Table of Contents

[Software Design Details 2](#_Toc467808782)

[3.1 UML Diagrams for Utility Classes [1] 2](#_Toc467808783)

[3.2 Uses Relationship 4](#_Toc467808784)

[3.3 Utility Classes 5](#_Toc467808785)

[Class 1: Pacemaker() 5](#_Toc467808786)

[Class 2: Activity() 5](#_Toc467808787)

[Class 3: HeartMonitor() 5](#_Toc467808788)

[Class 4: Logging() 6](#_Toc467808789)

[Class 5: Pace() 6](#_Toc467808790)

[Class 6: Communications() 6](#_Toc467808791)

[3.4 UI Class Methods 8](#_Toc467808792)

[3.5 Design Requirements Likely to Change 8](#_Toc467808793)

[3.6 Design Decisions Likely to Change 9](#_Toc467808794)

[References 10](#_Toc467808795)

[Appendix A [1] 11](#_Toc467808796)

# Software Design Details

* 1. UML Diagrams for Utility Classes [1]

The following diagrams represent the classes and methods within those classes that when called, execute tasks that will allow the Pacemaker to function. The diagrams also give insight into the permissions needed to access particular methods and variable values.

|  |
| --- |
| **Pace()** |
| PACESTATE: enum class  PACEMODE: enum class  prevBPM: private uint8\_t  vPaceAmp: public float  vPaceWidth\_milliseconds: public uint16\_t  pacingMode: public PACEMODE  pacingState: public PACESTATE  baseHeartRate: public uint8\_t  maxHeartRate: public uint8\_t  egramData: public float  \*activityTrip(): public bool  paceTicker: private Ticker  changeVar[6]: private Timeout  egramData: public float |
| Pace(bool\*): public Class-Object Constructor  getPaceMode(): protected PACEMODE  getPaceState(): protected PACESTATE  setPaceRate(uint8\_t): public void  updateEgramData(): public void  paceTick(): private void  paceVentricle(): private void  paceAtrium(): private void  paceTick(): private void  paceVentricleStage1(): private void  paceVentricleStage2(): private void  paceVentricleStage3(): private void  paceVentricleStage4(): private void  paceVentricleStage5(): private void  paceVentricleStage6(): private void |

|  |
| --- |
| **class\_name** |
| \*\*variable: type |
| \*\*method: type |

|  |
| --- |
| **main()** |
|  |
| main(): public int |

|  |
| --- |
| **Activity()** |
| activityUpdateRate: const private float  activityTimeThresholdSeconds: private uint8\_t  accX: private float  accY: private float  accZ: private float  activitySumMax: const private float  activitySumThreshold: const private float  updateActivityTicker: private Ticker  activityTrip: public bool  activityCounter: public uint64\_t  accMagnitudeSum: public float  accelerometer: protected FXOS8700Q\_acc  ACTIVITYSTATE: enum class |
| updateAcc(): private void  Activity(): public Class-Object Constructor  getAccX(): public float  getAccY(): public float  getAccZ(): public float  AccReady(): public uint32\_t  getAccMagnitude(): public float  getPatientActivity(): protected ACTIVITYSTATE |

|  |
| --- |
| **HeartMonitor()** |
| VRP: protected uint16\_t  V\_REF: private float |
| HeartMonitor(): public Class-Object Constructor getVentricleRate(): protected uint8\_t  getAtriumRate(): protected uint8\_t |

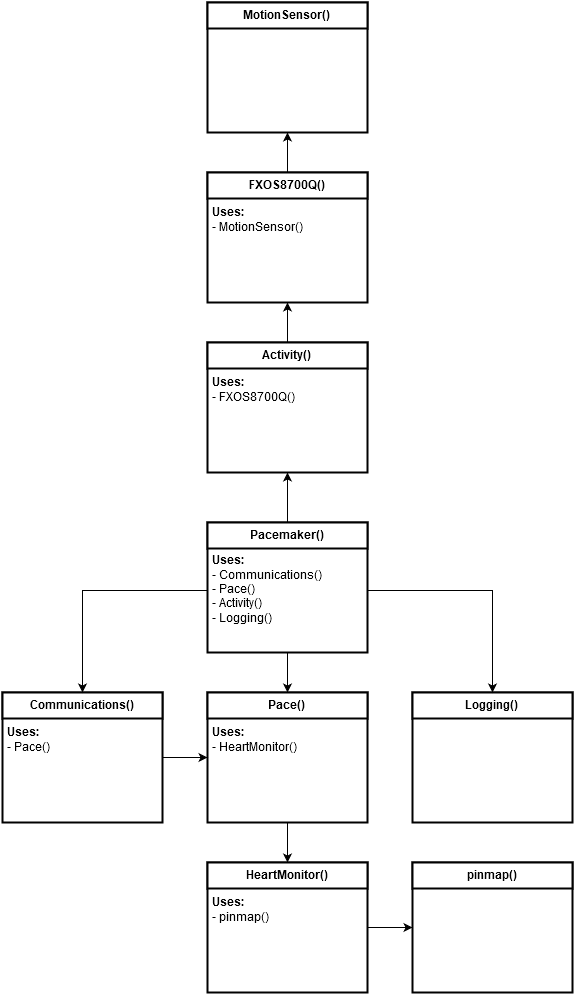
|  |
| --- |
| **Pacemaker()** |
| fnCode: private uint8\_t  deviceID: private char[64]  deviceImplantDate: private char[64]  leadImplantDate: private char[64]  communications: private Communications  activity: private Activity  logging: private Logging  pace: private Pace |
| Pacemaker(): public Class-Object Constructor  mainLoop(): public void |

|  |
| --- |
| **Communications()** |
| streamDataTicker: private Ticker  streaming: private bool  dataStreamRate: private float  \*streamingData: private float  serialBuffer[256]: volatile private uint8\_t  packetStruct: volatile private SERIAL\_PACKET  baudRate: private uint32\_t  dataInBuffer: public bool  DCMConnected: private bool  USBSerialConnection: public Serial  UPDATE\_DEVICE\_INFO: definition int  UPDATE\_PARAMS: definition int  DCM\_START\_STREAM\_SIG: definition int  DCM\_STOP\_STREAM\_SIG: definition int  DCM\_CONNECT\_SIG: definition int  DCM\_DISCONNECT\_SIG: definition int  READ\_PARAMS: definition int |
| Communications(): public Class-Object Constructor  streamDataTick(): private void  twoBytesFromBuffer(volatile uint8\_t[], uint16\_t): private uint16\_t  floatFromBuffer(volatile uint8\_t[], uint16\_t): private float  stringsFromBuffer(volatile uint8\_t[], uint8\_t, ...): private void  serialCallback(): private void  transmitDeviceInfo(): private void  setDataPointers(args): public void  readBuffer(): public void  initDataStream(float\*): public void  setStreamMode(bool): public void |

|  |
| --- |
| **Logging()** |
|  |
| Logging(): public Class-Object Constructor  addCardiacEvent(): protected bool  readCardiacEvents(): protected void  clearCardiacEvents(): protected bool |

Note: Valid values for programmable parameters in the following diagrams can be found in the table located in Appendix A.

|  |
| --- |
| **pinmap()** |
|  |
|  |

* 1. Uses Relationship
  2. Utility Classes

The following tables outlines the public, private and protected methods making up each class defined above in section 3.1.

Class 1: Pacemaker()

This class manages distribution of information that is specific to a particular pacemaker device, including, device ID, device implant date, and lead implant date. The class also tells the device whether or not to stream data to the DCM for interrogation.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Return Type** | **Description** |
| Pacemaker() | Class-Object | Constructor for Pacemaker() class.  Calls setDataPointers() and initDataStream() methods from Communications() class. Method takes memory address references for pointers to the following programmable parameters:   * fnCode * pacingState: Pace() * pacingMode: Pace() * hysteresis * hysteresisInterval * vPaceAmp: Pace() * vPaceWidth\_milliseconds: Pace() * VRP: Pace() * baseHeartRate: Pace() * maxHeartRate: Pace() * deviceID * deviceImplantDate * leadImplantDate |
| mainLoop() | void | This method acts as the ‘main loop’ of the pacemaker code. It periodically checks the dataInBuffer variable located in Communications() and calls readBuffer() of the same class if the value returned is true. Every cycle, this method also calls updateEgramData() in the Pace() class. |

|  |  |
| --- | --- |
| **Method Name** | **Function Dependencies**  **(if applicable)** |
| Pacemaker() | * setDataPointers(): Communications * initDataStream(): Communications |
| mainLoop() | * dataInBuffer(): Communications * USBSerialConnection(): Communications * readBuffer(): Communications * setStreamMode(): Communications * updateEgramData(): Pace |

Class 2: Activity()

This class contains variables and methods that are responsible for dealing with sensor input from the onboard accelerometer to the pacemaker device. The module hides information concerning sensor thresholds and configuration. Methods within this class interface with peripheral sensors through inherited GPIO port information. Accelerometer peripherals are accessed solely through this class which then stores the retrieved information for other classes to use.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Return Type** | **Description** |
| Activity() | Class-Object | Constructor for Activity() class. Implements and enables the accelerometer. Creates a ticker that is used to update the accelerometers values. |
| getPatientActivity() | ACTIVITYSTATE | Returns an activity state {V\_LOW,LOW,MED\_LOW,MED,MED\_HIGH,HIGH,V\_HIGH}  **[1 – 5.9A]** based on the user's activity. |
| AccReady() | uint32\_t | Returns binary Boolean value reflecting whether or not accelerometer data is available. |
| updateAcc() | void | Reads the accelerometer values into variables using the getter methods defined locally. These values are then summed, and applied through a condition to return a value between 0 and the maxSumActivity. If this value passes the activity threshold, the activityTrip variable is set to true. Otherwise, it is set to false. |
| getAccX() | float | Reads and returns the accelerometer x value. |
| getAccY() | float | Reads and returns the accelerometer y value. |
| getAccZ() | float | Reads and returns the accelerometer z value. |
| getAccMagnitude() | float | Sums all the accelerometer values (x, y, z) and returns the summed value. |

|  |  |
| --- | --- |
| **Method Name** | **Function Dependencies**  **(if applicable)** |
| Activity() | * updateActivityTicker(): Activity * accelerometer.enable():FXOS8700CQ |
| getPatientActivity() |  |
| AccReady() | * accelerometer.dataReady():FXOS8700CQ |
| updateAcc() | * getAccX(): Activity * getAccY(): Activity * getAccZ: Activity * getAccMagnitude(): Activity |
| getAccX() | * float |
| getAccY() | * float |
| getAccZ() | * float |
| getAccMagnitude() | * float |

Class 3: HeartMonitor()

This class contains variables and methods that are responsible for storing and monitoring pacing rate information from the ventricle and atrium. Methods in this class interface with the leads to return the ventricle and atrium heart rates.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Return Type** | **Description** |
| HeartMonitor() | Class-Object | Constructor for HeartMonitor() Class. |
| getVentricleRate() | uint8\_t | Sense ventricle contraction rate |
| getAtriumRate() | uint8\_t | Sense atrium contraction rate |

Class 4: Logging()

This class contains methods used to detect and record pacing anomalies. The Boolean methods are used to verify that the anomaly has been recorded successfully. The successful recording, can then be shown to doctors to isolate current and future problems.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Return Type** | | **Description** |
| Logging() | | Class-Object | Constructor for Logging() class. |
| addCardiacEvent() | | bool | Adds events to storage device (e.g. microSD card) |
| readCardiacEvents() | | void | Reads cardiac events from storage device. |
| clearCardiacEvents() | | bool | Clears events from storage device. |

Class 5: Pace()

This class is used to determine the pace mode and pace state. This class utilizes a pointer to look at the memory location that the pace mode from the communication class is stored. Within this class, the appropriate responses for each of the pace modes are stored in a case structure, waiting to be called upon.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Return Type** | | **Description** |
| Pace(bool \*aTrip) | | Class-Object | Constructor for Pace() class. Initializes the initial state of pacing, sets an activity pointer, and sets the pace rate to the base heart rate. |
| getPaceState() | | void | Returns current value of pacingState. **[1 – 3.6.3]** |
| getPaceMode() | | enum | Returns current value of pacingMode **[1 – 3.5 Table 2]** |
| setPaceRate(uint8\_t BPM) | | void | Takes a pace rate parameter and sets the interrupt ticker to fire at that rate. |
| paceTick() | | void | Based on the pacing mode, pace in each defined way (VOO, VOOR, ect.). |
| paceAtrium() | | void | Pace through atrium lead at vPaceAmp and vPaceWidth |
| paceVentricle() | | void | Pace through ventricle lead at vPaceAmp and vPaceWidth |
| paceVentricleStage1() | | void | Implements stage 1 of pacing ventricle. |
| paceVentricleStage2() | | void | Implements stage 2 of pacing ventricle. |
| paceVentricleStage3() | | void | Implements stage 3 of pacing ventricle. |
| paceVentricleStage4() | | void | Implements stage 4 of pacing ventricle. |
| paceVentricleStage5() | | void | Implements stage 5 of pacing ventricle. |
| paceVentricleStage6() | | void | Implements stage 6 of pacing ventricle. |

Class 6: Communications()

This class is the largest and most in-depth class. This class interfaces with the GUI to set the following parameters: pacing state, pacing mode, hysteresis interval, ventricle pacing amplitude, ventricle pacing width, ventricular refractory period, and base heart rate. This class is used to communicate the above information to other classes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Return Type** | | **Description** |
| Communications() | | Class-Object | Constructor for Communications() class. baudRate is set to 57600, and DCM is connected. |
| streamDataTick() | | void | Streams data packets to DCM at specified tick interval. |
| twoBytesFromBuffer() | | uint16\_t | Converts two bytes from buffer to 16bit unsigned integer |
| floatFromBuffer() | | float | Converts four bytes from buffer into a float |
| stringsFromBuffer() | | void | Converts a buffer into n newline-delimited strings |
| connectDCM() | | bool | Initializes communication packets with DCM |
| serialCallback() | | void | Initializes serial event interrupt to fire whenever serial data is available |
| transmitDeviceInfo() | | void | Transmits deviceID, implantDate, leadImplantDate, and battery information to DCM when communications initialized. |
| sendEGM() | | bool | Sends EGM data from Sense() class to DCM. |
| setDataPointers() | | void | Sets internal pointers whenever serial data becomes available in order for Communications() class to update variables without having to call methods. |
| initEGM() | | void | Initializes structure for transmitting EGM data. |
| readBuffer() | | void | Sets pointer values based on serial buffer. |
| initDataStream(float\*) | | void | Sets data stream-rate, along with data stream pointer. |
| setStreamMode(bool) | | void | Turns data stream on and off. |

* 1. UI Class Methods

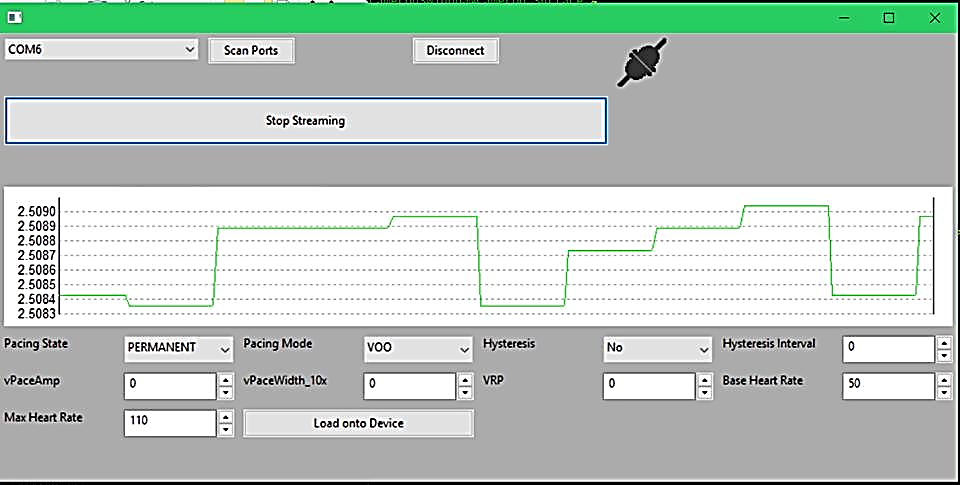


Figure 1 - Prototype DCM Interface

The user interface displayed above in Figure 1 - Prototype DCM Interface Figure 1 shows an approximate layout for the computer-driven DCM to be used by qualified doctors and nurses. The interface is designed to show important information such as patient info, device ID, communication status, and battery voltage in a clean, easy to read manner. The DCM is designed to take advantage of methods and parameters in the pacemaker code in order to customize functionality for individual patient needs, while maintaining information hiding constructs. All information received and transmitted by the DCM is routed through the Communications() object in the pacemaker code, effectively making this object and its methods an intermediary between DCM input, and the safety-critical state variables controlling the pacemaker’s overall behavior. Another important aspect of the DCM is the ability to read and display EGRAM values sent to it from the pacemaker. These values are graphed in real time on a display integrated into the DCM’s UI. Changes to the appearance and functionality can be expected in future versions, as more pacemaker functionality is added, or requirements are changed. The communication structure between the DCM and the pacemaker is unlikely to change due to the structure of the data packets, but the data processing on the pacemaker may change to meet new design decisions.

* 1. Design Requirements Likely to Change

|  |  |
| --- | --- |
| **Requirement** | **Reason for Potential Change** |
| Logged Detail of Cardiac Events Detected | Detailed logs of cardiac events may be kept for diagnostic purposes, however, given an abundance of such events, detail may need to be decreased in order to preserve storage space. |
| p\_vPaceAmp & p\_vPaceWidth | As scar-tissue generates over-top of pacemaker leads, resistance between leads subject to change. Applied voltage to induce ventricular contraction may need to be changed accordingly. |
| Base Heart Rate | Depending on patient age / level of physical activity, resting base heart rate should be customizable. |

* 1. Design Decisions Likely to Change

|  |  |
| --- | --- |
| **Design Decision** | **Reason for Potential Change** |
| Program Structure | Modified object or class structures may change to better suit the overall program |
| Variable names | Names may change to match naming convention or to match requirements. |
| Implementing VOOR | Current pacing signals change sharply (for VOOR mode) but will likely later change to be more gradual.  VOOR mode is currently programmed based on a threshold, with modifiable parameters being the rate of increase of the checked value, the threshold value, the maximum value that the checked value can rise to, and the rate at which the checked value will fall when there is no activity. |
| Implementing a more in depth Logging class | Logging class will be used to store events. This is not implemented yet. |
| More states need to be added | Additional states need to be implemented, such as an addition to fnCode variable and its effect in Communication() |

# References

|  |  |
| --- | --- |
| [1] | Boston Scientific, "PACEMAKER System Specifications," Boston Scientific, 2007. |

# Appendix A [1]

