SE 3KO4 - L01 G7

Assignment 2 - DCM Software

Carlos Capili

Raeed Hassan

Shaqeeb Momen

Aaron Pinto

Udeep Shah

Table of Contents

1.0 Requirements

1.1 Current Requirements

The current requirements for the DCM software are to properly implement all the necessary components required to monitor and manage a pacemaker.

1.1.1 Welcome Screen

The welcome screen provides an interface to register new users, or login as an existing user. A maximum of 10 users can be stored locally.

1.1.2 User Interface Essential Aspects

The user interface implements essential aspects necessary for the interface. These are:

- Capable of utilizing and managing windows for display of text and graphics
- Capable of processing user positioning and input buttons
- Capable of displaying all programmable parameters for review and modification
- Capable of visually indicating when the DCM and the device are communicating
- Capable of visually indicating when a different PACEMAKER device is approached than was previously interrogated

1.1.3 DCM Utility Functions

The DCM implements utility functions in the user interface to allow the user to access essential DCM utilities.

Utility Function	**Functionality**
About	Displays the following information: - Application model number - Application software revision - DCM serial number

	- Institution name
New Patient	Allow a new device to be interrogated without exiting the software application
Quit	Ends current telemetry session

Table 1.1.3: DCM Utility Functions

1.1.4 Pacing Mode Interfaces

The DCM Implements interfaces for pacemaker pacing modes. The pacing modes that are implemented are: AOO, AOOR, AAI, AAIR, VOO, VOOR, VVI, VVIR, DOO, DOOR.

1.1.5 Printed Reports

Printed reports allow the user to view pacemaker parameters, and pacemaker telemetry data in a straightforward manner. All printed reports display header information at the top of the report, including:

- Application model number
- Application software revision
- DCM serial number
- Device serial number
- Institution name
- Report name
- Date and time of report printing

Printed Report	**Information Presented**
Bradycardia Parameters	Programmable parameters for selected pacing mode
Electrogram Report	Atrial and ventricular lead data from pacemaker

Table 1.1.5: Printed Reports

1.1.6 Programmable Parameters

Programmable parameters are stored locally and validated with minimum and maximum increments, and valid increments for each parameter.

Programmable Parameter	**Units**	**Nominal Value**	**Minimum Value**	**Maximum Value**	**Increment**
Lower Rate Limit	ppm	60	30	175	5
Upper Rate Limit	ppm	120	50	175	5
Maximum Sensor Rate	ppm	120	50	175	5
Fixed AV Delay	ms	150	70	300	10
Atrial Amplitude	V	5	0	5	0.1
Atrial Pulse Width	ms	1	1	30	1
Atrial Sensitivity	V	3	0	5	0.1
Ventricular Amplitude	V	5	0	5	0.1
Ventricular Pulse Width	ms	1	1	30	1
Ventricular Sensitivity	V	3	0	5	0.1
ARP	ms	250	150	500	10
VRP	ms	320	150	500	10

PVARP	ms	250	150	500	10
Activity Threshold	N/A	Med	N/A	N/A	V-Low, Low, Med-Low, Med, Med-High, High, V-High
Reaction Time	sec	30	10	50	10
Response Factor	N/A	8	1	16	1
Recovery Time	min	5	2	16	1

Table 1.1.6: Programmable Parameters

1.1.7 Real-time Electrograms

Real-time electrograms display telemetry data received about the pacemaker's atrial and ventricular leads via serial communication with the pacemaker. The telemetry data is displayed on the DCM window as two scrolling graphs, one for atrial telemetry data and one for ventricular telemetry data. The user has the option of selecting which electrograms will be displayed on the DCM window (neither electrogram, one electrogram, or both electrograms).

The user is able to generate a printed report for the electrogram, that allows the user to view a screenshot of the telemetry data for when the report is generated, and provides the user with the option to export the telemetry data.

1.1.8 Serial Communication

The DCM implements serial communication with the pacemaker to send programmable parameters, initiate pacing on the pacemaker, and receive telemetry data from the pacemaker.

1.2 Expected Changes to Requirements

There are no expected changes to the broader systems of the DCM, all planned functions and components of the DCM have been implemented and verified. Support for additional pacing modes and programmable parameters may be required in future revisions, and the DCM modules have been developed to allow for simple integration of new pacing modes and programmable parameters. Potential new pacing modes are:

- Full dual sensing, pacing and adaptive operation (DDDR)

2.0 Software Model

2.1 General Design Decisions

The DCM and all its modules were programmed in Python 3. The decision to use Python was largely due to the programmers' familiarity with the Python language and Python libraries that could be used to handle different DCM systems. The release of Python decided was Python 3.7.9 an earlier release was selected to ensure compatibility with potential libraries.

The GUI was designed using PyQt, a Python library for Qt, a widget-toolkit for designing and creating graphical user interfaces. It was initially decided that Python's standard Tkinter package and it's associated tools would be used, however, it was later decided that a tool that would allow us to create our user interface graphically instead of programmatically would ease our workflow. The PyQt5 library was ultimately decided upon due to the ability to automatically generate the code for the graphical user interface from design files created using Qt Designer, a tool to graphically design GUI elements for Qt.

Serial communication between the DCM and Pacemaker and device recognition is handled by the pyserial library due to the programmers' familiarity with the library.
Graphs in the DCM are handled by the PyQtGraph library. The library was chosen due to its straightforward integration with the PyQt library and its ability to easily create interactive plots.

2.2 GUI

The DCM GUI implements all the user interface elements for the DCM. Most UI (user interface) elements are created in Qt Designer, and code for these elements are automatically generated using tools from the PyQt library. The individual GUI components are connected through code using the PyQt library.

2.2.1 Welcome Screen

The welcome window is the window the user is greeted by when the DCM program is launched, allowing the user to register users or login into the DCM. The Welcome Screen is a QStackedWidget, a Qt class that contains multiple widgets but only displays one at a time. The QStackedWidget class was chosen to allow all modules of the welcome screen to be accessible in a single window, while still allowing individual modules to behave independently. Additional modules can easily be added as new widgets without affecting implemented modules.

2.2.1.1 Welcome

The welcome window is the window the user is greeted by when the DCM program is launched, allowing the user to register users or login into the DCM. The Welcome Screen is a QStackedWidget, a Qt class that contains multiple widgets but only displays one at a time. The QStackedWidget class was chosen to allow all modules of the welcome screen to be accessible in a single window, while still allowing individual modules to behave independently. Additional modules can easily be added as new widgets without affecting implemented modules.

Push Button	**Function**
welcome_ui.reg_btn	welcome_gui.setCurrentIndex(1)
welcome_ui.log_btn	welcome_gui.setCurrentIndex(2)

Table 2.2.1.1: Welcome connections

2.2.1.2 Register

The register screen can only be navigated to from the welcome screen. The register screen gives the user the ability to register a new user using a username and password, or return to the welcome screen. The register widget is a QWidget with internal functions that cannot be accessed by the other widgets. The widget's functions are accepting user inputs in the username and password fields, authorizing the user inputs, or directing the user to another widget. The register button will call the authentication handler's register function with the text in the username and password fields as inputs. The back button will return the user to the welcome screen.

Push Button	Function
welcome_ui.reg_back_btn	welcome_gui.setCurrentIndex(0)
welcome_ui.reg_submit_btn	auth.register

Table 2.2.1.2: Register connections

2.2.1.3 Login

The login screen can only be navigated to from the welcome screen. The login screen gives the user the ability to login as an existing user using a username and password, or return to the welcome screen. The login widget is a QWidget with internal functions that cannot be accessed by the other widgets. The widget's functions are accepting user inputs in the username and password fields, authorizing the user inputs, or directing the user to another widget. The login button will call the authentication handler's login function with the text in the username and password fields as inputs. The back button will return the user to the welcome screen.

Push Button	Function
welcome_ui.log_back_btn	welcome_gui.setCurrentIndex(0)
welcome_ui.log_submit_btn	auth.login

Table 2.2.1.3: Login connections

2.2.2 DCM Main Window

The main window is the window which contains all the DCM functions. The main window has three main functions: presenting important information about the pacemaker (DCM's connection status with the pacemaker, the pacemaker's pacing mode, graphical telemetry data from the pacemaker), providing the user the ability to change pacing modes, and allowing the user to access the DCM's functions. The main window is a QMainWindow, a Qt class intended for main window management. The class was chosen due to its ability to add status bars, which will be used to display information about the DCM's connection with the pacemaker. The main window consists of three main sections: real-time electrograms, pacing mode interfaces, and DCM functions. While there are some interactions between these sections (particularly the pacing mode interfaces and the modules in the DCM functions), they are largely contained in their own modules that can function independently from each other. Future modules can be added to any section without affecting implemented modules (with the exception of modules that do not have any defined behaviour for additional pacing modes).

The real-time electrograms show real-time telemetry data from the atrial and ventricular leads on the pacemaker. Checkboxes for atrial and ventricular leads allow the user to select which graphs are shown.

The pacing mode interfaces allow the user to select the operating pacing mode of the pacemaker. Pacing mode selection is implemented using the QRadioGroup class, containing instances of the QRadioButton class for each pacing mode. The QRadioGroup class only allows one radio button to be switched on at a time, ensuring only a single pacing mode is selected at all times. Functions in other modules that rely on the pacing mode can rely on the functions of the QRadioGroup class, ensuring that the individual radio buttons cannot be altered.

The DCM's functions are operated through push buttons on the main window. The push buttons will either call a function from a handler (new patient), or direct the user to another window. Relevant handler functions are called in these new windows depending on user selected options presented in these windows.

Additional pacing modes are expected to be added in future revisions of the software, and behaviour for these pacing modes have to be accounted for in modules that require the pacing mode (or implement behaviour to handle unknown pacing modes).

UI Element	Function
dcm_ui.pace_btn	conn.send_data_to_pacemaker
dcm_ui.about_btn	about_gui.exec
dcm_ui.parameters_btn	params_gui.exec
dcm_ui.reports_btn	reports_gui.exec
dcm_ui.new_patient_btn	conn.register_device
dcm_ui.quit_btn	dcm_gui.close
dcm_ui.atrial_box	graphs.atri_vis
dcm_ui.vent_box	graphs.vent_vis

Table 2.2.2: DCM Main Window connections

2.2.2.1 Electrograms

The graphs use the PlotWidget class from PyQtGraph, a QGraphicsWidget with additional functions for plotting. The data in the plots is randomly generated and the plotted when the DCM main window is opened. The random data is currently a placeholder until the ability to receive telemetry data from the pacemaker is implemented. All functions of the plots are handled by graphs handler, including showing and hiding the graphs with the lead checkboxes.

2.2.1.2 Reports

The reports window provides the user with the option to view one of the following printed reports: the electrogram report, and the bradycardia parameters report. The reports window is a QDialog window that does not interact with other modules in the DCM. The window contains three push buttons that call functions in the reports handler with the pacing mode of the DCM as an input. The reports are shown in a popup window that is generated by the reports handler. Future reports can be implemented by adding additional push buttons to the window, then implementing the report generation in the reports handler.

Push Button	**Function**
reports_ui.egram_btn	self.show_egram_report
reports_ui.brady_btn	reports.generate_brady

Table 2.2.2.2: Reports connections

2.2.1.3 Parameters

The parameters window displays all the DCM programmable parameters to the user and allows the user to modify them and save their changes. The parameters window contains a QTableWidget, a widget providing an item-based table view. The horizontal headers contain a list of the programmable parameters that have been implemented, while the vertical headers contain the current values of each parameter. The current values in the cells of the table are pre-programmed to the nominal values of each parameter, and are used to determine the default values for each parameter in the code.

The current parameter values are programmed into the table, however they are immediately replaced by any changes that have been saved to the parameters file. The "Reset to defaults" button will reset the parameter files values to the default values pre-programmed into the UI. These functions are all handled by the parameters handler, with the push buttons calling functions within the handler.

Modules that rely on the programmable parameters will retrieve them from the parameters file through the parameters handler, the parameters window does not directly interact with any modules other than the parameters handler.

Input validation and display of the programmable parameters are done using Qt objects. Depending on the increment required for the programmable parameter, a QSpinBox, QDoubleSpinBox, or QComboBox is used. For numeric parameters, a QSpinBox or QDoubleSpinBox is used depending on the increment (QSpinBox for integer increments, QDoubleSpinBox for decimal increments), allowing the user to change the value of the parameter to any valid increment within pre-programmed minimum and maximum values. For non-numeric parameters, a QComboBox displays all possible values and allows the user to select one value.

The parameters window will only show programmable parameters relevant to the selected pacing mode, and all other programmable parameters will be hidden.

Any future parameters will need to be added to the parameters handler with, with an appropriate object with the necessary parameters. Additionally, it will need to be pre-programmed into the reports window with the nominal value for the parameter.

Programmable Parameter	**Object**
Lower Rate Limit	QSpinBox
Upper Rate Limit	QSpinBox
Maximum Sensor Rate	QSpinBox
Fixed AV Delay	QSpinBox
Atrial Amplitude	QDoubleSpinBox
Atrial Pulse Width	QSpinBox
Atrial Sensitivity	QDoubleSpinBox

Ventricular Amplitude	QDoubleSpinBox
Ventricular Pulse Width	QSpinBox
Ventricular Sensitivity	QDoubleSpinBox
ARP	QSpinBox
VRP	QSpinBox
PVARP	QSpinBox
Activity Threshold	QComboBox
Reaction Time	QSpinBox
Response Factor	QSpinBox
Recovery Time	QSpinBox

Table 2.2.1.3: Programmable Parameters

2.2.1.4 About

The about window displays the application model number, application software revision, DCM serial number and institution name. The About window contains a QTableWidget. The headers and cells of the table are set to the appropriate values and made unselectable, making the user unable to modify the cells' values. The widget only shows pre-programmed values and does not interact with any other modules in the DCM.

2.2.1.5 New Patient

The New Patient push button calls the register_device function in the connection handler. The connection handler will attempt to register the device and generate a message window depending on the results of the function.

2.2.1.6 Quit

The Quit push button closes the DCM GUI and exits the program.

2.2.1.7 Animated Status Bar

The animated status bar is a status bar for displaying the connection status between the DCM and the pacemaker with some integrated animations that depend on the connection status. The animated status bar is an AnimatedStatusBar, a class that extends the QStatusBar class from Qt with additional functions that interact with the connection handler. The AnimatedStatusBar's handle_conn_anim function is called whenever the state changes in the connection handler (the AnimatedStatusBar monitors a signal for state changes), and the function will display a programmed message and animation depending on the connection state.

2.2.1.8 Delegates

The delegates are extensions/subclasses of QStyledItemDelegates which are used in the parameters table GUI to constrain the possible values of each parameter. Each delegate for the corresponding parameter is applied to the entire table row, however the delegates have checks to only modify the column in the table with the param values. There are three types of delegates, to cover all the possible parameter types, ints, floats and strings. The delegates are essentially wrappers of the corresponding object listed above, in Table 2.2.1.3, so that we can replace the default text fields in QTableWidgets with the custom spin/combobox objects.

2.3 Handlers

The handlers are the backend of the program, where all safety-critical functions from the GUI are implemented. Each handler is completely independent from each other, but they can interact through passing in function arguments. For example with the reports and parameters handlers, the reports are generated from the input passed into each method which comes from the parameters handler. All private variables and functions are prefixed by an underscore (_).

2.3.1 Authentication

The authentication handler handles the authentication process for the user.

The authentication handler contains three public functions.

Method	**Inputs** variable (type)	**Description**
constructor	on_success (Callable[[str], None])	Initializes the credential store, which stores the user credentials while the program is running.
login	username (str) password (str)	Compares username and password inputs to existing credentials. Calls _on_success if the credentials match.
register	username (str) password (str)	Checks if the username exists or there are already 10 users registered. If neither are true it updates the credential store, writes it to disk, and calls _on_success.

Table 2.3.1.1: Authentication Public Functions

The authentication handler contains two state variables.

State Variable	**Type**	**Description**
_cred_store	Dict[str, str]	A dictionary that stores user credentials, where keys are usernames and values are passwords, while the program is running.
_on_success	Callable[[str], None]	The function to call when the authentication (register/login) process is successful.

Table 2.3.1.2: Authentication State Variables

The authentication handler contains one private function.

Method	**Inputs** variable (type)	**Description**
_show_alert	msg (str)	Displays the given message in an error alert box which will appear on top of the current window.

Table 2.3.1.3: Authentication Private Functions

2.3.2 Serial

The serial handler handles the serial communication between the DCM and the pacemaker. This includes sending the programmable parameters to the pacemaker, and receiving telemetry data from the pacemaker. It extends the PyQt5 QThread class in order to handle the communication in a separate thread.

The serial handler contains six public functions.

Method	**Inputs** variable (type)	**Description**
constructor	-	Calls the constructor of its superclass (QThread) and initializes all the state variables.
run	-	Gets called when the thread starts, overrides the method in QThread. It sets _running to True, then loops until the program is closed. Each loop, it checks if the serial connection with the pacemaker is open, and if it is it will write the params/_REQUEST_ECG bytes to the pacemaker. Then it will wait for a response, parse the response, and if we've received ECG data, unpack the bytes into atri/vent data and emit the ecg_data_update signal. If instead we've received params back, it will verify those params and display a success or fail verification message. If the serial connection with the pacemaker is not open, it will check if the Serial instance has a valid port and if it does it will try to open it. Otherwise it will sleep for 1 second.

stop	-	Stops the thread.
start_serial_comm	port (str)	Sets the serial connection port to that of the pacemaker, and clears the buffer.
stop_serial_comm	port (str)	Safely closes the serial connection and clears the port.
send_params_to_pacemaker	params_to_send (Dict[str, Union[int, float]])	Updates the bytes of the parameters to send to the pacemaker, and enables the send flag.

Table 2.3.2.1: Serial Public Functions

The serial handler contains nine state variables.

State Variable	**Type**	**Description**
_running	bool	Controls the while loop in the thread.
_buf	bytearray	A buffer that stores read bytes from the serial connection.
_conn	Serial	The pyserial Serial connection instance.
_num_bytes_to_read	int	The number of bytes to read and return per response
_sent_data	bytes	The bytes of the sent parameters. Used in verification of the received params.
_send_params	bool	A flag that determines whether we should send params to the pacemaker or not. Set to True by a Pace Now button press and set to False after we send the params.
_lock	Lock	A threading lock used to prevent concurrent accessing/modification of variables.

ecg_data_update	pyqtSignal(tuple, tuple)	A signal that's emitted every time we receive ECG data, containing the atrial and ventricular data.
params_received	pyqtSignal(bool, str)	A signal that's emitted upon pacemaker param verification, which is used to display the verification message and contains the result of the verification (True/False, success/fail) and the message to display..

Table 2.3.2.2: Serial State Variables

The serial handler contains three private functions.

Method	**Inputs** variable (type)	**Description**
_readline	-	Reads the output stream of the pacemaker and returns a bytearray of size _num_bytes_to_read. It uses non-blocking reading so it stores the read values in a buffer and checks the length of the buffer every cycle, to determine if sufficient bytes have been read.
_try_to_open_port	-	Attempts to safely open the serial connection with the pacemaker.
_verify_params	received_params (bytes)	Verifies that the params sent to the pacemaker are the same as the ones received back, by checking for equality. Emits the params_received signal with values depending on the result of the verification.

Table 2.3.2.3: Serial Private Functions

The data types we used to represent each parameter are the smallest possible type that we could use, that contained every possible value in the range needed for each parameter, and that was supported by both Python 3.7.9 and MATLAB/Simulink R2020a.

PARAM NAME	DATA TYPE (BYTES)
Mode (enumerated)	u_int_8 (1) -> 0:AOO, 1:AAI, 2:AOOR, 3: AAIR, 4:VOO, 5:VVI, 6:VOOR, 7:VVIR 8:DOO, 9:DOOR
Lower Rate Limit	u_int_8 (1)
Upper Rate Limit	u_int_8 (1)
Atrial Amplitude	single (4)
Atrial Pulse Width	u_int_8 (1)
Atrial Sensitivity	single (4)
Ventricular Amplitude	single (4)
Ventricular Pulse Width	u_int_8 (1)
Ventricular Sensitivity	single (4)
VRP	u_int_16 (2)
ARP	u_int_16 (2)
PVARP	u_int_16 (2)
Fixed AV Delay	u_int_16 (2)
Maximum Sensor Rate	u_int_8 (1)
Reaction Time	u_int_8 (1)
Response Factor	u_int_8 (1)
Recovery Time	u_int_8 (1)
Activity Threshold	u_int_8 (1) -> 0:V-Low, 1:Low, 2:Med-Low, 3: Med, 4:Med-High, 5:High, 6:V-High

2.3.3 Connection

The connection handler handles the serial connection between the DCM and pacemaker, and notifies the user of the current connection status. It extends the PyQt5 QThread class in order to handle the pacemaker connectivity in a separate thread.

The connection handler contains five public functions.

Method	**Inputs** variable (type)	**Description**
constructor	-	Calls the constructor of its superclass (QThread), initializes all the state variables and starts the serial handler thread.
run	-	Gets called when the thread starts, overrides the method in the QThread class. It sets _running to True, emits an initial state change signal and then loops until the program is closed, calling the _update_state method every 10ms approximately.
stop	-	Stops the thread and stops the serial handler thread.
register_device	-	Called when the New Patient button is pressed. It registers the pacemaker if the current state is CONNECTED, otherwise it displays an alert depending on other conditions. For example, it will display "Please plug in a pacemaker!" when the New Patient button is pressed and there are no pacemakers plugged into the PC.
send_data_to_pacemaker	params (Dict[str, Union[int, float]])	Called when the Pace Now button is pressed. It calls the serial handler send_params_to_pacemaker method if the current state is REGISTERED, otherwise it displays an alert depending on other conditions. For example, it will display "Please plug in a pacemaker!" when the Pace Now button is pressed and there are no pacemakers plugged into the PC.

Table 2.3.3.1: Connection Public Functions

The connection handler contains ten state variables.

State Variable	**Type**	**Description**

_running	bool	Controls the while loop in the thread.
_device	ListPortInfo	A pyserial ListPortInfo variable that stores the info of the connected pacemaker.
_devices	List[ListPortInfo]	Contains all the connected COM port devices.
_old_devices	List[ListPortInfo]	The same as devices but one cycle behind. It is used for figuring out which devices were added and removed between cycles, using the difference between the two lists.
_first_serial_num	str	Contains the serial number of the first pacemaker plugged into the PC, for auto-registration purposes.
_current_state	PacemakerState	Holds the current PacemakerState state.
_prev_state	PacemakerState	The same as current_state but one cycle behind and is used for detecting state transitions inside the current_state, after they have occurred.
_wanted_state	PacemakerState	The same as well but one cycle ahead of current_state.
serial	_SerialHandler	The _SerialHandler instance.
connect_status_change	pyqtSignal(PacemakerState, str)	A thread-safe signal that is used to inform the animated status bar when the connection status has changed. The str is the serial number and/or a message to display.

Table 2.3.3.2: Connection State Variables

The connection handler contains four private functions.

Method	Inputs	Description

	variable (type)	
_update_state	-	Contains a state machine for the pacemaker connection state. Detailed description below.
_handle_removed_device	removed (List[ListPortInfo])	Checks if a removed device matches the connected pacemaker's serial number. If it does, it sets the wanted state to NOT_CONNECTED, emits the connect_status_change signal and resets device to its initial value.
_show_alert	msg (str)	Displays msg on an information alert box that appears on top of the current window.
_filter_devices	data (List[ListPortInfo])	Filters data by Vendor ID and Product ID so that we only connect to pacemaker devices, and then it returns that filtered list.

Table 2.3.3.3: Connection Private Functions

_update_state was implemented like this because it offers us many benefits such as cleaner, easier to read code, ensuring that a pacemaker gets registered only once, handling multiple pacemakers being plugged into the same computer, and handling the New Patient button presses in a much simpler way.

Every 10ms cycle, the method gets the filtered list of connected COM port devices, and finds the devices that were added or removed between cycles, using the difference between devices and old_devices. Then it updates the current_state if it's not aligned with the wanted_state.

If we're not connected to a pacemaker, we figure out if any new devices were added, and then set the wanted_state accordingly, based on the state table below. If we are connected to a pacemaker, regardless of whether or not it is registered, we call _handle_removed_device to check if any devices were removed, and the transition state accordingly (refer to the table below).

On state transitions, we emit the connect_status_change signal with the corresponding current or wanted state, depending on if the emit happens before or after the current_state is updated.

At the end of the method, we update the variables that store previous cycle information, namely prev_state and old_devices. As you can see below in Table 2.3.3.4, the state transitions are straightforward.

		Previous State		
Event	Condition	NOT_CONNECTED	CONNECTED	REGISTERED
_update_state	len(added) > 0 and ((self.firstserialnum == "") or (self.firstserialnum == self.device. serialnum))	REGISTERED	-	-
	len(added) >0 and ! ((self.firstserialnum == "") or (self.firstserialnum == self.device. serialnum))	CONNECTED	-	-
register_ device	-	-	REGISTERED	-
handle removed_ device	len(removed) > 0 and (self.device. serialnumber == removed[0]. serialnumber)	-	NOT_CONNECTED	NOT_CONNECTED

Table 2.3.3.4: Connection Stateflow

2.3.4 Graphs

The graphs handler handles displaying the telemetry data on the PyQtGraph graphs in the DCM main window.

The graph handler contains four public functions.

Method	**Inputs** variable (type)	**Description**
constructor	atri_plot (PlotWidget), vent_plot (PlotWidget), data_size (int)	Updates the properties of the PyQtGraph PlotWidgets, initializes all the state variables and plots the initial data with length data_size.
update_data	atri_data (tuple) vent_data (tuple)	Updates and plots new received data. It shifts the _atri_data and _vent_data lists by the length of the new data, and adds the new data to the end of the lists, keeping the overall length of the plot data the same.
atri_vis	show (bool)	Shows or hides the atrial plot data item on the graphs, depending on the input, show.
vent_vis	show (bool)	Shows or hides the ventricular plot data item on the graphs, depending on the input, show.

Table 2.3.4.1: Graphs Public Functions

The graph handler contains four state variables.

State Variable	**Type**	**Description**
_atri_data	ndarray	Stores the sensed atrial lead data as a 1D array of length data_size.
_vent_data	ndarray	Stores the sensed ventricular lead data as a 1D array of length data_size.
_atri_plot	PlotDataItem	The PyQtGraph PlotDataItem for the atrial graph.
_vent_plot	PlotDataItem	The PyQtGraph PlotDataItem for the ventricular graph.

Table 2.3.4.2: Graphs State Variables

The graph handler contains one private function.

Method	**Inputs** variable (type)	**Description**
_plot_data	-	Plot the sense data on the graphs, by setting the data of the corresponding PlotDataItem.

Table 2.3.4.3: Graphs Private Functions

2.3.5 Parameters

The parameters handler handles the changing, displaying and validating the programmable parameters.

The parameters handler contains seven public functions..

Method	**Inputs** variable (type)	**Description**
constructor	table (QTableWidget)	table is used to retrieve and update values from the GUI, so they are only hard-coded once. The constructor also initializes all the state variables, and updates the _default_params_store and _units with the GUI defaults from table. It also tries and optionally loads existing parameters from the parameters file.
update_params_on_user_auth	username (str)	Updates the parameter values to the user-specific ones based on the user that is authenticated. It updates the _username and _user_params_store variables and the params GUI.
confirm		Called when the confirm button is clicked. Updates the param store and writes the values to file.
reset		Called when the reset button is clicked. Prompts user for

		confirmation and if yes, updates the user param store to the GUI defaults, writes the values to file and updates the params GUI.
update_row_visibility	pacing_mode (str)	Updates which rows/params are visible in the params table GUI based on the pacing mode. Gets the list of params to show from the state variable and then iterates through the table, showing and hiding each row, depending on if it exists in the params_to_show list.
filter_params	pace_mode (str)	Returns a pacing mode dependent dict of parameters with the names as keys, and param values with units as values.
get_params	pace_mode (int)	Returns a dict of all the user parameters including the pacing mode, casted to their respective data type, where the keys are the param name and the values are the cast param value.

Table 2.3.5.1: Parameters Public Functions

The parameters handler contains eight state variables.

State Variable	Type	Description
_table	QTableWidget	Stores the QTableWidget table, from the constructor.
_params_per_mode	Dict[str, List[str]]	A dict of parameters per pacing mode, where the pacing mode is the key and the values are the params required by that pacing mode.
_objects_per_param	Dict[str, Union[SpinBoxDelegate, DoubleSpinBoxDelegate, ComboBoxDelegate]]	A dict containing the object delegate for each parameter. The object delegate handles the range, increment and unit for each param. Keys are the param name, values are

		an instance of a SpinBoxDelegate, DoubleSpinBoxDelegate, or ComboBoxDelegate.
_default_params_store	Dict[str, str]	A dict containing the GUI default param values. Keys are the parameter name, values are the param value.
_units	Dict[str, str]	A dict containing the units for each param. Keys are the parameter name, values are the param units.
_user_params_store	Dict[str, str]	A dict that stores the parameters for a specific user (stored in _username). Keys are the parameter name, values are the param value.
_username	str	Stores the authenticated user's username.
_params_store	Dict[str, Dict[str, str]]	A dict that stores the per user params. Keys are usernames, values are equivalent to _user_params_store.

Table 2.3.5.2: Parameters State Variables

The parameters handler contains two private functions.

Method	**Inputs** variable (type)	**Description**
update params_file	-	Write the params store to a file, creating a new one if it doesn't exist. The file name is stored in the constant _PARAMETERS_FILE_PATH.
update params_gui	-	Updates the parameters GUI table with the values from the params store

Table 2.3.5.3: Parameters Private Functions

2.3.6 Reports

The reports handler handles generating and displaying all the reports for the DCM window.

The reports handler contains four public functions.

Method	**Inputs** variable (type)	**Description**
constructor	egram_report_ui (Ui_Dialog)	Initializes all the state variables.
generate_egram	header (Dict[str, str]), atri_snap (QPixmap), vent_snap (QPixmap)	Handles the generation and presentation of the electrogram report. It sets the report name to “Electrogram” and the report date/time to the current date/time. It also sets the report header info from the header dict, and then displays the atrial and ventricular plots from the passed in pixmaps.
generate_brady	header (Dict[str, str]) params (Dict[str, str]))	Handles the generation and presentation of the bradycardia report. It sets the report name to “Bradycardia” and the report date/time to the current date/time. It also sets the report header info from the header dict, and the report parameter values from the passed in params dict..
export_pdf	widget (QDialog)	Export the electrogram report as a pdf, for printing. The function gets a user-specified file path through a file dialog, and if the file name exists, adds a .pdf suffix if it doesn't exist, sets up the QPrinter and QPainter to create the pdf file, scales the page to fit the input electrogram report widget, dumps the contents of the electrogram report widget to the pdf, and then writes it to disk.

Table 2.3.6.1: Reports Public Functions

The reports handler contains two state variables.

State Variable	**Type**	**Description**
_egram_report_ui	Ui_Dialog (auto-generated from GUI)	Stores the electrogram report GUI.
_report_gen_time	str	Stores the last generated report time, and is updated every time a new report is generated. It is used for specifying the file name of the electrogram report pdf.

Table 2.3.6.2: Reports State Variables

The reports handler contains three private functions.

Method	**Inputs** variable (type)	**Description**
_format_params	params (Dict[str, str])	Formats the dictionary of parameters as a string, aligned to the colon, with each parameter name/value on a new line. No param name is greater than 23 characters so we do left and right alignments in a 25 character long block for even spacing.
_plain_format_params	params (Dict[str, str])	Formats the dictionary of parameters as a string, with each parameter name/value on a new line.
_show_report	report (str)	Shows a customized message box with the specified text. The font is set to "Consolas" because it is a monospaced font.

Table 2.3.6.3: Reports Private Functions

3.0 Testing

All testing was done through the DCM user interface.

3.1 Welcome Screen

<u>Window</u>	<u>Test</u>	<u>Test Case</u>	<u>Expected Outcome</u>	<u>Test Outcome</u>
Welcome	“Register a new user” button	Press button	Active widget set to register widget	Active widget set to register widget
	“Login as an existing user” button	Press button	Active widget set to login widget	Active widget set to login widget
Register	“Back” button	Press button	Active widget set to welcome widget	Active widget set to welcome widget
	Username Field	Enter 4 key string “hehe”	“hehe” in field	“hehe” in field
		Enter 15 key string “test123!@#$%^&*”	“test123!@#$%^&*” in field	“test123!@#$%^&*” in field
		Enter 20 key string “thisisover15keyslong”	“thisisover15key” in field	“thisisover15key” in field
	Password Field	Enter 4 key string “hehe”	“••••” in field	“••••” in field
		Enter 15 key string “test123!@#$%^&*”	“•••••••••••••••” in field	“•••••••••••••••” in field
		Enter 20 key string “thisisover15keyslong”	“•••••••••••••••” in field	“•••••••••••••••” in field
	“Register” button	Register with username “test123” and password “321tset”	Enter DCM main window, credentials file updated with (“test123”: “321tset”)	Enter DCM main window, credentials file updated with (“test123”: “321tset”)

		Register with username “test123” and password “321tset” (already registered)	Popup window stating “Username already exists, please login instead!”	Popup window stating “Username already exists, please login instead!”
Login	“Back” button	Press button	Active widget set to welcome widget	Active widget set to welcome widget
	Username Field	Enter 4 key string “hehe”	“hehe” in field	“hehe” in field
		Enter 15 key string “test123!@#$%^&*”	“test123!@#$%^&*” in field	“test123!@#$%^&*” in field
		Enter 20 key string “thisisover15keyslong”	“thisisover15key” in field	“thisisover15key” in field
	Password Field	Enter 4 key string “hehe”	“••••” in field	“••••” in field
		Enter 15 key string “test123!@#$%^&*”	“•••••••••••••••” in field	“•••••••••••••••” in field
		Enter 20 key string “thisisover15keyslong”	“•••••••••••••••” in field	“•••••••••••••••” in field
	“Login” button	Login with username “test” and password “test”	Pop window stating “Username does not exist, please register instead!”	Pop window stating “Username does not exist, please register instead!”
		Login in with username “test123” and password “321tset” (already registered)	Enter DCM main window	Enter DCM main window

3.2 DCM Main Window

3.2.1 DCM Utility Functions

Window	Test	Test Case	Expected Outcome	Test Outcome
DCM	“About”	Press button	Popup window	Popup window

	button		“About” showing: Application mode, Software revision, DCM serial number, Device serial number and Institution name	“About” showing: Application mode, Software revision, DCM serial number, Device serial number and Institution name
	“New Patient” button	Not connected to pacemaker	Popup window stating “Please plug in a pacemaker!”	Popup window stating “Please plug in a pacemaker!”
		Connected to registered pacemaker	Popup window stating “Already registered this pacemaker!”	Popup window stating “Already registered this pacemaker!”
		Connected to unregistered pacemaker or multiple pacemakers	Unable to test due to access to single pacemaker	
	“Quit” button	Press button	Exit program	Exit program

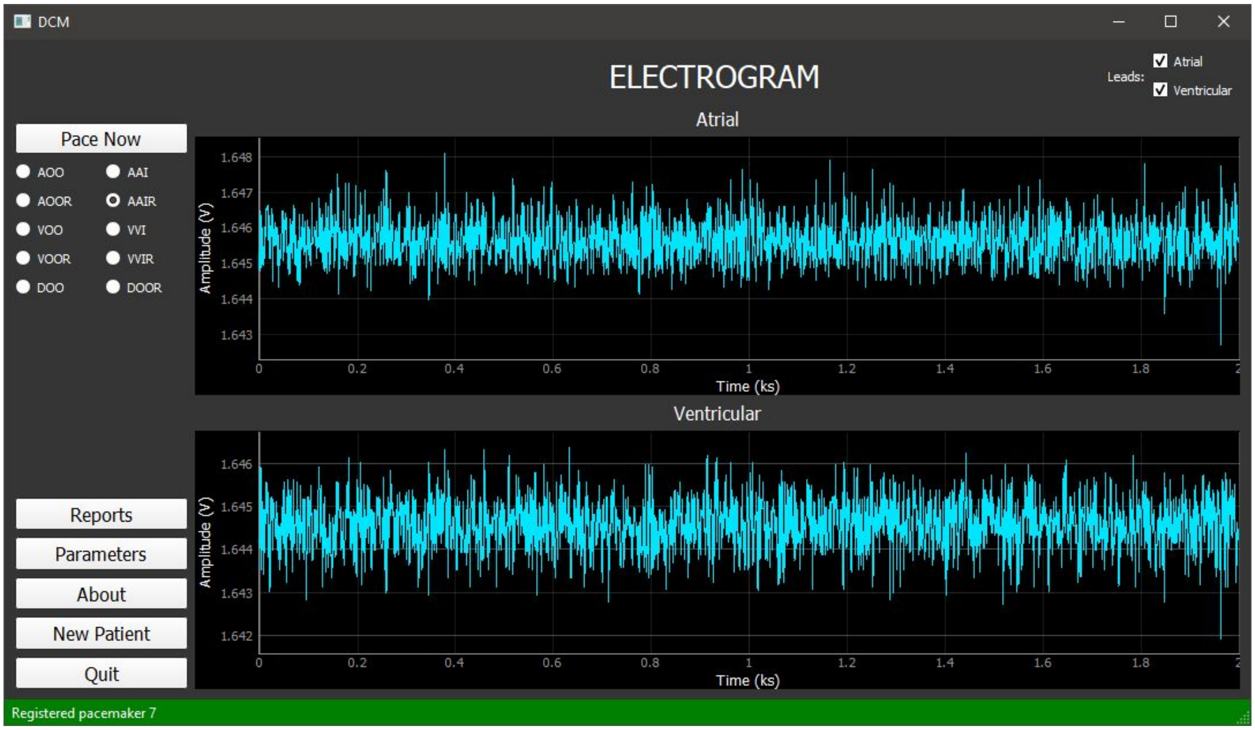
3.2.2 Pacing Mode Selection

Window	Test	Test Case	Expected Outcome	Test Outcome
DCM	Select pacing mode	Select “DOO”	DOO radio button selected, all others unselected	DOO radio button selected, all others unselected
		Select “AAI”	AAI radio button selected, all others unselected	AAI radio button selected, all others unselected

3.2.3 Electrograms

Window	Test	Test Case	Expected Outcome	Test Outcome
DCM	Electrogram	Receive telemetry data from connected pacemaker	Graph changing values and scrolling as it receives telemetry data	Graph changing values and scrolling as it receives telemetry data

		Hold right click and move mouse right	Zoom in on graph	Zoom in on graph
		Hold right click and move mouse left	Zoom out on graph	Zoom out on graph
	Lead checkbox	Uncheck “Atrial” checkbox	Atrial graph no longer shown	Atrial graph no longer shown
		Check “Atrial” checkbox	Atrial graph shown	Atrial graph shown
		Uncheck “Ventricular” checkbox	Ventricular graph no longer shown	Ventricular graph no longer shown
		Check “Ventricular” checkbox	Ventricular graph shown	Ventricular graph shown



Registered pacemaker 7

Figure 3.2.3: Telemetry Data Electrograms

3.2.4 Pacing

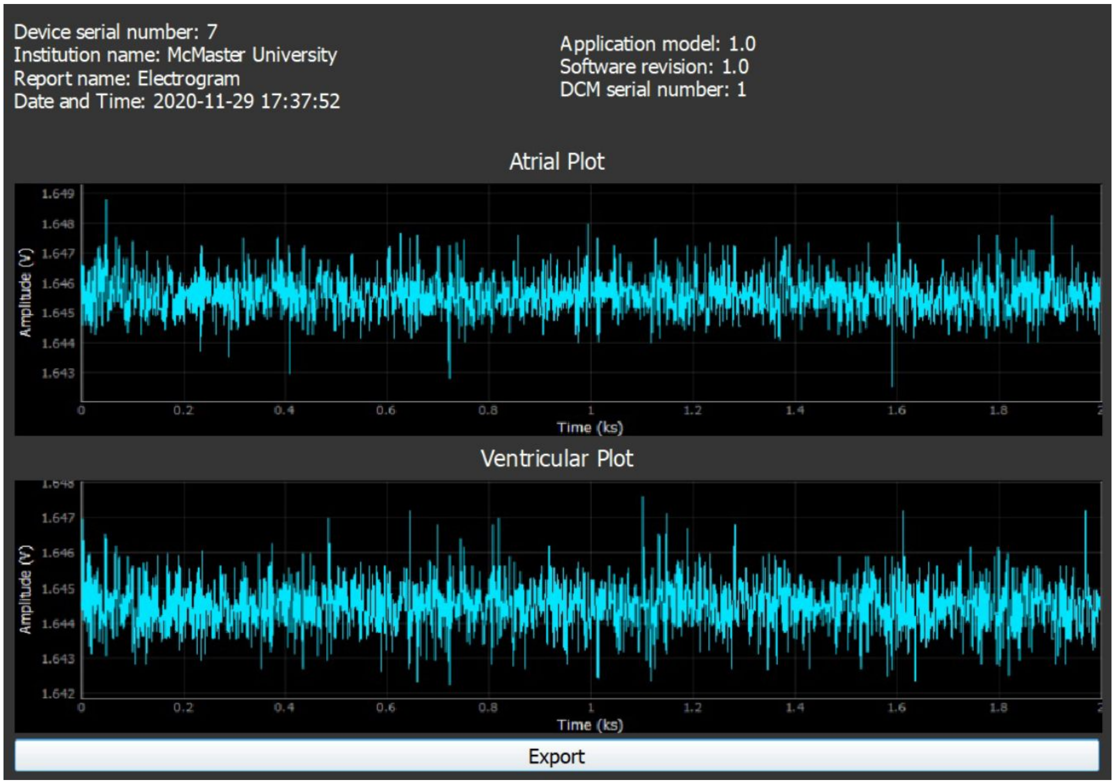
<u>Window</u>	<u>Test</u>	<u>Test Case</u>	<u>Expected Outcome</u>	<u>Test Outcome</u>

DCM	“Pace Now” button	Pace with default values in “AOO”	Received (0, 60, 120, 5.0, 10, 3.0, 5.0, 1, 3.0, 320, 250, 250, 150, 120, 30, 8, 5, 3) from pacemaker echo	Received (0, 60, 120, 5.0, 10, 3.0, 5.0, 1, 3.0, 320, 250, 250, 150, 120, 30, 8, 5, 3) from pacemaker echo
		Pace with “V-High” Activity Threshold, 4.2 V Ventricular Amplitude, 1 Response Factor, default values on rest in “DOOR”	Received (9, 60, 120, 5.0, 10, 3.0, 4.2, 1, 3.0, 320, 250, 250, 150, 120, 30, 1, 5, 6) from pacemaker echo	Received (9, 60, 120, 5.0, 10, 3.0, 4.199999809265137 , 1, 3.0, 320, 250, 250, 150, 120, 30, 1, 5, 6) from pacemaker echo

3.2.5 Reports

<u>Window</u>	<u>Test</u>	<u>Test Case</u>	<u>Expected Outcome</u>	<u>Test Outcome</u>
DCM	“Report” button	Press button	Open Reports window	Open Reports window
Reports	“Electrogram” button	Press button	Open Electrogram Report window with header information matching “About” and two screenshots of electrogram graphs.	Open Electrogram Report window with header information matching “About” and two screenshots of electrogram graphs.
	“Bradycardia Parameters” button	Press button in “VOO”	Open Bradycardia Report window with header information matching “About” and four VOO parameters listed.	Open Bradycardia Report window with header information matching “About” and four VOO parameters listed.
		Press button in “DOOR”	Open Bradycardia Report window with header information matching “About” and twelve DOOR parameters listed.	Open Bradycardia Report window with header information matching “About” and twelve DOOR parameters listed.
Electrogram Report	“Export” button	Press button	Open file explorer window allowing user to save PDF file. PDF file contains screenshot of report	Open file explorer window allowing user to save PDF file. PDF file contains screenshot of report

			window.	window.



Device serial number: 7
Institution name: McMaster University
Report name: Electrogram
Date and Time: 2020-11-29 17:37:52

Application model: 1.0
Software revision: 1.0
DCM serial number: 1

Figure 3.2.5: Exported Electrogram Report PDF

3.2.6 Parameters

Window	Test	Test Case	Expected Outcome	Test Outcome
DCM	“Parameters” button	Press button	Open Parameters window	Open Parameters window
Parameters	Open window in selected pacing mode	Open window in “AAI”	Show "Lower Rate Limit", "Upper Rate Limit", "Atrial Amplitude", "Atrial Pulse Width", "Atrial Sensitivity", "ARP", "PVARP"	Show "Lower Rate Limit", "Upper Rate Limit", "Atrial Amplitude", "Atrial Pulse Width", "Atrial Sensitivity", "ARP", "PVARP"
		Open window in “VOOR”	Show "Lower Rate Limit", "Upper Rate Limit", "Maximum Sensor Rate", "Ventricular Amplitude”,	Show "Lower Rate Limit", "Upper Rate Limit", "Maximum Sensor Rate", "Ventricular Amplitude”,

			"Ventricular Pulse Width", "Activity Threshold", "Reaction Time", "Response Factor", "Recovery Time"	"Ventricular Pulse Width", "Activity Threshold", "Reaction Time", "Response Factor", "Recovery Time"
	Change parameter values	Change “Lower Rate Limit” in “AOO” to 30 ppm and press “Confirm changes”	Parameters value for user “dank” updated in parameters file. Pace now echos 30 for “Lower Rate Limit”	Parameters value for user “dank” updated in parameters file. Pace now echos 30 for “Lower Rate Limit”
		Login in as different user (aaron)	Parameter value for user “aaron” remains 60 ppm for “Lower Rate Limit”	Parameter value for user “aaron” remains 60 ppm for “Lower Rate Limit”
		Attempt to decrease “Atrial Amplitude” below 0.0 V	“Atrial Amplitude” value stops at 0.0 V	“Atrial Amplitude” value stops at 0.0 V
		Attempt to increase “Reaction Time” over 50 sec	“Reaction Time” value stops at 50 sec	“Reaction Time” value stops at 50 sec
	“Reset to defaults” button	Press button	All parameter values reset to nominal values, file updated for user.	All parameter values reset to nominal values, file updated for user.

3.3 Test Conclusions

In all tests, the DCM worked as expected and all test cases passed.