Table of Contents

[Software Design Details 2](#_Toc467763000)

[3.1 UML Diagrams for Utility Classes 2](#_Toc467763001)

[3.2 Utility Classes 3](#_Toc467763002)

[Class 1: Pacemaker() 3](#_Toc467763003)

[Class 2: Activity() 3](#_Toc467763004)

[Class 3: HeartMonitor() 3](#_Toc467763005)

[Class 4: Logging() 3](#_Toc467763006)

[3.3 UI Class Methods 4](#_Toc467763007)

[3.4 Design Requirements Likely to Change 4](#_Toc467763008)

[3.5 Design Decisions Likely to Change 4](#_Toc467763009)

# Software Design Details

* 1. UML Diagrams for Utility Classes

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  vPaceAmp: protected float  vPaceWidth\_milliseconds: protected uint16\_t  pacingMode: protected PACEMODE  pacingState: protected PACESTATE  baseHeartRate: private uint8\_t  maxHeartRate: private uint8\_t  paceTicker: private Ticker |
| Pace(): public Class-Object  getPaceMode(): protected PACEMODE  getPaceState(): protected PACESTATE  setPaceRate(uint8\_t): private void  paceTick(): private void  paceVentricle(): private void  paceAtrium(): private void |

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

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

|  |
| --- |
| **Activity()** |
| activityTimeThresholdSeconds: private uint8\_t  ACTIVITYSTATE: enum class |
| Activity(): public Class-Object  getPatientActivity(): protected ACTIVITYSTATE |

|  |
| --- |
| **Pacemaker()** |
| fnCode: private uint8\_t  deviceID: private char[64]  deviceImplantDate: private char[64]  leadImplantDate: private char[64] |
| Pacemaker(): public Class-Object  mainLoop(): public void |

|  |
| --- |
| **HeartMonitor()** |
| VRP: protected uint16\_t |
| HeartMonitor(): public Class-Object getVentricleRate(): protected uint8\_t  getAtriumRae(): protected uint8\_t |

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

* 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. Note that the *Sense* and *Communications* classes extend the *Pacemaker* class allowing them to inherit the properties defined in the Pacemaker class. The *Pace* class extends the *Sense* class in order to inherit properties of both Pacemaker and Sense. This allows information to be hidden in an appropriate class but made accessible without storing in multiple locations through getter and setter methods.

Class 1: Pacemaker()

This class stores information essential to the operation of a generic pacemaker. It includes variables describing the status of the battery, location of GPIO ports and memory addresses for TxRx I2C operations. The methods and variables in this class are limited in scope and provide only a support framework on which other classes are able to operate within.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Return Type** | **Description** |
| Pacemaker() | Class-Object | Constructor for Pacemaker() class.  Calls setDataPointers() method from Communications() class. Method takes memory address references for pointers to programmable parameters. |
| mainLoop() | void | This method acts as the ‘main loop’ of the pacemaker code. It periodically check the dataInBuffer variable located in Communications() and calls readBuffer() of the same class if value is true. |

Class 2: Activity()

This class contains variables and methods that are responsible for dealing with sensor input 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 and access / store information for use by other modules.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Return Type** | **Description** |
| Activity() | Class-Object | Constructor for Activity() class. |
| getPatientActivity() | ACTIVITYSTATE | Reads accel |

Class 3: HeartMonitor()

This class is responsible for using serial communication protocols in order to send and receive data to and from the DCM application. It includes data structures to store and transmit EGM data as well as send and receive critical device information e.g. deviceID, implantDate, etc.

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

The pace class contains variables and methods that are used to produce the prescribed external pacemaker functionality required by the patient. It contains methods for setting the desired pacing mode, pacing parameter values and other variables enabling the desired therapeutic effect to be achieved within the patient. This class uses its inheritance from both the sense and pacemaker classes in order to interface with the attached leads and onboard sensors.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Return Type** | | **Description** |
| setPaceMode(enum) | | void | Takes desired pace mode as enum per Generic NBG code {VVI, VOO, AOO, DDDR, etc} |
| getPaceMode() | | enum | Returns current value of pacingMode |

* 1. UI Class Methods



Figure 1 - Prototype DCM Interface

The user interface displayed above in Figure 1 - Prototype DCM InterfaceFigure 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() class in the pacemaker code effectively making this class and it’s methods an intermediary between the user input and the safety-critical state variables controlling the pacemaker’s overall behavior. Changes to the look and functionality can be expected as more pacemaker functionality is added, however this intermediary behavior is expected to remain unchanged.

* 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** |
| Appearance and features offered on the User Interface | In the future, because of the relative ease with which software can be changed, features may need to be added or removed from the UI. |
| Checks on whether a value is in appropriate range are not implemented at this point. | In order to minimize risk to patients and maximize safety of the implanted device, safety checks will be written as software development progresses to ensure values changed in the device by doctors or other medical staff are within a safe operating range as outlined in the requirements. |
| Data structures responsible for holding communications data between pacemaker and DCM are pre-declared arrays of fixed-size | When EGM data is transmitted in software testing, size of data-structures may need to be amended as number of stored points increases in practise. Provisions for dynamic arrays & vectors may also be added. |