# 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.

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

|  |
| --- |
| **main()** |
| patientFirstName: private string  patientLastName: private string  patientAge: private uint18\_t  doctorNotes: private string |
|  |

|  |
| --- |
| **Pacemaker()** |
| deviceID: private const string  deviceImplantDate: private const string  leadImplantDate: private const string  replaceBattVoltage: protected const float  batteryVoltage: protected float  cardiacEvents: protected [Object]  leadImpedance: protected float  leadImpedanceThreshold: private float  leadOneInPin: private enum  leadTwoInPin: private enum  leadOneOutPin: private enum  leadTwoOutPin: private enum  maxVOut: private float  comPort: private uint8\_t  txRegister: private uint8\_t  rxRegister: private uint8\_t |
| setLeadPins([enum]): protected void  getLeadPins(): protected [enum]  setMaxVOut(uint16\_t): protected void  getMaxVOut(): protected uint16\_t  setTxRxReg([uint8\_t]): protected void  getTxRxReg(): protected [uint8\_t]  voltageTest(float): protected float  getCardiacEvents(): public Object  clearCardiacEvents(): private void  getLeadImpedance(): protected float  getBatteryStatus(): public enum |

|  |
| --- |
| **Communications() extends Pacemaker** |
| i\_CommIn: [16bytes]  vRaw: uint16\_t  f\_Mmarker: uint16\_t  o\_CommOut: uint8\_t  baudRate: uint32\_t |
| connectDCM(): private bool  sendEGM(): private bool  initEGM(): public void  recieveDeviceInfo(): protected [string]  transmitDeviceInfo([string]): private void |

|  |
| --- |
| **Sense() extends Pacemaker** |
| chambersSensed: private enum  activityResponse: private enum  magnetInPlace: private bool  activityThreshold: private enum  maxSensorRate: protected uint16\_t |
| setChambersSensed(enum): protected void  getChambersSensed(): public enum  setActivityResponse(enum): protected void  getActivityResponse(): public enum  setMagnetInPlace(bool): protected void  getMagnetInPlace(): public bool  measureLeadImpedance(): protected void  measureBatteryVoltage(): protected void  setActivityThreshold(enum): protected void  getActivityThreshold(): protected enum |

|  |
| --- |
| **Pace() extends Sense** |
| pacingState: private enum  pacingMode: private enum  hysteresis: private Boolean  hysteresisInterval: private uint16\_t  lowrateInterval: private uint16\_t  vPaceAmp: private uint16\_t  vPaceWidth: private uint16\_t  VRP: private uint16\_t  maxHeartRate: private uint8\_t  baseHeartRate: private uint8\_t |
| setPaceMode(enum): protected void  getPaceMode(): public enum  setPaceState(enum): protected void  getPaceState(): public enum  setHysteresisInterval(uint16\_t): protected void  getHysteresisInterval(): public uint16\_t  setLowRateInterval(uint16\_t): protected void  getLowRateInterval(): public uint16\_t  setvPaceAmp(uint16\_t): protected void  getvPaceAmp(): public uint16\_t  setvPaceWidth(uint16\_t): protected void  getvPaceWidth(): public uint16\_t  setVRP(uint16\_t): protected void  getVRP(): public uint16\_t  setMaxHeartRate(uint8\_t): protected void  getMaxHeartRate(): protected uint8\_t  setBaseHeartRate(uint8\_t): protected void  getBaseHeartRate(): protected uint8\_t |

* 1. 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** | **Next Action (If action event triggered)** |
| setLeadPins([enum]) | void | Sets values for Lead(x)InPin,Lead(x)OutPin based on hardware GPIO requirements | None |
| getLeadPins() | [enum] | Accesses values of Lead(x)InPin,Lead(x)OutPin | None |
| setMaxVOut(float) | void | Sets maxVOut variable to maximum safe pace amplitude based on battery capacity | None |
| getMaxVOut() | float | Gets vale of maxVOut | None |
| setTxRxReg([uint8\_t]) | void | Sets hex memory locations of Tx and Rx registers storing serial buffer | None |
| getTxRxReg() | [int8\_t] | Gets array of Tx / Rx register locations | None |
| voltageTest(float) | float | Takes arg min pace amplitude and increases voltage until ERM registers P-QRS-T sequence. Returns this voltage. | None |
| getCardiacEvents() | Object | Return object containing all stored cardiac events in EEPROM | None |
| clearCardiacEvents() | void | Erases EEPROM containing stored cardiac event data | None |
| getLeadImpedance() | float | Gets value of leadImpedance | None |
| getBatteryStatus() | enum | Uses values of batteryVoltage and replaceBatteryVoltage to determine battery status {BOL,ERN,ERT,ERP} | None |

Class 2: Sense()

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** | **Next Action (If action event triggered)** |
| setChambersSensed(enum) | void | Takes chambers sensed as enum type {NONE, ATRIUM, VENTRICLE, DUAL} and sets value of private variable chambersSensed | None |
| getChambersSensed() | enum | Returns current value of chambersSensed | None |
| setActivityResponse(enum) | void | Takes activity response as enum type {NONE, TRIGGERED, INHIBITED, DUAL} and sets value of private variable activityResponse | None |
| getActivityResponse() | enum | Returns current value of activityResponse | None |
| setMagnetInPlace(bool) | void | Sets value of boolean var magnetInPlace. | None |
| getMagnetInPlace() | bool | Returns value of magnetInPlace that can be used to determine if diagnostic magnetism source in place | None |
| measureLeadImpedance() | void | Used internally to sense and set value of variable leadImpedance following measurement. | If impedance measured greater than leadImpedanceThreshold, set vPaceAmp in pace class to maxVOut. Log event. |
| measureBatteryVoltage() | void | Used internally to sense battery voltage and set value of batteryVoltage variable following measurement | If battery voltage below thresholdBatteryVoltage, enter power-saving state. |
| setActivityThreshold(enum) | void | Sets value of activityThreshold {V-Low, Low, Med-Low, Med, Med-High, High, V-High} | None |
| getActivityThreshold() | enum | Returns value of activityThreshold | None |

Class 3: Communications()

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** | **Next Action (If action event triggered)** |
| connectDCM() | bool | Contains required serial authentication procedures. Returns true on successful connection. | None |
| sendEGM() | bool | Method begins transmitting EGM phase and amplitude data over serial when called | None |
| initEGM() | void | Configures serial connection to send 16 byte EGM packets to DCM interface | None |
| transmitDeviceInfo() | [string] | Sends device info {deviceID, implant date, lead implant date, battery votage, cardiac events,…,etc} to DCM for interrogation | None |
| receiveDeviceInfo([string]) | void | Configures serial connection to receive & store device data. | None |

Class 4: Pace()

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** | **Next Action (If action event triggered)** |
| setPaceMode(enum) | void | Takes desired pace mode as enum per Generic NBG code {VVI, VOO, AOO, DDDR, etc} | Calls setChambersSensed(enum) and setActivityResponse(enum) from Sense() class. |
| getPaceMode() | enum | Returns current value of pacingMode | None |
| setPaceState(enum) | void | Takes pace state as enum type {PERMANENT, TEMPORARY, PACE\_NOW, MAGNET, POWER\_ON\_RESET}, sets value of pacingState | Triggers appropriate methods in Pace() and Pacemaker() classes |
| getPaceState() | enum | Returns current value of pacingState | None |
| setHysteresisInterval(uint16\_t) | void | Sets vale of hysteresisInterval which defines an additional delay interval used when value of hysteresis is True | None |
| getHysteresisInterval() | uint16\_t | Returns current value of hysteresisInterval | None |
| setLowRateInterval(uint16\_t) | void | Sets value of lowrateInterval that specifies maximum delay after a ventricle pace without a spontaneous sense or another pace | None |
| getLowRateInterval() | uint16\_t | Returns current value of lowrateInterval | None |
| setvPaceAmp(uint16\_t) | void | Sets value of vPaceAmp variable representing current amplitude of ventricle pacing output voltage | None |
| getvPaceAmp() | uint16\_t | Returns current value of vPaceAmp variable | None |
| setvPaceWidth(uint16\_t) | void | Sets value of vPaceWidth private variable representing current width of ventrical pace signal (ms) | None |
| getvPaceWidth() | uint16\_t | Returns current value of vPaceWidth | None |
| setVRP(uint16\_t) | void | Sets the value of variable VRP, duration of ventricular refractory period | None |
| getVRP() | uint16\_t | Returns current value of VRP variable | None |
| setMaxHeartRate(uint8\_t) | void | Sets the value of maxHeartRate later used to set upper frequency of pacing | None |
| getMaxHeartRate() | uint8\_t | Returns current value of maxHeartRate | None |
| setBaseHeartRate(uint8\_t) | void | Sets value of baseHeartRate later set to set minimum safe frequency of pacing for particular patient | None |
| getBaseHeartRate() | uint8\_t | Returns current value of baseHeartRate | None |

* 1. UI Class Methods

The following table outline the methods making up the Java classes driving the User Interface.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Return Type** | **Description** | **Next Action (If action event triggered)** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* 1. Design Requirements Likely to Change

|  |  |
| --- | --- |
| **Requirement** | **Reason for Potential Change** |
|  |  |
|  |  |
|  |  |

* 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. |
|  |  |

