SFWRENG 3K04: Software Development

Assignment 2 – Part 2 – DCM Design

Dr. Alan Wassyng

December 4, 2019

Group 5: More Life Pacemaker

Adam Bujak – bujaka – 400113347

Eric Hillebrand – hillebre – 400143468

Harneet Singh – singhh76 – 400110275

Karan Gill – gillk20 – 400130973

Stephen Scott – scotts24 – 400139933

Table of Contents

[Likely Requirement Changes 3](#_Toc26383678)

[Likely Design Changes 3](#_Toc26383679)

[Modules 4](#_Toc26383680)

[Application 4](#_Toc26383681)

[GUI 4](#_Toc26383682)

[GUI Controller 4](#_Toc26383683)

[GUI Abstraction Layer 7](#_Toc26383684)

[Data Management 10](#_Toc26383685)

[User Account Manager 10](#_Toc26383686)

[Database Manager 14](#_Toc26383687)

[SQLite ORM Library 17](#_Toc26383688)

[SQLite Database 17](#_Toc26383689)

[Pacemaker Communication 17](#_Toc26383690)

[Protocol 17](#_Toc26383691)

[DCM Communication Controller 17](#_Toc26383692)

[DCM Serial Manager 20](#_Toc26383693)

[Appendix 23](#_Toc26383694)

[Module Flowchart 23](#_Toc26383695)

# Likely Requirement Changes

* The user interface shall be capable of visually indicating when a different pacemaker device is approached than was previously interrogated
* The Set Clock function shall set the data and time of the device
* The New Patient function shall allow a new device to be interrogated without exiting the software application
* Upon the user’s request:
  + A Bradycardia Parameters Report shall be available
  + A Temporary Parameters Report shall be available

# Likely Design Changes

* The current GUI has some issues where certain features aren’t very intuitive. For instance, if all fields are blank the user can still press program, and the last saved data in the database will be programmed to the pacemaker. This can be confusing and may be rectified in later versions.
* Add additional comments onto the interface in order to make it more user friendly
* Change the DCM interface such that it is easier to distinguish between the programmable parameters and the echoed parameters
* Implement a way for the DCM to detect which COM port the pacemaker is on

# Modules

## Application

#### Description

This is the main application module for all other components of the Device Controller-Monitor (DCM). It starts the Graphical User Interface (GUI) and handles the transmission of data between modules.

## GUI

### GUI Controller

#### Description

The GUI controller handles the drawing of the custom interface to the display by calling the GUI Library through the GUI Abstraction Layer (see Appendix for a flowchart of the modules). The GUI controller takes care of tasks such as starting menus, filling forms with values, and building the GUI.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| LoginMenuData() | \_\_init\_\_(self, userNameLabel, passwordLabel, signInButtonText, newUserButtonText)  setCallbacks(self, callbacks) |
| ProgramMenuData() | \_\_init\_\_(self, fieldLabels, buttonTexts, dropDownLabelText, dropDownOptions, currentOption)  setCallbacks(self, callbacks) |
| CreateUserData() | \_\_init\_\_(self, userNameLabel, passwordLabel, createUserButtonText, cancelButtonText)  setCallbacks(self, callbacks) |
| Screen | \_\_init\_\_(self, screenname, data) |
| GUIC | \_\_init\_\_(self)  setCallbacks(self, callbacks)  startGUI(self)  drawScreen(self, screen)  updateGUI(self)  getLoginData(self)  getNewUserData(self)  getProgramData(self)  drawFirstScreen(self)  p\_drawLoginScreen(self, data)  p\_drawProgrammingScreen(self, data)  p\_drawCreateUserScreen(self, data) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Output(s) |
| ALL \_\_init\_\_() FUNCTIONS | Parameters for a class structure | Sets the values of the structure with the given inputs. Returns nothing |
| ALL setCallbacks(self, callbacks) FUNCTIONS | Callbacks | Enables button functionality/interaction. Returns nothing |
| startGUI(self) | None | Draws the first screen. Returns nothing |
| drawScreen(self, screen) | Screen class | Draws a screen of either type Login, Programming, or Create User. Returns nothing |
| updateGUI(self) | None | Updates the interface. Returns nothing |
| getLoginData(self) | Input field data | Returns GUI field data on Login Screen in a LoginData class |
| getNewUserData(self) | Input field data | Returns GUI field data on Create User Screen in a LoginData class |
| getProgramData(self) | Input field data | Returns GUI field data on Programming Screen in a ProgrammedData class |
| p\_drawFirstScreen(self) | None | Draws the Login screen. Returns nothing |
| p\_drawLoginScreen(self, data) | Data/text to fill the Login screen | Draws the Login screen. Returns nothing |
| p\_drawProgrammingScreen(self, data) | Data/text to fill Programming screen | Draws the Programming screen. Returns nothing |
| p\_drawCreateUserScreen(self, data) | Data/text to fill Create User screen | Draws the Create User screen. Returns nothing |

#### Global Variables

|  |  |
| --- | --- |
| Variable(s) | Description |
| LoginData | Imported class from the User Account Manager module, each of which contains a username and password for a user |
| ApplicationCallbacks | Imported class from a file that contains numerous button callbacks |
| Enum | Imported Enum from enum |
| loginScreen | A Screen class created with ScreenNames(0) and the LoginMenuData class |
| programmingScreen | A Screen class created with ScreenNames(1) and the ProgramMenuData class. |
| createUserMenuScreen | A Screen class created with ScreenNames(2) and the CreateUserData class. |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| ALL \_\_init\_\_() FUNCTIONS | For all classes this is a constructor function that is automatically called with the creation of every instance of a class. This function allows the class to initialize itself with the attributes/values it is given. |
| ALL setCallbacks(self, callbacks) FUNCTIONS | Tkinter’s button widget provides a command callback when a user clicks a button. This function enables the functionality for all relevant classes without the need to create extra GUI objects. |
| startGUI(self) | Uses the p\_drawFirstScreen method to start the GUI |
| drawScreen(self, screen) | Clears the window, determines which screen type should be drawn (LOGIN\_SCREEN, PROGRAMMING\_SCREEN, or CREATE\_USER\_SCREEN), and then uses the appropriate method to draw the screen. |
| updateGUI(self) | Updates the GUI |
| getLoginData(self) | Retrieves data from the GUI input fields when the Login Screen is up and returns the data in a LoginData class |
| getNewUserData(self) | Retrieves data from the GUI input fields when the Create User Screen is up and returns the data in a LoginData class |
| getProgramData(self) | Retrieves data from the GUI input fields when the Programming Screen is up and returns the data in a ProgrammedData class |
| p\_drawFirstScreen(self) | Draws the first screen (Login Screen) of the GUI by using self.drawScreen(loginScreen) |
| p\_drawLoginScreen(self, data) | Draws the Login Screen with field labels, button texts, and button callbacks. |
| p\_drawProgrammingScreen(self, data) | Draws the Programming Screen with drop-down labels, drop-down options, field labels, button texts, and button callbacks. |
| p\_drawCreateUserScreen(self, data) | Draws the Create User Screen with field labels, button texts, and button callbacks. |

### 

### GUI Abstraction Layer

#### Description

The GUI Abstraction Layer hides the specific working details of drawing the GUI from the GUI controller in order to create a Separation of Concerns; this helps allow for the reuse and independent maintenance/upgrades of modules.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| GUIAL | \_\_init\_\_(self)  update(self)  setTitle(self, title)  clearWindow(self)  drawTwoFieldsTwoButtonLayout(self, fieldLabels, buttonTexts, buttonCallbacks)  drawNFieldsNButtonsOneDropDownLayout(self, dropDownLabelText, currentDropDownItem, dropDownOptions, fieldLabels, buttonTexts, buttonCallbacks)  getEntryData(self)  displayLabelEntry(self, programMode, dropDownOptions, fieldLabels)  getProgramMode(self)  displayErrorMessageLoginS(self, errorCode)  displayErrorMessageProgramS(self, errorCodeRate, errorCodeChamber)  setNEntryData(self, data) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Output(s) |
| \_\_init\_\_() | Parameters for a class structure | Sets the values of the structure with the given inputs. Returns nothing |
| update(self) | None | Updates the GUI. Returns nothing |
| setTitle(self, title) | Title string | Writes the input in the title bar of the GUI |
| Function | **Input(s)** | **Output(s)** |
| clearWindow(self) | None | Clears the window. Returns nothing |
| drawTwoFieldsTwoButtonLayout(self, fieldLabels, buttonTexts, buttonCallbacks) | Input field text, button text, and button callbacks | Draws a screen with two user input fields and buttons, all with associated text. Returns nothing |
| drawNFieldsNButtonsOneDropDownLayout(self, dropDownLabelText, currentDropDownItem, dropDownOptions, fieldLabels, buttonTexts, buttonCallbacks) | Drop down menu label, value of the current drop-down option, the set of drop-down options, an array of input field texts, button text, and button callbacks | Draws a screen with ten user input fields, a drop-down menu, and buttons, all with associated texts. Returns nothing |
| displayLabelEntry(self, programMode, dropDownOptions, fieldLabels) | The program’s current state, the options for the drop-down menu, and the text labels to display on the input fields | Sets the label entries to their respective values. Returns nothing |
| getProgramMode(self) | None | Returns the program’s current state |
| displayErrorMessageLoginS(self, errorCode) | The relevant error code for the login screen | Displays a login error message |
| displayErrorMessageProgramS(self, errorCodeRate, errorCodeChamber) | The relevant error code parameters | Displays a programming error message |
| getEntryData(self) | None | Returns an array of the data inputted into the data fields of the current screen |
| setNEntryData(self, data) | Array of values to be set | Sets each value to the entries in the data array. Returns nothing |

#### Global Variables

|  |  |
| --- | --- |
| Variable(s) | Description |
| Variables From src.dcm\_constants | The file dcm\_constants.py contains the variables that hold the text to be displayed when drawing various screens (i.e. C\_LOGIN\_USERNAME\_LABEL = “Username”, C\_PROGRAM\_BUTTON\_TEXT = “Program”) |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| \_\_init\_\_() | Constructor function that is automatically called with the creation of every instance of a class. This function allows the class to initialize itself with the attributes/values it is given |
| update(self) | Updates the GUI and initializes it if it hasn’t been already |
| setTitle(self, title) | Sets the text to be shown in the title bar of the GUI if it has been initialized |
| clearWindow(self) | Clears all elements on the current screen by parsing through the list that is the output of the getEntryData(self) function |
| drawTwoFieldsTwoButtonLayout(self, fieldLabels, buttonTexts, buttonCallbacks) | Draws two user input fields and a button to the screen. fieldLabels is an array of labels to be used for the input fields. buttonTexts is an array of texts to be displayed in the button. buttonCallbacks is an array of button callback functions. These are drawn using the GUI Library (tkinter) |
| drawNFieldsNButtonsOneDropDownLayout(self, dropDownLabelText, currentDropDownItem, dropDownOptions, fieldLabels, buttonTexts, buttonCallbacks) | Draws ten user input fields and a button to the screen. dropDownLabelText is the label for dropDownMenu. currentDropDownItem is the value of the current dropDownOption. dropDownOptions is the set of dropDownOptions. fieldLabels is an array of labels to be used for the input fields. buttonTexts is an array of texts to be displayed in the button. buttonCallbacks is an array of button callback functions. These are drawn using the GUI Library (tkinter).  This method creates a tkinter variable and uses it to set the currentDropDownItem. It also uses a tkinter link function to change the dropdown. |
| displayLabelEntry(self, programMode, dropDownOptions, fieldLabels) | Displays the necessary label entry by comparing the inputs with the program mode. The functions goes through a series of “if” statements where the programMode is compared with the values in the dropDownOptions[] array. |
| getProgramMode(self) | Returns the program’s current mode |
| displayErrorMessageLoginS(self, errorCode) | Sets the label to be displayed with the error code text inside it. Returns nothing |
| displayErrorMessageProgramS(self, errorCodeRate, errorCodeChamber) | Sets the labels to be displayed with the error code texts within. Returns nothing |
| getEntryData(self) | Gets the information inputted into the data field and returns it in an array |
| setNEntryData(self, data) | Sets the values in the parameter array to be single strings that the user inputs using tk.Entry |

## Data Management

### User Account Manager

#### Description

The User Account Manager handles the creation of and logging in/out of Device Control-Monitor users. It also sets the devices programmable parameters: the upper/lower rate limits and the atrium/ventricle pulse amplitudes, pulse widths, sensing thresholds, and refractory periods.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| LoginData | \_\_init\_\_(self, p\_username, p\_password) |
| ProgammedData | \_\_init\_\_(self, p\_upperRateLim, p\_lowerRateLim, p\_atriumPulseAmp, p\_atriumPulseWidth, p\_atriumSensThres, p\_atriumRefracPeriod, p\_ventriclePulseAmp, p\_ventriclePulseWidth, p\_ventricleSensThres, p\_ventricleRefracPeriod) |
| DUAM | \_\_init\_\_(self)  getSessionState(self)  signInUser(self, p\_loginData)  signOut(self)  p\_makeAdminUser(self)  makeNewUser(self, p\_loginData, p\_adminPassword)  changeUserPassword(self, p\_username, p\_existingPassword, p\_newPassword)  validUser(self)  validNumUsers(self)  programRateLim(self, p\_upperRateLim, p\_lowerRateLim)  programAtriaPara(self, p\_atriumPulseAmp, p\_atriumPulseWidth, p\_atriumSensThres, p\_atriumRefracPeriod)  programVentriclePara(self, p\_ventriclePulseAmp, p\_ventriclePulseWidth, p\_ventricleSensThres, p\_ventricleRefracPeriod) |
| None | hash\_password(password)  verify\_password(stored\_password, provided\_password) |
| None | frange(start, stop, step) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Output(s) |
| ALL \_\_init\_\_() FUNCTIONS | Parameters for a class structure | Sets the values of the structure with the given inputs. Returns nothing |
| getSessionState(self) | None | Returns the state of the session (logged in/out) |
| signInUser(self, p\_loginData) | Username and password from input fields | If the credentials are valid, the user is logged in and a success code is returned. Else, a failure code of “invalid credentials” is returned if either the username or password are invalid |
| signOut(self) | None | Signs the user out of the DCM. Returns True |
| p\_makeAdminUser(self) | None | Creates the admin user in the database. Returns nothing |
| makeNewUser(self, p\_loginData, p\_adminPassword) | Login data (username & password) and admin password | If successful, it creates a user in the database with a username, password, and role and a success code is returned. Else, a failure code is returned denoting “missing permissions”, “too many users”, or “existing user” |
| changeUserPassword(self, p\_username. P\_existingPassword, p\_newPassword) | Username, current password, and the new password | If successful, changes the old password of a user with the new one given as input. Else, returns a failure code denoting “invalid credentials” if either the username or password are incorrect |
| validUser(self) | None | If logged out, returns False. Else, returns True |
| validNumUsers(self) | None | Returns False if the number of users in the database is greater than or equal to 10. Else, returns True |
| programRateLim(self, p\_upperRateLim, p\_lowerRateLim) | Input field data for the upper and lower rate limits | Sets the user’s data with the appropriate inputs. Returns nothing |
| Function | **Input(s)** | **Output(s)** |
| programAtriaPara(self, p\_atriumPulseAmp, p\_atriumPulseWidth, p\_atriumSensThres, p\_atriumRefracPeriod) | Input field data for the atrium’s pulse amplitude, pulse width, sensing threshold, and refractory period | Sets the user’s data with the appropriate inputs. Returns nothing |
| programVentriclePara(self, p\_ventriclePulseAmp, p\_ventriclePulseWidth, p\_ventricleSensThres, p\_ventricleRefracPeriod) | Input field data for the ventricle’s pulse amplitude, pulse width, sensing threshold, and refractory period | Sets the user’s data with the appropriate inputs. Returns nothing |
| hash\_password(password) | Password | Returns the hashed password |
| verify\_password(stored\_password, provided\_password) | A password in the database and a password from the user | Returns True if the inputs are the same. Else, False |
| frange(start, stop, step) | 3 float numbers | Returns a float number |

#### Global Variables

|  |  |
| --- | --- |
| Variable(s) | Description |
| Variables From src.dcm\_constants | The imported file dcm\_constants.py contains the variables that hold the text to be displayed when drawing various screens (i.e. C\_LOGIN\_USERNAME\_LABEL = “Username”, C\_PROGRAM\_BUTTON\_TEXT = “Program”) |
| Variables From DCMDatabase.dbpm | This imported file contains the User class which hold the username, password, and userRole. It also has the DBPM class (Database Peewee Manager) |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| ALL \_\_init\_\_() FUNCTIONS | For all classes this is a constructor function that is automatically called with the creation of every instance of a class. This function allows the class to initialize itself with the attributes/values it is given.  The init function for the DUAM class however also creates the admin user when the database is empty, and the admin user has never been created yet. |
| getSessionState(self) | Returns the current state of the user (logged out/in) |
| signInUser(self, p\_loginData) | Signs the user in. It checks that the user exists in the database and then fetches the user data to compare with the inputted data and decide whether or not to log the user in. |
| signOut(self) | Signs the user out |
| p\_makeAdminUser(self) | Adds the admin user to the database if it is not already there |
| Function | **Description** |
| makeNewUser(self, p\_loginData, p\_adminPassword) | Adds a new user and its login credentials to the database. Returns a failure code if there are too many users, if the user already exists, or if the user is invalid |
| changeUserPassword(self, p\_username. P\_existingPassword, p\_newPassword) | Changes a user’s stored password through the database manager after checking that the user exists and verifying the existing password |
| validUser(self) | Checks if the current user is valid by returning False if signed out or if the user object is None |
| validNumUsers(self) | Checks if the number of users in the data base is greater than or equal to 10. Returns False if this is the case |
| programRateLim(self, p\_upperRateLim, p\_lowerRateLim) | Sets the current user’s upper and lower rate limits in the database given the inputs |
| programAtriaPara(self, p\_atriumPulseAmp, p\_atriumPulseWidth, p\_atriumSensThres, p\_atriumRefracPeriod) | Sets the current user’s atrium pulse amplitude, pulse width, sensing threshold, and refractory period in the database given the inputs |
| programVentriclePara(self, p\_ventriclePulseAmp, p\_ventriclePulseWidth, p\_ventricleSensThres, p\_ventricleRefracPeriod) | Sets the current user’s ventricle pulse amplitude, pulse width, sensing threshold, and refractory period in the database given the inputs |
| hash\_password(password) | Hashes a password for storing |
| verify\_password(stored\_password, provided\_password) | Verifies a stored password against one provided by the user |
| frange(start, stop, step) | Performs the same function as the regular Range() function but with floating point numbers. If the inputs “stop” and “step” are null, then “start” is set to 0.0 and “step” to 1.0 |

### Database Manager

#### Description

The Database Manager handles the actual working details of storing, retrieving, and changing user information by communicating with the database.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| User | \_\_init\_\_(self, p\_username, p\_password, p\_userRole)  getUsername(self)  getPassword(self)  getRole(self) |
| UserProgramData | \_\_init\_\_(self, p\_upperRateLim, p\_lowerRateLim, p\_atriumPulseAmp, p\_atriumPulseWidth, p\_atriumSensThres, p\_atriumRefracPeriod, p\_ventriclePulseAmp, p\_ventriclePulseWidth, p\_ventricleSensThres, p\_ventricleRefracPeriod)  *There are 10 functions for getting all of these values and another 10 for setting these values after initialization. For simplicity’s sake, they will be listed next to each other.*  getLowerRateLimit(self) / setLowerRateLimit(self)  getUpperRateLimit(self) / setUpperRateLimit(self)  getAtrialAmplitude(self) / setAtrialAmplitude(self)  getAtrialPulseWidth(self) / setAtrialPulseWidth(self)  getAtrialSensingThreshold(self) / setAtrialSensingThreshold(self)  getAtrialRefractoryPeriod(self) / setAtrialRefractoryPeriod(self)  getVentricularAmplitude(self) / setVentricularAmplitude(self)  getVentricularPulseWidth(self) / setVentricularlPulseWidth(self)  getVentricularSensingThreshold(self) / setVentricularSensingThreshold(self)  getVentricularRefractoryPeriod(self) / setVentricularRefractoryPeriod(self) |
| Class | **Functions/Methods** |
| DBPM | \_\_init\_\_(self)  closeDatabase(self)  getDatabaseInstance(self)  createUser(self, p\_username, p\_password, p\_role)  userExists(self, p\_username)  getUserData(self, p\_username)  getNumUsers(self) |

#### Private Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| DBPM | p\_createDataTables(self) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Outputs(s) |
| ALL \_\_init\_\_() FUNCTIONS | Parameters for the class structure | Sets the values of the structure with the given inputs. Returns nothing |
| getUsername(self) | None | Returns the username of the user |
| getPassword(self) | None | Returns the password of the user |
| getRole(self) | None | Returns the role of the user (doctor, nurse, admin, etc.) |
| closeDatabase(self) | None | Returns with the method that closes the database |
| getDatabaseInstance(self) | None | Returns with the method that gets the database instance |
| createUser(self, p\_username, p\_password, p\_role, p\_data) | Username, password, and a role | Creates the user in the database and stores the data in the database. Returns nothing |
| userExists(self, p\_username) | Username | Returns True if a username is the database matches the input. Else, returns False |
| getUserData(self, p\_username) | Username | Returns a User class with username, password, and role if a username matching the input is found. Else, returns None |
| getNumUsers(self) | None | Returns the number of users in the database |
| Function | **Input(s)** | **Outputs(s)** |
| p\_createDataTables(self) | None | Creates the tables used in the application. Returns nothing |
| *For the set functions mentioned in the Public Functions table* | None | *Returns the value corresponding to the function name* |
| *For the get functions mentioned in the Public Functions table* | User input from a data field | *Sets the value from the input field corresponding to the field label and the function name. Returns nothing* |

#### Global Variables

|  |  |
| --- | --- |
| Function | Input(s) |
| Variables From src.dcm\_constants | The imported file dcm\_constants.py contains the variables that hold the text to be displayed when drawing various screens (i.e. C\_LOGIN\_USERNAME\_LABEL = “Username”, C\_PROGRAM\_BUTTON\_TEXT = “Program”) |
| database | SqliteDatabase structure with a given path, size, and keys |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| ALL \_\_init\_\_(self) FUNCTIONS | For all classes this is a constructor function that is automatically called with the creation of every instance of a class. This function allows the class to initialize itself with the attributes/values it is given.  The init function for the DBMP class also connects to the database and creates data tables. |
| getUsername(self) | Returns the username of the user |
| getPassword(self) | Returns the password of the user |
| getRole(self) | Returns the role of the user (doctor, nurse, admin, etc.) |
| closeDatabase(self) | Closes the database |
| getDatabaseInstance(self) | Returns the current database instance |
| createUser(self, p\_username, p\_password, p\_role, p\_data) | Creates a new user and data variable in the database. It stores the data from the input variable into their appropriate locations as well as the username, password, and role of the new user |
| userExists(self, p\_username) | Checks if a user exists by looping through users in the database and comparing usernames. Returns True or False |
| getUserData(self, p\_username) | Loops through user data in the database looking for a username matching the input. If one is found, returns a User class (with username, password, role, and data) |
| Function | **Description** |
| getNumUsers(self) | Loops through user data in the database and counts the number of users |
| p\_createDataTables(self) | Creates all tables used in the application |

### SQLite ORM Library

An ORM is an Object-Relational-Mapper. It acts as a layer between the SQLite database and the object-oriented code (database manager). The one used for this DCM is called Peewee and works with both Python and SQLite. This allows us to use the database without having to transform objects into a suitable format for it.

### SQLite Database

SQLite is a software library that provides a relational database management system (RDBMS). RDBMSes are a type of database manager that store data in a format using rows and columns.

## Pacemaker Communication

### Protocol

#### Description

The standard communication protocol is to send a start byte, followed by a command byte, followed by data of the form ‘3B2H2B5H2B’ where B represents an unsigned uint8 and H represents a uint16.

#### Example

To program the pacemaker parameters the DCM would send a start byte (0x16), then the pacemaker parameter programming command byte (0x55), then 21 bytes of parameter data.

### DCM Communication Controller

#### Description

This module acts as an abstraction layer for the DCM Serial Manager. It handles the communication between the DCM and the Pacemaker by opening/closing then connection and reading/writing the values stored on the Pacemaker.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| DCC | \_\_init\_\_(self)  deinit(self)  openConnection(self)  closeConnection(self)  programPacemaker(self, params)  getPacemakerData(self)  getElectrogram(self) |

#### Private Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| DCC | p\_prependDataWithStartCode(self, data)  p\_transmitData(self, data)  p\_convertToInts(self, number, numberOfBytes)  p\_convertPacemakerParamsToByteArray(self, pacemakerParams)  p\_sendEchoCommand(self) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Outputs(s) |
| \_\_init\_\_(self) | None | Initializes serialManager. Returns nothing |
| deinit(self) | None | Deinitializes serialManager. Returns nothing |
| openConnection(self) | None | Initializes serialManager. Returns nothing |
| closeConnection(self) | None | Deinitializes serialManager. Returns nothing |
| p\_prependDataWithStartCode(self, data) | The data to be sent to the Pacemaker | Returns the data prepended with the serial start byte |
| p\_transmitData(self, data) | The data to be sent to the Pacemaker | Sends data to the Pacemaker and prints a fail/success statement. Statements correspond with the possible returns of True, False, or nothing. |
| Function | **Input(s)** | **Outputs(s)** |
| programPacemaker(self, params) | Programmable parameters for the Pacemaker | Converts params into the byte array. Can return False |
| p\_convertToInts(self, number, numberOfBytes) | Number to be converted and the number of bytes | Returns the converted number in an array. |
| p\_convertPacemakerParamsToByteArray(self, pacemakerParams) | Programmable parameters for the Pacemaker | Returns a byte array of the Pacemaker parameters |
| p\_sendEchoCommand(self) | None | Returns the success of the data transmit; True or False |
| getPacemakerData(self) | None | Returns the parameters store in the Pacemaker in a byte array |
| getElectrogram(self) | None | Returns an array of electrogram values from the Pacemaker |

#### Global Variables

|  |  |
| --- | --- |
| Variable(s) | Description |
| Variables From src.dcm\_constants | The imported file dcm\_constants.py contains the variables that hold the text to be displayed when drawing various screens (i.e. C\_LOGIN\_USERNAME\_LABEL = “Username”, C\_PROGRAM\_BUTTON\_TEXT = “Program”) |
| Variables from common.datatypes | The imported file datatypes.py contains general variables used throughout the DCM. Most of these are the Pacemaker parameters and the various programming modes the Pacemaker can be in. |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| \_\_init\_\_(self) | Initializes serialManager. |
| deinit(self) | Deinitializes serialManager. |
| openConnection(self) | Opens the serial connection by initializing the serialManager |
| closeConnection(self) | Closes the serial connection by deinitializing the serialManager |
| p\_prependDataWithStartCode(self, data) | Prepends the serial communication start byte to the byte array. Returns that array |
| p\_transmitData(self, data) | Prepares the Pacemaker data using p\_prependDataWithStartCode(self, data)then sends it using the DCM Serial Manager |
| programPacemaker(self, params) | Sends the Pacemaker programmable parameters after converting them into a byte array |
| p\_convertToInts(self, number, numberOfBytes) | Converts the inputted number using bit shifting and logical operations |
| Function | **Description** |
| p\_convertPacemakerParamsToByteArray(self, pacemakerParams) | Converts the Pacemaker parameters into a byte array by using p\_convertToInts(self, number, numberOfBytes) and adding the returns from each function call to the array |
| p\_sendEchoCommand(self) | Sends an echo command to the Pacemaker using p\_transmitData(self, data) |
| getPacemakerData(self) | Gets the parameters stored on the Pacemaker in a byte array using the serialManager |
| getElectrogram(self) | Reads the electrogram from the Pacemaker and returns the values in an array |

### DCM Serial Manager

#### Description

This module handles the lower level details and the minute organization of communication necessary for the Pacemaker to send and receive data from the DCM.

#### Public Functions

|  |  |
| --- | --- |
| Class | Functions/Methods |
| DSM | \_\_init\_\_(self)  deinit(self)  write(self, data)  writeString(self, dataStr )  readUntil(self, expected)  readLine(self)  getSerialPort(self)  checkSerialPort(self) |

#### Black Box Behaviour

|  |  |  |
| --- | --- | --- |
| Function | Input(s) | Outputs(s) |
| \_\_init\_\_(self) | None | Initializes serialManager. Can print “Error Opening Com Port” if an error occurs. Returns nothing |
| deinit(self) | None | Deinitializes the serialManager. Returns nothing |
| Function | **Input(s)** | **Outputs(s)** |
| write(self, data) | Input byte array to write | Prints the serial data and its length. Writes this data to the serial port. Can return a failure code |
| writeString(self, dataStr) | Input string to write | Writes the byte array to the serial port. Returns nothing |
| readUntil(self, expected) | Expected end character | Reads the serial data until the expected character. Returns this data; can also return a failure code |
| read (self, n) | Number of bytes to read | Reads n number of bytes of serial data. Returns this data; can also return a failure code. |
| readLine(self) | None | Reads and returns one line of serial data. Can return a failure code |
| getSerialPort(self) | None | Returns the name of the serial port. Can return a failure code |
| checkSerialPort(self) | None | Returns true if the serial port is available. Else, returns a failure code |

#### Global Variables

|  |  |
| --- | --- |
| Variable(s) | Description |
| Variables From src.dcm\_constants | The imported file dcm\_constants.py contains the variables that hold the text to be displayed when drawing various screens (i.e. C\_LOGIN\_USERNAME\_LABEL = “Username”, C\_PROGRAM\_BUTTON\_TEXT = “Program”) |
| Variables from common.failCodes | The imported file failCodes.py contains the enumeration for the possible errors that may occur such as there being too many users, invalid rate input, etc… |

#### Function Descriptions

|  |  |
| --- | --- |
| Function | Description |
| \_\_init\_\_(self) | Initializes the serialManager or prints an error message if it cannot for some reason |
| deinit(self) | Deinitializes the serialManager by closing the serial port |
| write(self, data) | Writes the data to the serial port and prints it along with its length to the screen. Returns a failure code if the port is not open/available |
| writeString(self, dataStr) | Writes the data string to the serial port using write(self, data) |
| Function | **Description** |
| readUntil(self, expected) | Reads the serial data until the expected character and returns that data. Returns a failure code if the serial port is not open/available |
| read (self, n) | Reads n number of bytes of serial data. Returns a failure code if the serial port is not open/available |
| readLine(self) | Reads one line of serial data using readUntil(self, expected) with no input. Returns a failure code if the serial port is not open/available |
| getSerialPort(self) | Returns the name of the serial port. Returns a failure code if the serial port is not open/available |
| checkSerialPort(self) | Checks to see if the serial port is open and available. Returns a failure code if the serial port is not open/available |

# Appendix

## Module Flowchart

