table primary key.  4. *Data Structures:* None  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* Unique |
| RoomName | 1. *Description*: Name or facilities room number of a lab.  2. *Set By:* User input.  3. *Used By:* Room record maintenance.  4. *Data Structures:* None  5. *Variable Type:* Char(20)  6. *Validity Rules:* Alphanumeric with spaces. |
| RoomXDimension | 1. *Description*: Longest length in meters of the longest part of the room.  2. *Set By:* User input.  3. *Used By:* Location triangulation algorithm.  4. *Data Structures:* RoomDimension  5. *Variable Type:* Double  6. *Validity Rules:* None |
| RoomYDimension | 1. *Description*: Widest width in meters of the widest part of the room.  2. *Set By:* User input.  3. *Used By:* Location triangulation algorithm.  4. *Data Structures:* RoomDimension  5. *Variable Type:* Double  6. *Validity Rules:* None |
| ScanTime | 1. *Description*: Database generated timestamp of the location scan.  2. *Set By:* Software algorithm.  3. *Used By:* Location histories, location triangulation.  4. *Data Structures:* None  5. *Variable Type:* Date  6. *Validity Rules:* None |
| SerialNumber | 1. *Description*: The item’s manufacturer assigned serial number.  2. *Set By:* User input  3. *Used By:* Item Table record required characteristic.  4. *Data Structures:* RequiredCharacteristics  5. *Variable Type:* Char(50)  6. *Validity Rules:* Unique |
| TagID | 1. *Description*: Uniquely generated ID number for RFID tag, based on  attached object’s serial number hashing.  2. *Set By:* Software-generated based on SerialNumber  3. *Used By:* Primary key for Item table, foreign key in location history.  4. *Data Structures:* None  5. *Variable Type:* 4 byte Integer  6. *Validity Rules:* Unique |
| UserID | 1. *Description*: Uniquely generated ID number for each user account.  2. *Set By:* Automatically generated by Database.  3. *Used By:* User authentication, setting ID numbers in  added/modified/owner fields. Used as foreign keys in Item table,  primary key in User table.  4. *Data Structures:* None  5. *Variable Type:* Int  6. *Validity Rules:* Unique |
| UserName | 1. *Description*: Username used for actors to log into the system.  2. *Set By:* User input.  3. *Used By:* User logins.  4. *Data Structures:* None  5. *Variable Type:* char(20)  6. *Validity Rules:* Unique |

**A.2 DATA STRUCTURES**

|  |  |
| --- | --- |
| RequiredCharacteristics | 1. *Description*: The minimum required item characteristics that must  be defined upon item creation.  2. *Set By:* Software algorithm.  3. *Used By:* Anything that refers to an item.  4. *Data Structures:* None  5. *Composition:* TagID, SerialNumber, ItemMake, ItemModel,  ItemClassID, AddedBy, DateAdded, Enabled. All required. |
| OptionalCharacteristics | 1. *Description*: Optional characteristics for item table entries that are  not required, but give additional search criteria to match.  2. *Set By:* Software algorithm.  3. *Used By:* Item search criteria, Item details listing.  4. *Data Structures:* None  5. *Composition:* KeywordList, ModifiedBy, DateModified,  CurrentOwner. All optional. |
| AntennaCoordinate | 1. *Description*: The x,y coordinates of antenna nodes.  2. *Set By:* User input.  3. *Used By:* Item triangulation calculations, area scans.  4. *Data Structures:* None  5. *Composition:* AntennaX, AntennaY. |
| RoomDimension | 1. *Description*: The total length and width of the longest/widest areas  of the room.  2. *Set By:* User input.  3. *Used By:* Item triangulation calculations, area scans, graphical  interface layout.  4. *Data Structures:* None  5. *Composition:* RoomXDimension, RoomYDimension |
| UserFullName | 1. *Description*: The first and last name of each user in the system.  2. *Set By:* User input.  3. *Used By:* User administration.  4. *Data Structures:* None  5. *Composition:* FirstName, LastName. |

**A.3 DATA TABLES**

|  |  |
| --- | --- |
| AntennaTable | 1. *Description*: This table stores the information relevant to each individual  antenna node in the system.  2. *Set By:* Installation program.  3. *Used By:* Item triangulation and location history generation.  4. *Number of Records:* N records for N antennas in the system.  5. *Expected Record Growth:* Growth should only occur when new rooms  are added to the system, or obstacles require additional antennas  for improved reliability.  6. *Record Size*: 25 to 54 bytes per record  7. Composition:   * AntennaName – 30 byte Varchar * AntennaID – 4 byte Integer, Primary Key * AntennaX – 8 byte Double * AntennaY – 8 byte Double * RoomID – 4 byte Integer, Foreign Key to RoomTable |
| ItemTable | 1. *Description*: This table stores the information relevant to each individual  Item in the tracking system.  2. *Set By:* Installation program.  3. *Used By:* Item triangulation and location history generation.  4. *Number of Records:* N records for N items being tracked.  5. *Expected Record Growth:* Growth and decay will be consistent, reaching  an eventual equilibrium number of items.  6. *Record Size*: 39-404 bytes  7. Composition:   * TagID – 4 byte Integer, Primary Key * SerialNumber – 40 byte Varchar * ItemMake – 30 byte Varchar * Item Model – 40 byte Varchar * ItemClassID – 4 byte Integer, Foreign Key into ItemClassTable * KeywordList – 255 byte Varchar (optional) * AddedBy – 4 byte Integer, Foreign Key into UserTable * DateAdded – 7 byte Date * ModifiedBy – 4 byte Integer, Foreign Key into UserTable * DateModified – 7 byte Date * CurrentOwner – 4 byte Integer, Foreign Key into UserTable (Opt) * Enabled – 1 byte TinyInt |
| LocationTable | 1. *Description*: This table stores the location tracking history for each scan performed on the room.  2. *Set By:* Installation program.  3. *Used By:* Item triangulation and location history generation.  4. *Number of Records:* Minimum of N, Maximum of N\*10 records, N =  number of items being tracked.  5. *Expected Record Growth:* Only items that move often will cause record  growth. That population is estimated to be small.  6. *Record Size*: 21-171 bytes  7. Composition:   * LocationID – 4 byte Integer, Primary Key * TagID – 4 byte Integer, Foreign Key into ItemTable, repeating. * FoundBy – 50 byte Varchar, delimited strings are Foreign Keys into AntennaTable. * PowerFound – 100 byte Varchar, delimited strings of doubles. * ScanTime – 7 byte Date * RoomID – 4 byte Integer, Foreign Key into RoomTable. |
| UserTable | 1. *Description*: This table stores user account information.  2. *Set By:* Installation program.  3. *Used By:* User management and verification.  4. *Number of Records:* N records for N users in the system.  5. *Expected Record Growth:* Growth will occur only during hiring of actors.  6. *Record Size*: 12-102 bytes  7. Composition:   * UserID – 4 byte Integer, Primary Key * UserName – 20 byte Varchar * FirstName – 20 byte Varchar * Last Name – 30 byte Varchar * Password – 20 byte encrypted Varchar * Permissions – 4 byte Integer |
| ItemClassTable | 1. *Description*: This table stores the different item category types to be  used when adding items to the system.  2. *Set By:* Installation program.  3. *Used By:* Item organization.  4. *Number of Records:* N records for N categories in the system.  5. *Expected Record Growth:* Rarely a record or two added.  6. *Record Size*: 6-524 bytes  7. Composition:   * ItemClassID – 4 Byte integer, Primary Key * ItemClassName – 20 byte Varchar * ItemClassDescription – 500 byte Varchar |
| RoomTable | 1. *Description*: This table stores the information relevant to each individual  room using the tracking system.  2. *Set By:* Installation program.  3. *Used By:* System organization.  4. *Number of Records:* N records for N rooms being tracked.  5. *Expected Record Growth:* For the scope of senior project, none.  6. *Record Size*: 22-5350 bytes  7. Composition:   * RoomID – 4 byte Integer, Primary Key * RoomName – 30 byte Varchar * RoomDescription – 255 byte Varchar * RoomXDimension – 8 byte Double * RoomYDimension – 8 byte Double * RoomGraphic – 5000 byte image binary file. |