# G8R

Generated by Doxygen 1.9.1

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

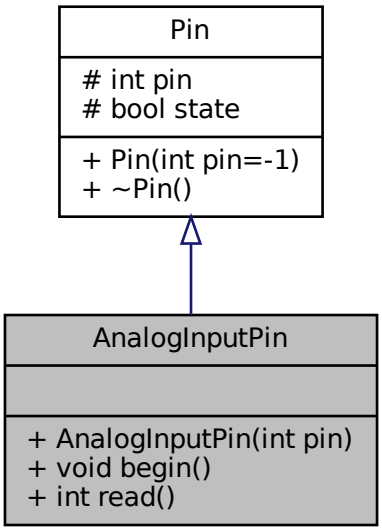
# Chapter 4

# Class Documentation

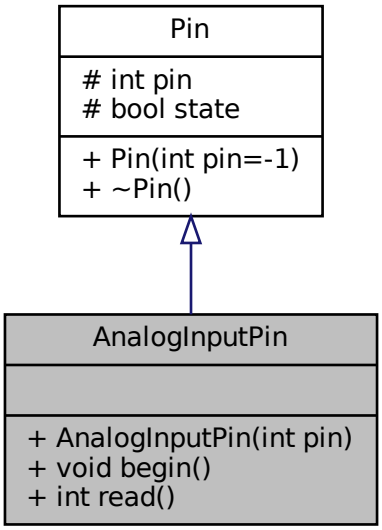
## 4.1 AnalogInputPin Class Reference

`#include <Pin.h>`

Inheritance diagram for AnalogInputPin:



Collaboration diagram for AnalogInputPin:




## Public Member Functions

- AnalogInputPin (int pin)
- void begin ()
- int read ()

## Additional Inherited Members

### 4.1.1 Constructor & Destructor Documentation

#### 4.1.1.1 AnalogInputPin()

```
AnalogInputPin::AnalogInputPin (
            int pin )
```

### 4.1.2 Member Function Documentation

#### 4.1.2.1 begin()

```
void AnalogInputPin::begin ( )
```

#### 4.1.2.2 read()

```
int AnalogInputPin::read ( )
```

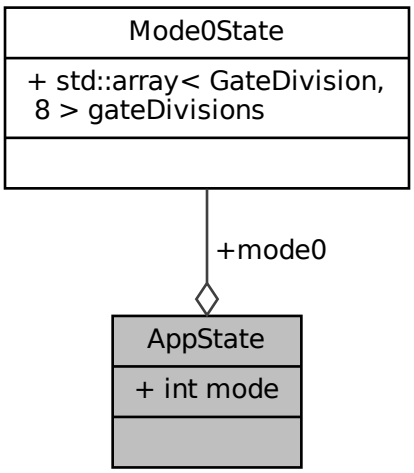
The documentation for this class was generated from the following files:

- include/Pin.h
- src/Pin.cpp

## 4.2 AppState Struct Reference

```
#include <AppState.h>
```

Collaboration diagram for AppState:




### Public Attributes

- int mode
- Mode0State mode0

### 4.2.1 Member Data Documentation

#### 4.2.1.1 mode

```
int AppState::mode
```

#### 4.2.1.2 mode0

```
Mode0State AppState::mode0
```

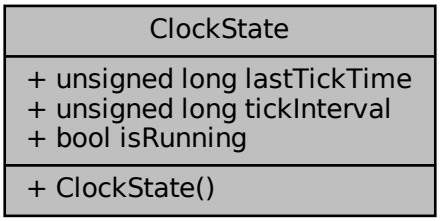
The documentation for this struct was generated from the following file:

- include/AppState.h

## 4.3 ClockState Struct Reference

```
#include <EurorackClock.h>
```

Collaboration diagram for ClockState:




### Public Member Functions

- ClockState ()

### Public Attributes

- unsigned long lastTickTime
- unsigned long tickInterval
- bool isRunning

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 ClockState()

```
ClockState::ClockState ( )  [inline]
```

### 4.3.2 Member Data Documentation

#### 4.3.2.1 isRunning

```
bool ClockState::isRunning
```

#### 4.3.2.2 lastTickTime

```
unsigned long ClockState::lastTickTime
```

#### 4.3.2.3 tickInterval

```
unsigned long ClockState::tickInterval
```

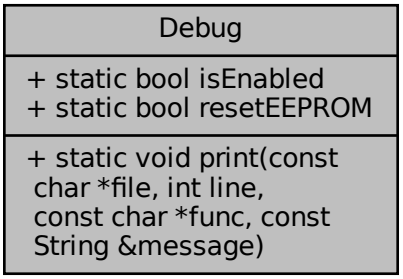
The documentation for this struct was generated from the following file:

- include/EurorackClock.h

## 4.4 Debug Class Reference

```
#include <Debug.h>
```

Collaboration diagram for Debug:



### Static Public Member Functions

- static void print (const char *file, int line, const char *func, const String &message)

### Static Public Attributes

- static bool isEnabled = false
- static bool resetEEPROM = false

## 4.4.1 Member Function Documentation

#### 4.4.1.1 print()

```
void Debug::print (
            const char * file,
            int line,
            const char * func,
            const String & message )  [static]
```

## 4.4.2 Member Data Documentation

#### 4.4.2.1 isEnabled

```
bool Debug::isEnabled = false  [static]
```

#### 4.4.2.2 resetEEPROM

```
bool Debug::resetEEPROM = false  [static]
```

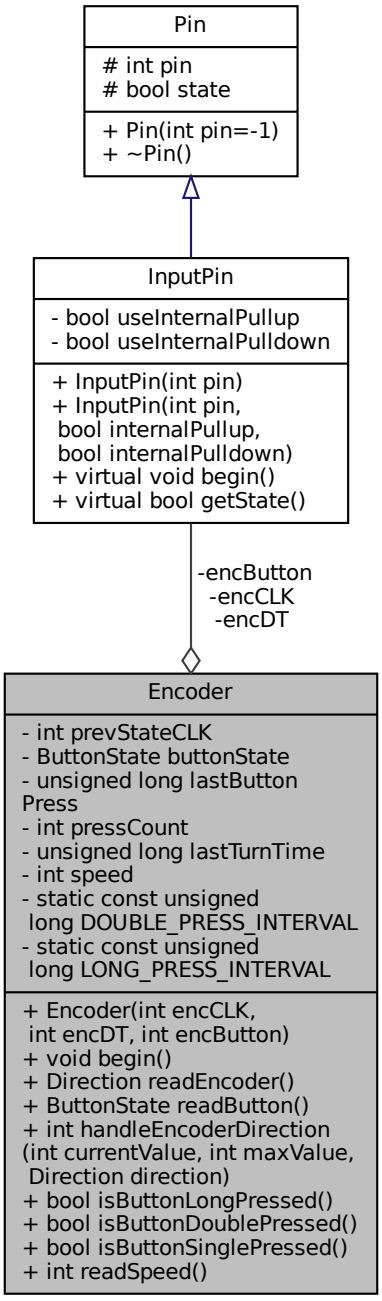
The documentation for this class was generated from the following files:

- include/Debug.h
- src/Debug.cpp

## 4.5 Encoder Class Reference

`#include <Encoder.h>`

Collaboration diagram for Encoder:




### Public Types

- enum Direction { NONE , CW , CCW }
- enum ButtonState { OPEN , PRESSED }

### Public Member Functions

- Encoder (int encCLK, int encDT, int encButton)
- void begin ()
- Direction readEncoder ()
- ButtonState readButton ()
- int handleEncoderDirection (int currentValue, int maxValue, Direction direction)
- bool isButtonLongPressed ()
- bool isButtonDoublePressed ()
- bool isButtonSinglePressed ()
- int readSpeed ()

### Private Attributes

- InputPin encCLK
- InputPin encDT
- InputPin encButton
- int prevStateCLK
- ButtonState buttonState
- unsigned long lastButtonPress
- int pressCount
- unsigned long lastTurnTime
- int speed

### Static Private Attributes

- static const unsigned long DOUBLE_PRESS_INTERVAL = 500
- static const unsigned long LONG_PRESS_INTERVAL = 1000

## 4.5.1 Member Enumeration Documentation

#### 4.5.1.1 ButtonState

`enum Encoder::ButtonState`

**Enumerator**

| | |
|---|---|
| OPEN | |
| PRESSED | |

#### 4.5.1.2 Direction

`enum Encoder::Direction`

**Enumerator**

| | |
|---|---|
| NONE | |
| CW | |
| CCW | |

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 Encoder()

```
Encoder::Encoder (
            int encCLK,
            int encDT,
            int encButton )
```

### 4.5.3 Member Function Documentation

#### 4.5.3.1 begin()

```
void Encoder::begin ( )
```

#### 4.5.3.2 handleEncoderDirection()

```
int Encoder::handleEncoderDirection (
            int currentValue,
            int maxValue,
            Direction direction )
```

#### 4.5.3.3 isButtonDoublePressed()

```
bool Encoder::isButtonDoublePressed ( )
```

#### 4.5.3.4 isButtonLongPressed()

```
bool Encoder::isButtonLongPressed ( )
```

#### 4.5.3.5 isButtonSinglePressed()

```
bool Encoder::isButtonSinglePressed ( )
```

#### 4.5.3.6 readButton()

```
Encoder::ButtonState Encoder::readButton ( )
```

#### 4.5.3.7 readEncoder()

```
Encoder::Direction Encoder::readEncoder ( )
```

#### 4.5.3.8 readSpeed()

```
int Encoder::readSpeed ( )
```

### 4.5.4 Member Data Documentation

#### 4.5.4.1 buttonState

```
ButtonState Encoder::buttonState  [private]
```

#### 4.5.4.2 DOUBLE_PRESS_INTERVAL

```
const unsigned long Encoder::DOUBLE_PRESS_INTERVAL = 500  [static], [private]
```

#### 4.5.4.3 encButton

```
InputPin Encoder::encButton  [private]
```

#### 4.5.4.4 encCLK

```
InputPin Encoder::encCLK  [private]
```

#### 4.5.4.5 encDT

```
InputPin Encoder::encDT  [private]
```

#### 4.5.4.6 lastButtonPress

```
unsigned long Encoder::lastButtonPress  [private]
```

#### 4.5.4.7 lastTurnTime

```
unsigned long Encoder::lastTurnTime  [private]
```

#### 4.5.4.8 LONG_PRESS_INTERVAL

```
const unsigned long Encoder::LONG_PRESS_INTERVAL = 1000  [static], [private]
```

#### 4.5.4.9 pressCount

```
int Encoder::pressCount  [private]
```

#### 4.5.4.10 prevStateCLK

```
int Encoder::prevStateCLK  [private]
```

#### 4.5.4.11 speed

```
int Encoder::speed  [private]
```

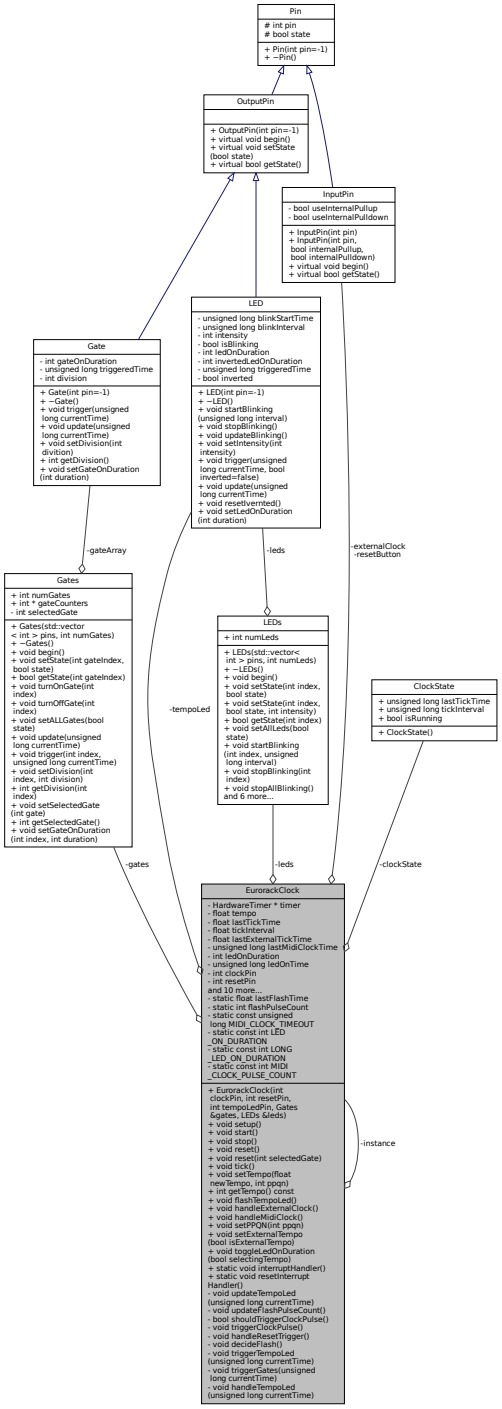
The documentation for this class was generated from the following files:

- include/Encoder.h
- src/Encoder.cpp

## 4.6 EurorackClock Class Reference

```
#include <EurorackClock.h>
```

Collaboration diagram for EurorackClock:




## Public Member Functions

- EurorackClock (int clockPin, int resetPin, int tempoLedPin, Gates &gates, LEDs &leds)
- void setup ()
- void start ()
- void stop ()
- void reset ()

- void reset (int selectedGate)
- void tick ()
- void setTempo (float newTempo, int ppqn)
- int getTempo () const
- void flashTempoLed ()
- void handleExternalClock ()
- void handleMidiClock ()
- void setPPQN (int ppqn)
- void setExternalTempo (bool isExternalTempo)
- void toggleLedOnDuration (bool selectingTempo)

## Static Public Member Functions

- static void interruptHandler ()
- static void resetInterruptHandler ()

## Private Member Functions

- void updateTempoLed (unsigned long currentTime)
- void updateFlashPulseCount ()
- bool shouldTriggerClockPulse ()
- void triggerClockPulse ()
- void handleResetTrigger ()
- void decideFlash ()
- void triggerTempoLed (unsigned long currentTime)
- void triggerGates (unsigned long currentTime)
- void handleTempoLed (unsigned long currentTime)

## Private Attributes

- ClockState clockState
- HardwareTimer * timer
- LED tempoLed
- InputPin externalClock
- InputPin resetButton
- Gates & gates
- LEDs & leds
- float tempo
- float lastTickTime
- float tickInterval
- float lastExternalTickTime
- unsigned long lastMidiClockTime
- int ledOnDuration = LONG_LED_ON_DURATION
- unsigned long ledOnTime = 0
- int clockPin
- int resetPin
- int ppqn
- bool isRunning
- bool isExternalTempo
- bool isMidiClock
- bool timeToFlash
- bool resetTriggered
- float externalTempo
- int lastClockState
- unsigned long lastClockTime
- int tickCount

## Static Private Attributes

- static EurorackClock * instance = nullptr
- static float lastFlashTime = 0
- static int flashPulseCount = 0
- static const unsigned long MIDI_CLOCK_TIMEOUT = 1000
- static const int LED_ON_DURATION = 10
- static const int LONG_LED_ON_DURATION = 50
- static const int MIDI_CLOCK_PULSE_COUNT = 24

### 4.6.1 Constructor & Destructor Documentation

#### 4.6.1.1 EurorackClock()

```
EurorackClock::EurorackClock (
            int clockPin,
            int resetPin,
            int tempoLedPin,
            Gates & gates,
            LEDs & leds )
```

### 4.6.2 Member Function Documentation

#### 4.6.2.1 decideFlash()

```
void EurorackClock::decideFlash ( )  [private]
```

#### 4.6.2.2 flashTempoLed()

```
void EurorackClock::flashTempoLed ( )
```

#### 4.6.2.3 getTempo()

```
int EurorackClock::getTempo ( ) const
```

#### 4.6.2.4 handleExternalClock()

```
void EurorackClock::handleExternalClock ( )
```

#### 4.6.2.5 handleMidiClock()

```
void EurorackClock::handleMidiClock ( )
```

#### 4.6.2.6 handleResetTrigger()

```
void EurorackClock::handleResetTrigger ( )  [private]
```

#### 4.6.2.7 handleTempoLed()

```
void EurorackClock::handleTempoLed (
            unsigned long currentTime )  [private]
```

#### 4.6.2.8 interruptHandler()

```
static void EurorackClock::interruptHandler ( )  [inline], [static]
```

#### 4.6.2.9 reset() [1/2]

```
void EurorackClock::reset ( )
```

#### 4.6.2.10 reset() [2/2]

```
void EurorackClock::reset (
            int selectedGate )
```

#### 4.6.2.11 resetInterruptHandler()

```
static void EurorackClock::resetInterruptHandler ( )  [inline], [static]
```

#### 4.6.2.12 setExternalTempo()

```
void EurorackClock::setExternalTempo (
            bool isExternalTempo )
```

#### 4.6.2.13 setPPQN()

```
void EurorackClock::setPPQN (
            int ppqn )
```

#### 4.6.2.14 setTempo()

```
void EurorackClock::setTempo (
            float newTempo,
            int ppqn )
```

#### 4.6.2.15 setup()

```
void EurorackClock::setup ( )
```

#### 4.6.2.16 shouldTriggerClockPulse()

```
bool EurorackClock::shouldTriggerClockPulse ( )  [private]
```

#### 4.6.2.17 start()

```
void EurorackClock::start ( )
```

#### 4.6.2.18 stop()

```
void EurorackClock::stop ( )
```

#### 4.6.2.19 tick()

```
void EurorackClock::tick ( )
```

#### 4.6.2.20 toggleLedOnDuration()

```
void EurorackClock::toggleLedOnDuration (
            bool selectingTempo )
```

#### 4.6.2.21 triggerClockPulse()

```
void EurorackClock::triggerClockPulse ( )  [private]
```

#### 4.6.2.22 triggerGates()

```
void EurorackClock::triggerGates (
            unsigned long currentTime )  [private]
```

#### 4.6.2.23 triggerTempoLed()

```
void EurorackClock::triggerTempoLed (
            unsigned long currentTime )  [private]
```

#### 4.6.2.24 updateFlashPulseCount()

```
void EurorackClock::updateFlashPulseCount ( )  [private]
```

#### 4.6.2.25 updateTempoLed()

```
void EurorackClock::updateTempoLed (
            unsigned long currentTime )  [private]
```

### 4.6.3 Member Data Documentation

#### 4.6.3.1 clockPin

```
int EurorackClock::clockPin  [private]
```

#### 4.6.3.2 clockState

```
ClockState EurorackClock::clockState  [private]
```

#### 4.6.3.3 externalClock

```
InputPin EurorackClock::externalClock  [private]
```

#### 4.6.3.4 externalTempo

```
float EurorackClock::externalTempo  [private]
```

#### 4.6.3.5 flashPulseCount

```
int EurorackClock::flashPulseCount = 0  [static], [private]
```

#### 4.6.3.6 gates

```
Gates& EurorackClock::gates  [private]
```

#### 4.6.3.7 instance

```
EurorackClock * EurorackClock::instance = nullptr  [static], [private]
```

#### 4.6.3.8 isExternalTempo

```
bool EurorackClock::isExternalTempo  [private]
```

#### 4.6.3.9 isMidiClock

```
bool EurorackClock::isMidiClock  [private]
```

#### 4.6.3.10 isRunning

```
bool EurorackClock::isRunning  [private]
```

#### 4.6.3.11 lastClockState

```
int EurorackClock::lastClockState  [private]
```

#### 4.6.3.12 lastClockTime

```
unsigned long EurorackClock::lastClockTime  [private]
```

#### 4.6.3.13 lastExternalTickTime

```
float EurorackClock::lastExternalTickTime  [private]
```

#### 4.6.3.14 lastFlashTime

```
float EurorackClock::lastFlashTime = 0  [static], [private]
```

#### 4.6.3.15 lastMidiClockTime

```
unsigned long EurorackClock::lastMidiClockTime  [private]
```

#### 4.6.3.16 lastTickTime

```
float EurorackClock::lastTickTime  [private]
```

#### 4.6.3.17 LED_ON_DURATION

```
const int EurorackClock::LED_ON_DURATION = 10  [static], [private]
```

#### 4.6.3.18 ledOnDuration

```
int EurorackClock::ledOnDuration = LONG_LED_ON_DURATION  [private]
```

#### 4.6.3.19 ledOnTime

```
unsigned long EurorackClock::ledOnTime = 0  [private]
```

#### 4.6.3.20 leds

```
LEDs& EurorackClock::leds  [private]
```

#### 4.6.3.21 LONG_LED_ON_DURATION

```
const int EurorackClock::LONG_LED_ON_DURATION = 50  [static], [private]
```

#### 4.6.3.22 MIDI_CLOCK_PULSE_COUNT

```
const int EurorackClock::MIDI_CLOCK_PULSE_COUNT = 24  [static], [private]
```

#### 4.6.3.23 MIDI_CLOCK_TIMEOUT

```
const unsigned long EurorackClock::MIDI_CLOCK_TIMEOUT = 1000  [static], [private]
```

#### 4.6.3.24 ppqn

```
int EurorackClock::ppqn  [private]
```

#### 4.6.3.25 resetButton

```
InputPin EurorackClock::resetButton  [private]
```

#### 4.6.3.26 resetPin

```
int EurorackClock::resetPin  [private]
```

#### 4.6.3.27 resetTriggered

```
bool EurorackClock::resetTriggered  [private]
```

#### 4.6.3.28 tempo

```
float EurorackClock::tempo  [private]
```

#### 4.6.3.29 tempoLed

```
LED EurorackClock::tempoLed  [private]
```

#### 4.6.3.30 tickCount

```
int EurorackClock::tickCount  [private]
```

#### 4.6.3.31 tickInterval

```
float EurorackClock::tickInterval  [private]
```

#### 4.6.3.32 timer

```
HardwareTimer* EurorackClock::timer  [private]
```

#### 4.6.3.33 timeToFlash

```
bool EurorackClock::timeToFlash  [private]
```

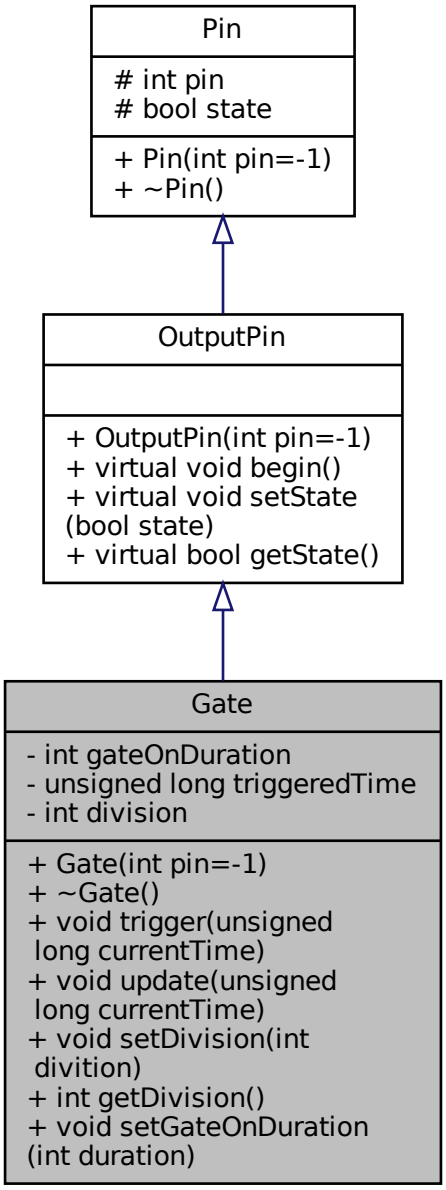
The documentation for this class was generated from the following files:

- include/EurorackClock.h
- src/EurorackClock.cpp

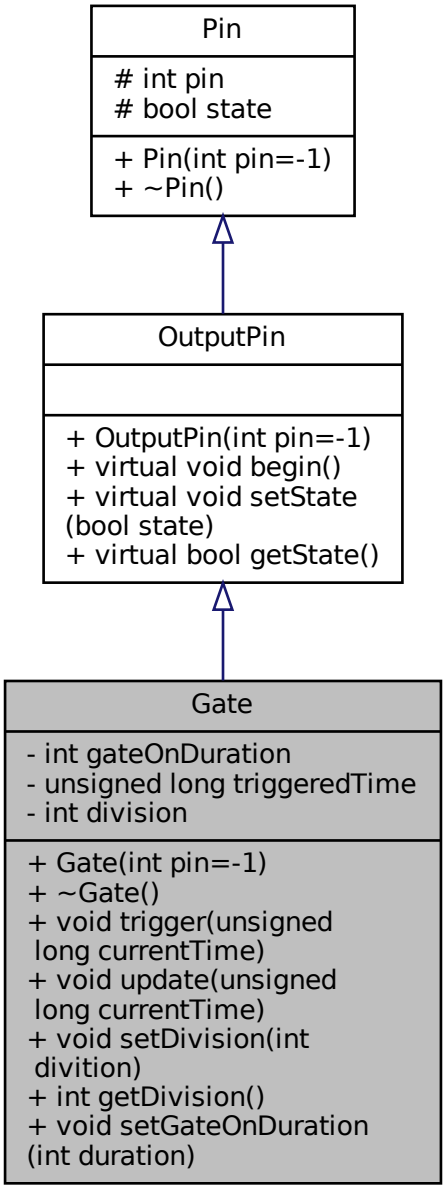
## 4.7 Gate Class Reference

```
#include <Gate.h>
```

Inheritance diagram for Gate:



Collaboration diagram for Gate:




## Public Member Functions

- Gate (int pin=-1)
- ~Gate ()
- void trigger (unsigned long currentTime)
- void update (unsigned long currentTime)
- void setDivision (int divition)
- int getDivision ()
- void setGateOnDuration (int duration)

### Private Attributes

- int gateOnDuration = 10
- unsigned long triggeredTime = 0
- int division = internalPPQN

### Additional Inherited Members

## 4.7.1 Constructor & Destructor Documentation

#### 4.7.1.1 Gate()

```
Gate::Gate (
            int pin = -1 )
```

#### 4.7.1.2 ~Gate()

```
Gate::~Gate ( )
```

## 4.7.2 Member Function Documentation

#### 4.7.2.1 getDivision()

```
int Gate::getDivision ( )
```

#### 4.7.2.2 setDivision()

```
void Gate::setDivision (
            int divition )
```

#### 4.7.2.3 setGateOnDuration()

```
void Gate::setGateOnDuration (
            int duration )
```

#### 4.7.2.4 trigger()

```
void Gate::trigger (
            unsigned long currentTime )
```

#### 4.7.2.5 update()

```
void Gate::update (
            unsigned long currentTime )
```

### 4.7.3 Member Data Documentation

#### 4.7.3.1 division

```
int Gate::division = internalPPQN  [private]
```

#### 4.7.3.2 gateOnDuration

```
int Gate::gateOnDuration = 10  [private]
```

#### 4.7.3.3 triggeredTime

```
unsigned long Gate::triggeredTime = 0  [private]
```

The documentation for this class was generated from the following files:

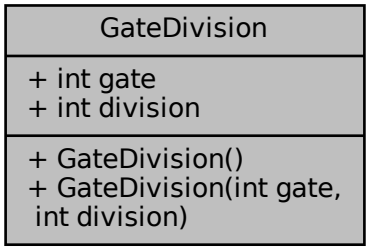
- include/Gate.h
- src/Gate.cpp

## 4.8 GateDivision Struct Reference

This is a global struct that holds the state of the application. It mainly holds items that need to persist after a power cycle. The object is initialized managed by the StateManager class.

`#include <AppState.h>`

Collaboration diagram for GateDivision:




### Public Member Functions

- GateDivision ()
- GateDivision (int gate, int division)

### Public Attributes

- int gate
- int division

### 4.8.1 Detailed Description

This is a global struct that holds the state of the application. It mainly holds items that need to persist after a power cycle. The object is initialized managed by the StateManager class.

The fefault values are set in the StateManager class, in the initializeEEPROM() function. To avoid issues with the EEPROM memory, make sure you initialize all values in the StateManager class.

This object is updated through out the app, however saving to EEPROM is only done when the app is in mode selection mode. It saves when long pressing and also when the mode is successfully changed.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 GateDivision() [1/2]

```
GateDivision::GateDivision ( )  [inline]
```

#### 4.8.2.2 GateDivision() [2/2]

```
GateDivision::GateDivision (
            int gate,
            int division )  [inline]
```

### 4.8.3 Member Data Documentation

#### 4.8.3.1 division

```
int GateDivision::division
```

#### 4.8.3.2 gate

```
int GateDivision::gate
```

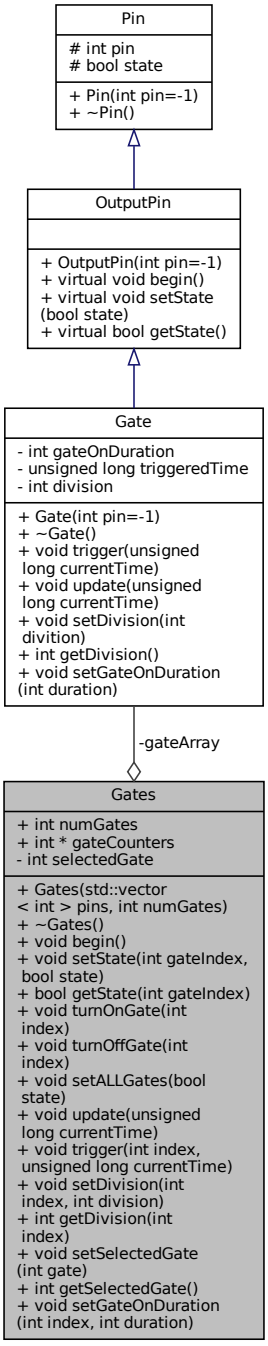
The documentation for this struct was generated from the following file:

- include/AppState.h

## 4.9 Gates Class Reference

```
#include <Gates.h>
```

Collaboration diagram for Gates:




## Public Member Functions

- Gates (std::vector< int > pins, int numGates)
- ~Gates ()
- void begin ()
- void setState (int gateIndex, bool state)
- bool getState (int gateIndex)

- void turnOnGate (int index)
- void turnOffGate (int index)
- void setALLGates (bool state)
- void update (unsigned long currentTime)
- void trigger (int index, unsigned long currentTime)
- void setDivision (int index, int division)
- int getDivision (int index)
- void setSelectedGate (int gate)
- int getSelectedGate ()
- void setGateOnDuration (int index, int duration)

## Public Attributes

- int numGates
- int * gateCounters

## Private Attributes

- Gate * gateArray
- int selectedGate

### 4.9.1 Constructor & Destructor Documentation

#### 4.9.1.1 Gates()

```
Gates::Gates (
            std::vector< int > pins,
            int numGates )
```

#### 4.9.1.2 ~Gates()

```
Gates::~Gates ( )
```

### 4.9.2 Member Function Documentation

#### 4.9.2.1 begin()

```
void Gates::begin ( )
```

#### 4.9.2.2 getDivision()

```
int Gates::getDivision (
            int index )
```

#### 4.9.2.3 getSelectedGate()

```
int Gates::getSelectedGate ( )
```

#### 4.9.2.4 getState()

```
bool Gates::getState (
            int gateIndex )
```

#### 4.9.2.5 setALLGates()

```
void Gates::setALLGates (
            bool state )
```

#### 4.9.2.6 setDivision()

```
void Gates::setDivision (
            int index,
            int division )
```

#### 4.9.2.7 setGateOnDuration()

```
void Gates::setGateOnDuration (
            int index,
            int duration )
```

#### 4.9.2.8 setSelectedGate()

```
void Gates::setSelectedGate (
            int gate )
```

#### 4.9.2.9 setState()

```
void Gates::setState (
            int gateIndex,
            bool state )
```

#### 4.9.2.10 trigger()

```
void Gates::trigger (
            int index,
            unsigned long currentTime )
```

#### 4.9.2.11 turnOffGate()

```
void Gates::turnOffGate (
            int index )
```

#### 4.9.2.12 turnOnGate()

```
void Gates::turnOnGate (
            int index )
```

#### 4.9.2.13 update()

```
void Gates::update (
            unsigned long currentTime )
```

### 4.9.3 Member Data Documentation

#### 4.9.3.1 gateArray

```
Gate* Gates::gateArray  [private]
```

#### 4.9.3.2 gateCounters

```
int* Gates::gateCounters
```

#### 4.9.3.3 numGates

```
int Gates::numGates
```

#### 4.9.3.4 selectedGate

```
int Gates::selectedGate  [private]
```

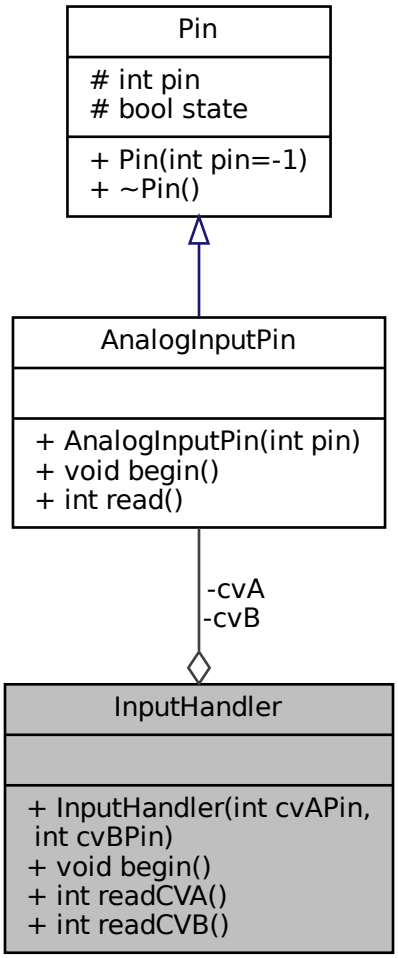
The documentation for this class was generated from the following files:

- include/Gates.h
- src/Gates.cpp

## 4.10 InputHandler Class Reference

```
#include <InputHandler.h>
```

Collaboration diagram for InputHandler:




## Public Member Functions

- InputHandler (int cvAPin, int cvBPin)
- void begin ()
- int readCVA ()
- int readCVB ()

## Private Attributes

- AnalogInputPin cvA
- AnalogInputPin cvB

### 4.10.1 Constructor & Destructor Documentation

#### 4.10.1.1 InputHandler()

```
InputHandler::InputHandler (
            int cvAPin,
            int cvBPin )
```

### 4.10.2 Member Function Documentation

#### 4.10.2.1 begin()

```
void InputHandler::begin ( )
```

#### 4.10.2.2 readCVA()

```
int InputHandler::readCVA ( )
```

#### 4.10.2.3 readCVB()

```
int InputHandler::readCVB ( )
```

### 4.10.3 Member Data Documentation

#### 4.10.3.1 cvA

```
AnalogInputPin InputHandler::cvA  [private]
```

#### 4.10.3.2 cvB

```
AnalogInputPin InputHandler::cvB  [private]
```

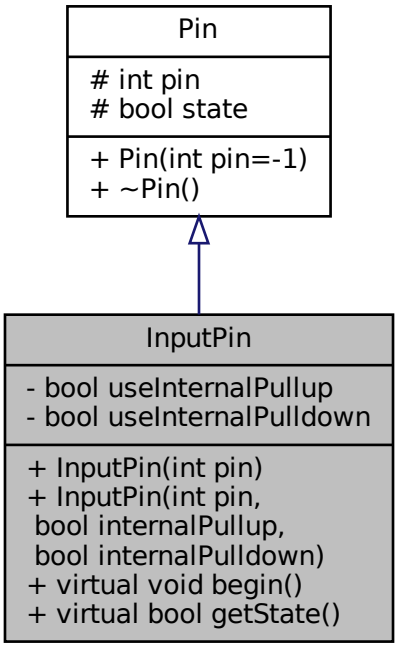
The documentation for this class was generated from the following files:

- include/InputHandler.h
- src/InputHandler.cpp

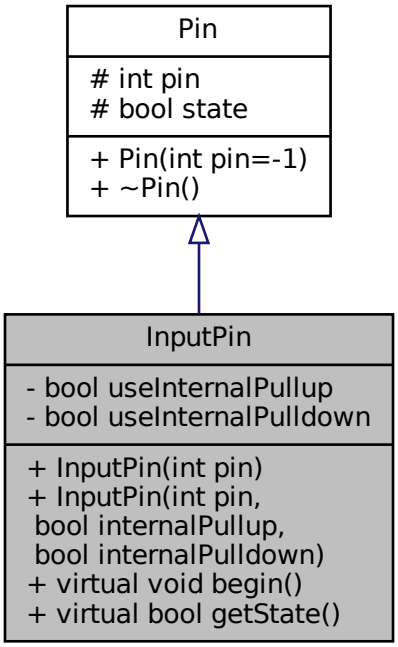
## 4.11 InputPin Class Reference

`#include <Pin.h>`

Inheritance diagram for InputPin:



Collaboration diagram for InputPin:




## Public Member Functions

- InputPin (int pin)
- InputPin (int pin, bool internalPullup, bool internalPulldown)
- virtual void begin ()
- virtual bool getState ()

## Private Attributes

- bool useInternalPullup
- bool useInternalPulldown

## Additional Inherited Members

### 4.11.1 Constructor & Destructor Documentation

#### 4.11.1.1 InputPin() [1/2]

```
InputPin::InputPin (
            int pin )
```

#### 4.11.1.2 InputPin() [2/2]

```
InputPin::InputPin (
            int pin,
            bool internalPullup,
            bool internalPulldown )
```

### 4.11.2 Member Function Documentation

#### 4.11.2.1 begin()

```
void InputPin::begin ( )  [virtual]
```

#### 4.11.2.2 getState()

```
bool InputPin::getState ( )  [virtual]
```

### 4.11.3 Member Data Documentation

#### 4.11.3.1 useInternalPulldown

```
bool InputPin::useInternalPulldown  [private]
```

#### 4.11.3.2 useInternalPullup

```
bool InputPin::useInternalPullup  [private]
```

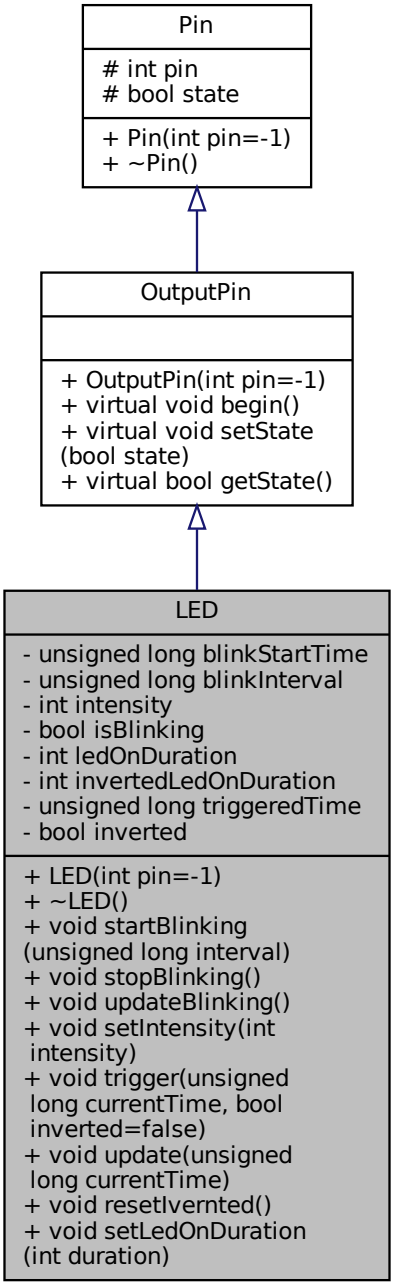
The documentation for this class was generated from the following files:

- include/Pin.h
- src/Pin.cpp

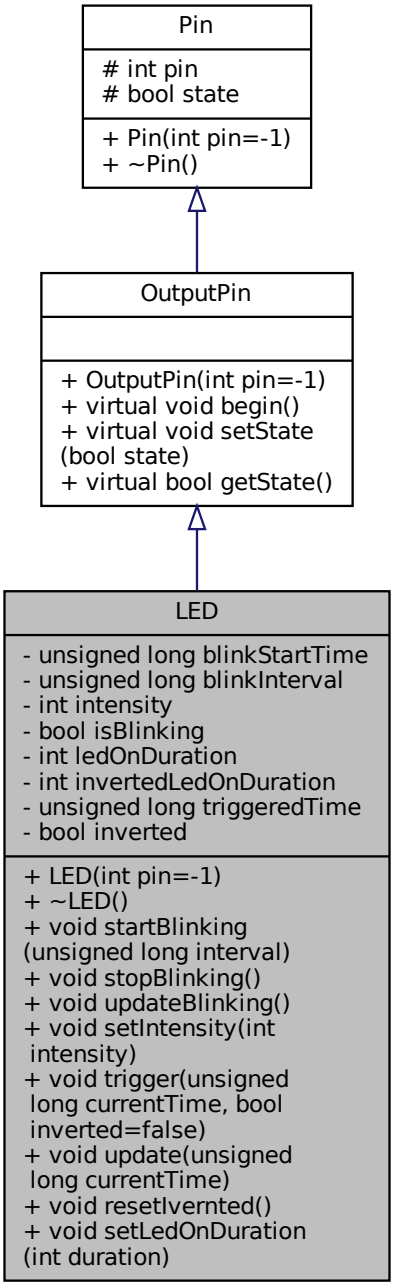
## 4.12 LED Class Reference

`#include <LED.h>`

Inheritance diagram for LED:



Collaboration diagram for LED:




## Public Member Functions

- LED (int pin=-1)
- ~LED ()
- void startBlinking (unsigned long interval)
- void stopBlinking ()
- void updateBlinking ()

- void setIntensity (int intensity)
- void trigger (unsigned long currentTime, bool inverted=false)
- void update (unsigned long currentTime)
- void resetIvernted ()
- void setLedOnDuration (int duration)

### Private Attributes

- unsigned long blinkStartTime
- unsigned long blinkInterval
- int intensity = 255
- bool isBlinking
- int ledOnDuration = 25
- int invertedLedOnDuration = 40
- unsigned long triggeredTime = 0
- bool inverted = false

### Additional Inherited Members

## 4.12.1 Constructor & Destructor Documentation

#### 4.12.1.1 LED()

```
LED::LED (
              int pin = -1 )
```

#### 4.12.1.2 ~LED()

```
LED::~LED ( )
```

## 4.12.2 Member Function Documentation

#### 4.12.2.1 resetIvernted()

```
void LED::resetIvernted ( )
```

#### 4.12.2.2 setIntensity()

```
void LED::setIntensity (
            int intensity )
```

#### 4.12.2.3 setLedOnDuration()

```
void LED::setLedOnDuration (
            int duration )
```

#### 4.12.2.4 startBlinking()

```
void LED::startBlinking (
            unsigned long interval )
```

#### 4.12.2.5 stopBlinking()

```
void LED::stopBlinking ( )
```

#### 4.12.2.6 trigger()

```
void LED::trigger (
            unsigned long currentTime,
            bool inverted = false )
```

#### 4.12.2.7 update()

```
void LED::update (
            unsigned long currentTime )
```

#### 4.12.2.8 updateBlinking()

```
void LED::updateBlinking ( )
```

### 4.12.3 Member Data Documentation

#### 4.12.3.1 blinkInterval

```
unsigned long LED::blinkInterval  [private]
```

#### 4.12.3.2 blinkStartTime

```
unsigned long LED::blinkStartTime  [private]
```

#### 4.12.3.3 intensity

```
int LED::intensity = 255  [private]
```

#### 4.12.3.4 inverted

```
bool LED::inverted = false  [private]
```

#### 4.12.3.5 invertedLedOnDuration

```
int LED::invertedLedOnDuration = 40  [private]
```

#### 4.12.3.6 isBlinking

```
bool LED::isBlinking  [private]
```

#### 4.12.3.7 ledOnDuration

```
int LED::ledOnDuration = 25  [private]
```

#### 4.12.3.8 triggeredTime

```
unsigned long LED::triggeredTime = 0  [private]
```

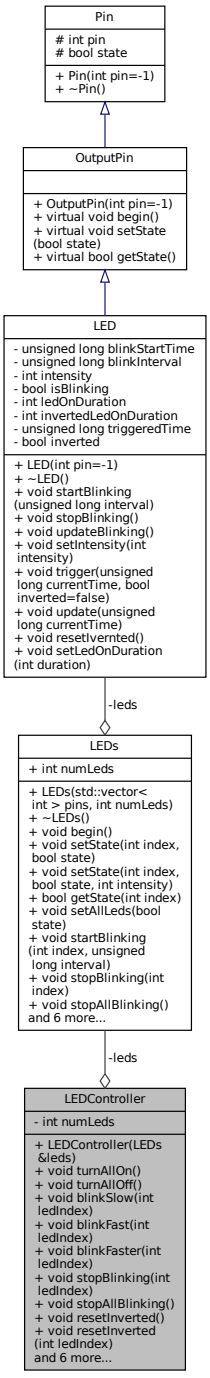
The documentation for this class was generated from the following files:

- include/LED.h
- src/LED.cpp

## 4.13 LEDController Class Reference

```
#include <LEDController.h>
```

Collaboration diagram for LEDController:




## Public Member Functions

- LEDController (LEDs &leds)
- void turnAllOn ()
- void turnAllOff ()
- void blinkSlow (int ledIndex)
- void blinkFast (int ledIndex)

- void blinkFaster (int ledIndex)
- void stopBlinking (int ledIndex)
- void stopAllBlinking ()
- void resetInverted ()
- void resetInverted (int ledIndex)
- int getNumLeds ()
- void update ()
- void clearAndResetLEDs ()
- void clearLEDs ()
- void updateBlinking ()
- void setState (int ledIndex, bool state)

## Private Attributes

- LEDs & leds
- int numLeds

### 4.13.1 Constructor & Destructor Documentation

#### 4.13.1.1 LEDController()

```
LEDController::LEDController (
            LEDs & leds )
```

### 4.13.2 Member Function Documentation

#### 4.13.2.1 blinkFast()

```
void LEDController::blinkFast (
            int ledIndex )
```

#### 4.13.2.2 blinkFaster()

```
void LEDController::blinkFaster (
            int ledIndex )
```

#### 4.13.2.3 blinkSlow()

```
void LEDController::blinkSlow (
            int ledIndex )
```

#### 4.13.2.4 clearAndResetLEDs()

```
void LEDController::clearAndResetLEDs ( )
```

#### 4.13.2.5 clearLEDs()

```
void LEDController::clearLEDs ( )
```

#### 4.13.2.6 getNumLeds()

```
int LEDController::getNumLeds ( )
```

#### 4.13.2.7 resetInverted() [1/2]

```
void LEDController::resetInverted ( )
```

#### 4.13.2.8 resetInverted() [2/2]

```
void LEDController::resetInverted (
            int ledIndex )
```

#### 4.13.2.9 setState()

```
void LEDController::setState (
            int ledIndex,
            bool state )
```

#### 4.13.2.10 stopAllBlinking()

```
void LEDController::stopAllBlinking ( )
```

#### 4.13.2.11 stopBlinking()

```
void LEDController::stopBlinking (
            int ledIndex )
```

#### 4.13.2.12 turnAllOff()

```
void LEDController::turnAllOff ( )
```

#### 4.13.2.13 turnAllOn()

```
void LEDController::turnAllOn ( )
```

#### 4.13.2.14 update()

```
void LEDController::update ( )
```

#### 4.13.2.15 updateBlinking()

```
void LEDController::updateBlinking ( )
```

### 4.13.3 Member Data Documentation

#### 4.13.3.1 leds

```
LEDs& LEDController::leds  [private]
```

#### 4.13.3.2 numLeds

```
int LEDController::numLeds  [private]
```

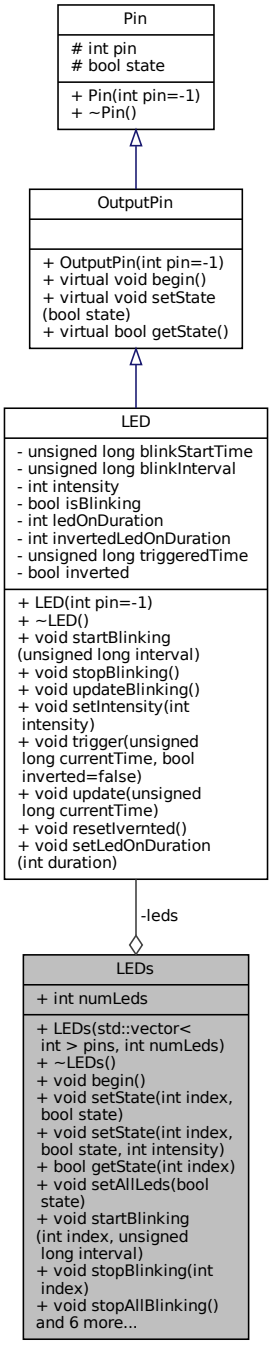
The documentation for this class was generated from the following files:

- include/LEDController.h
- src/LEDController.cpp

## 4.14 LEDs Class Reference

```
#include <LEDs.h>
```

Collaboration diagram for LEDs:




## Public Member Functions

- LEDs (std::vector< int > pins, int numLeds)
- ~LEDs ()
- void begin ()
- void setState (int index, bool state)
- void setState (int index, bool state, int intensity)

- bool getState (int index)
- void setAllLeds (bool state)
- void startBlinking (int index, unsigned long interval)
- void stopBlinking (int index)
- void stopAllBlinking ()
- void updateBlinking ()
- void setIntensity (int index, int intensity)
- void setAllintensity (int intensity)
- void update (unsigned long currentTime)
- void trigger (int index, unsigned long currentTime, bool inverted=false)
- void resetInverted (int index)

## Public Attributes

- int numLeds

## Private Attributes

- LED * leds

### 4.14.1 Constructor & Destructor Documentation

#### 4.14.1.1 LEDs()

```
LEDs::LEDs (
            std::vector< int > pins,
            int numLeds )
```

#### 4.14.1.2 ~LEDs()

```
LEDs::~LEDs ( )
```

### 4.14.2 Member Function Documentation

#### 4.14.2.1 begin()

```
void LEDs::begin ( )
```

#### 4.14.2.2 getState()

```
bool LEDs::getState (
            int index )
```

#### 4.14.2.3 resetInverted()

```
void LEDs::resetInverted (
            int index )
```

#### 4.14.2.4 setAllintensity()

```
void LEDs::setAllintensity (
            int intensity )
```

#### 4.14.2.5 setAllLeds()

```
void LEDs::setAllLeds (
            bool state )
```

#### 4.14.2.6 setIntensity()

```
void LEDs::setIntensity (
            int index,
            int intensity )
```

#### 4.14.2.7 setState() [1/2]

```
void LEDs::setState (
            int index,
            bool state )
```

#### 4.14.2.8 setState() [2/2]

```
void LEDs::setState (
            int index,
            bool state,
            int intensity )
```

#### 4.14.2.9 startBlinking()

```
void LEDs::startBlinking (
            int index,
            unsigned long interval )
```

#### 4.14.2.10 stopAllBlinking()

```
void LEDs::stopAllBlinking ( )
```

#### 4.14.2.11 stopBlinking()

```
void LEDs::stopBlinking (
            int index )
```

#### 4.14.2.12 trigger()

```
void LEDs::trigger (
            int index,
            unsigned long currentTime,
            bool inverted = false )
```

#### 4.14.2.13 update()

```
void LEDs::update (
            unsigned long currentTime )
```

#### 4.14.2.14 updateBlinking()

```
void LEDs::updateBlinking ( )
```

### 4.14.3 Member Data Documentation

#### 4.14.3.1 leds

```
LED* LEDs::leds  [private]
```

#### 4.14.3.2 numLeds

```
int LEDs::numLeds
```

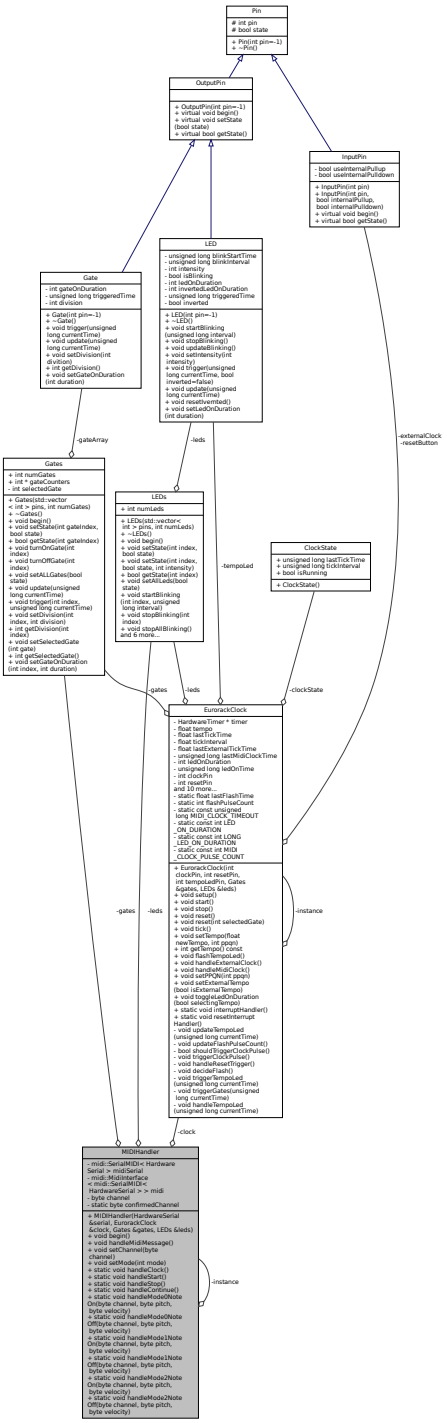
The documentation for this class was generated from the following files:

- include/LEDs.h
- src/LEDs.cpp

## 4.15 MIDIHandler Class Reference

```
#include <MIDIHandler.h>
```

Collaboration diagram for MIDIHandler:




## Public Member Functions

- MIDIHandler (HardwareSerial &serial, EurorackClock &clock, Gates &gates, LEDs &leds)
- void begin ()
- void handleMidiMessage ()
- void setChannel (byte channel)
- void setMode (int mode)

## Static Public Member Functions

- static void handleClock ()
- static void handleStart ()
- static void handleStop ()
- static void handleContinue ()
- static void handleMode0NoteOn (byte channel, byte pitch, byte velocity)
- static void handleMode0NoteOff (byte channel, byte pitch, byte velocity)
- static void handleMode1NoteOn (byte channel, byte pitch, byte velocity)
- static void handleMode1NoteOff (byte channel, byte pitch, byte velocity)
- static void handleMode2NoteOn (byte channel, byte pitch, byte velocity)
- static void handleMode2NoteOff (byte channel, byte pitch, byte velocity)

## Private Attributes

- midi::SerialMIDI< HardwareSerial > midiSerial
- midi::MidiInterface< midi::SerialMIDI< HardwareSerial > > midi
- EurorackClock & clock
- byte channel = 10
- Gates & gates
- LEDs & leds

## Static Private Attributes

- static MIDIHandler * instance = nullptr
- static byte confirmedChannel = 9

### 4.15.1 Constructor & Destructor Documentation

#### 4.15.1.1 MIDIHandler()

```
MIDIHandler::MIDIHandler (
            HardwareSerial & serial,
            EurorackClock & clock,
            Gates & gates,
            LEDs & leds )
```

### 4.15.2 Member Function Documentation

#### 4.15.2.1 begin()

```
void MIDIHandler::begin ( )
```

#### 4.15.2.2 handleClock()

```
void MIDIHandler::handleClock ( )  [static]
```

#### 4.15.2.3 handleContinue()

```
void MIDIHandler::handleContinue ( )  [static]
```

#### 4.15.2.4 handleMidiMessage()

```
void MIDIHandler::handleMidiMessage ( )
```

#### 4.15.2.5 handleMode0NoteOff()

```
void MIDIHandler::handleMode0NoteOff (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.6 handleMode0NoteOn()

```
void MIDIHandler::handleMode0NoteOn (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.7 handleMode1NoteOff()

```
void MIDIHandler::handleMode1NoteOff (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.8 handleMode1NoteOn()

```
void MIDIHandler::handleMode1NoteOn (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.9 handleMode2NoteOff()

```
void MIDIHandler::handleMode2NoteOff (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.10 handleMode2NoteOn()

```
void MIDIHandler::handleMode2NoteOn (
            byte channel,
            byte pitch,
            byte velocity )  [static]
```

#### 4.15.2.11 handleStart()

```
void MIDIHandler::handleStart ( )  [static]
```

#### 4.15.2.12 handleStop()

```
void MIDIHandler::handleStop ( )  [static]
```

#### 4.15.2.13 setChannel()

```
void MIDIHandler::setChannel (
            byte channel )
```

#### 4.15.2.14 setMode()

```
void MIDIHandler::setMode (
            int mode )
```

### 4.15.3 Member Data Documentation

#### 4.15.3.1 channel

```
byte MIDIHandler::channel = 10  [private]
```

#### 4.15.3.2 clock

```
EurorackClock& MIDIHandler::clock  [private]
```

#### 4.15.3.3 confirmedChannel

```
byte MIDIHandler::confirmedChannel = 9  [static], [private]
```

#### 4.15.3.4 gates

```
Gates& MIDIHandler::gates  [private]
```

#### 4.15.3.5 instance

```
MIDIHandler * MIDIHandler::instance = nullptr  [static], [private]
```

#### 4.15.3.6 leds

```
LEDs& MIDIHandler::leds  [private]
```

#### 4.15.3.7 midi

```
midi::MidiInterface<midi::SerialMIDI<HardwareSerial> > MIDIHandler::midi  [private]
```

#### 4.15.3.8 midiSerial

```
midi::SerialMIDI<HardwareSerial> MIDIHandler::midiSerial  [private]
```

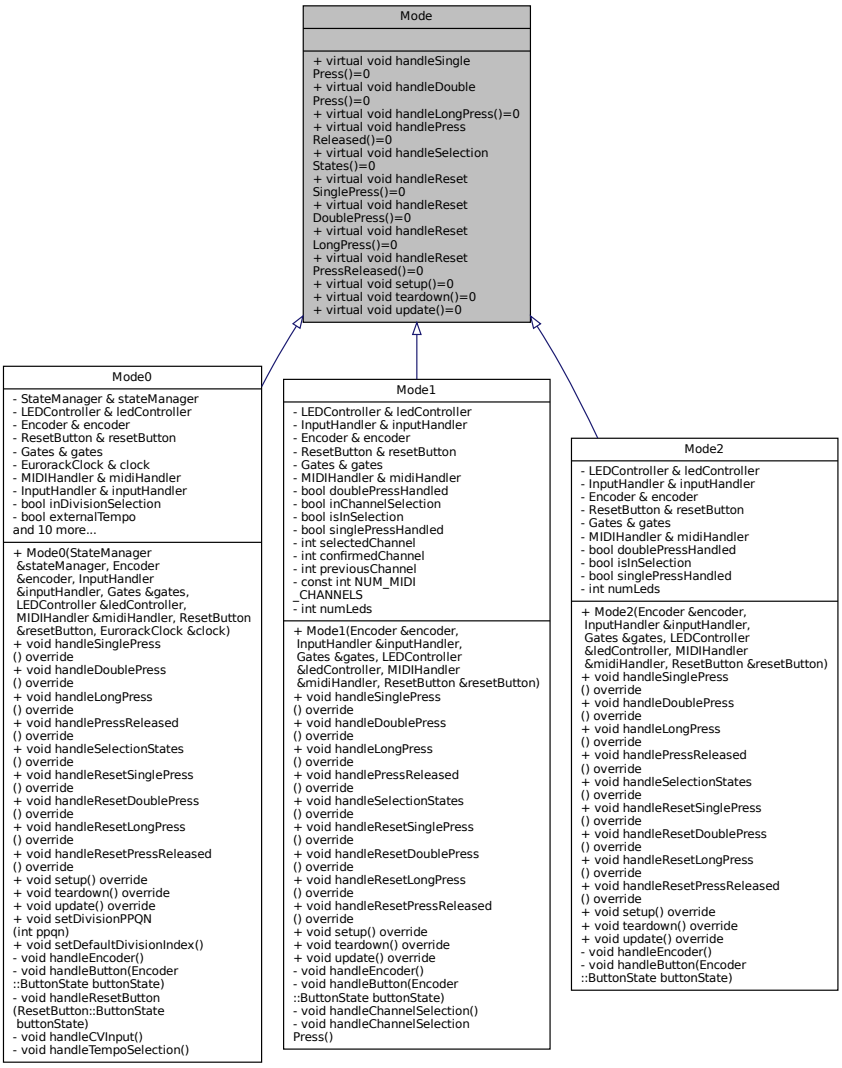
The documentation for this class was generated from the following files:

- include/MIDIHandler.h
- src/MIDIHandler.cpp

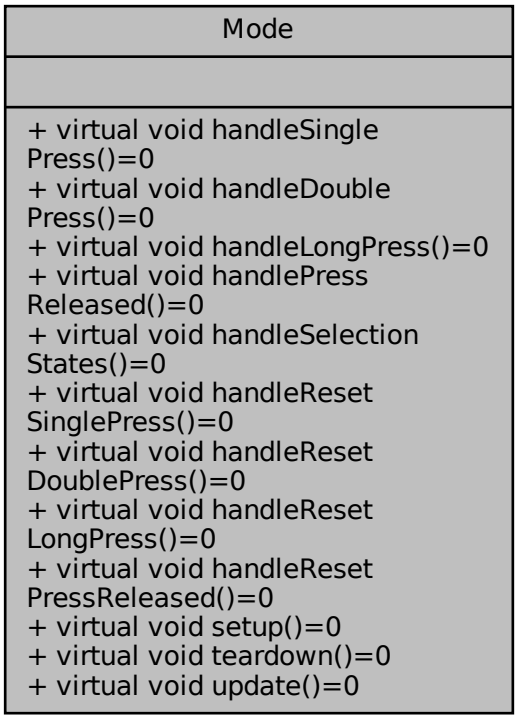
## 4.16 Mode Class Reference

```
#include <Mode.h>
```

Inheritance diagram for Mode:



Collaboration diagram for Mode:




## Public Member Functions

- virtual void handleSinglePress ()=0
- virtual void handleDoublePress ()=0
- virtual void handleLongPress ()=0
- virtual void handlePressReleased ()=0
- virtual void handleSelectionStates ()=0
- virtual void handleResetSinglePress ()=0
- virtual void handleResetDoublePress ()=0
- virtual void handleResetLongPress ()=0
- virtual void handleResetPressReleased ()=0
- virtual void setup ()=0
- virtual void teardown ()=0
- virtual void update ()=0

### 4.16.1 Member Function Documentation

#### 4.16.1.1 handleDoublePress()

```
virtual void Mode::handleDoublePress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.2 handleLongPress()

```
virtual void Mode::handleLongPress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.3 handlePressReleased()

```
virtual void Mode::handlePressReleased ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.4 handleResetDoublePress()

```
virtual void Mode::handleResetDoublePress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.5 handleResetLongPress()

```
virtual void Mode::handleResetLongPress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.6 handleResetPressReleased()

```
virtual void Mode::handleResetPressReleased ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.7 handleResetSinglePress()

```
virtual void Mode::handleResetSinglePress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.8 handleSelectionStates()

```
virtual void Mode::handleSelectionStates ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.9 handleSinglePress()

```
virtual void Mode::handleSinglePress ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.10 setup()

```
virtual void Mode::setup ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.11 teardown()

```
virtual void Mode::teardown ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

#### 4.16.1.12 update()

```
virtual void Mode::update ( )  [pure virtual]
```

Implemented in Mode2, Mode1, and Mode0.

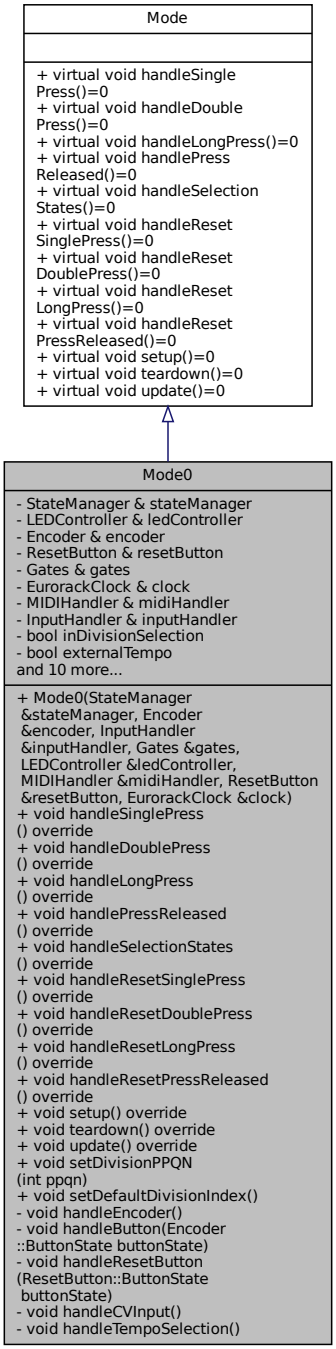
The documentation for this class was generated from the following file:

- include/Mode.h

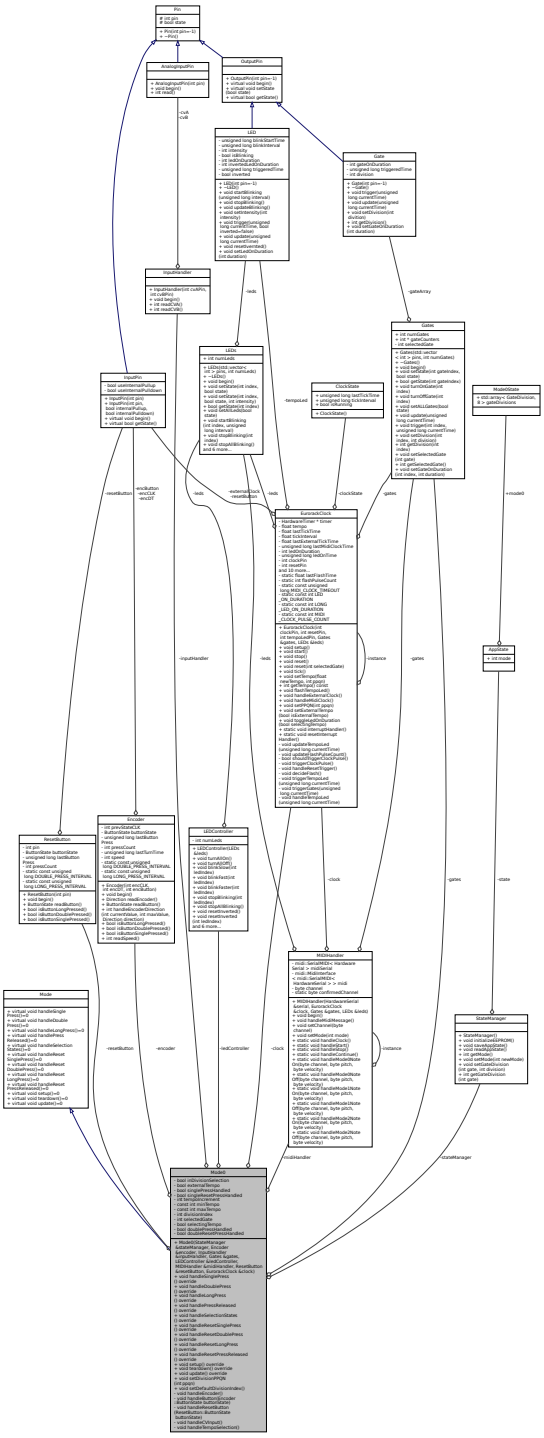
## 4.17 Mode0 Class Reference

`#include <Mode0.h>`

Inheritance diagram for Mode0:



Collaboration diagram for Mode0:



## Public Member Functions

- Mode0 (StateManager &stateManager, Encoder &encoder, InputHandler &inputHandler, Gates &gates, LEDController &ledController, MIDIHandler &midiHandler, ResetButton &resetButton, EurorackClock &clock)
- void handleSinglePress () override

  *Handle single press. Default behavior is to toggle between division selection and gate selection.*
- void handleDoublePress () override

*Handle double press. Default behavior is to enter or exit tempo selection mode.*

- void handleLongPress () override
- void handlePressReleased () override
- void handleSelectionStates () override

  *Handle selection states. Default behavior is to handle tempo selection.*
- void handleResetSinglePress () override

  *Handle reset single press. Default behavior is to reset the selected gate so it can by synced with the clock.*
- void handleResetDoublePress () override

  *Handle reset double press. Default behavior is to reset the clock so it can be synced with an external clock.*
- void handleResetLongPress () override
- void handleResetPressReleased () override
- void setup () override

  *Setup and teardown methods are meant to be called when Mode selector switches modes. This is where you can put code that should only run once when the mode is switched to. It is configured to run once when the mode is switched to and once when the mode is switched from.*
- void teardown () override

  *This block of code is executed once whenever we switch modes. The code here is intended to be cleanup code. This is where you can put code that should only run once when the mode is switched from.*
- void update () override

  *The update method is meant to be called every loop iteration. This is where you can put code that should run every loop iteration.*
- void setDivisionPPQN (int ppqn)
- void setDefaultDivisionIndex ()

  *Set the default division index based on the internal PPQN value, only used by the constructor to avoid compile errors.*

## Private Member Functions

- void handleEncoder ()

  *Detects the direction of the encoder and updates the selected gate or division based on the direction.*
- void handleButton (Encoder::ButtonState buttonState)

  *This block of code is used to handle button presses. It is called by the update method.*
- void handleResetButton (ResetButton::ButtonState buttonState)

  *This block of code is used to handle reset button presses. It is called by the update method.*
- void handleCVInput ()

  *block of code is here to handle inputs from the CV Input Jacks. It doesn't do anything now but is here for future use.*
- void handleTempoSelection ()

  *Handle tempo selection. Default behavior is to increase or decrease the tempo based on the encoder direction.*

## Private Attributes

- StateManager & stateManager
- LEDController & ledController
- Encoder & encoder
- ResetButton & resetButton
- Gates & gates
- EurorackClock & clock
- MIDIHandler & midiHandler
- InputHandler & inputHandler
- bool inDivisionSelection = false
- bool externalTempo = false
- bool singlePressHandled = false

- bool singleResetPressHandled = false
- int tempoIncrement = 1
- const int minTempo = 20
- const int maxTempo = 340
- int divisionIndex = 24
- int selectedGate = 0
- bool selectingTempo = false
- bool doublePressHandled = false
- bool doubleResetPressHandled = false

## 4.17.1 Constructor & Destructor Documentation

### 4.17.1.1 Mode0()

```
Mode0::Mode0 (
            StateManager & stateManager,
            Encoder & encoder,
            InputHandler & inputHandler,
            Gates & gates,
            LEDController & ledController,
            MIDIHandler & midiHandler,
            ResetButton & resetButton,
            EurorackClock & clock )
```

## 4.17.2 Member Function Documentation

### 4.17.2.1 handleButton()

```
void Mode0::handleButton (
            Encoder::ButtonState buttonState )  [private]
```

This block of code is used to handle button presses. It is called by the update method.

**Parameters**

| | |
|---|---|
| *buttonState* | |

### 4.17.2.2 handleCVInput()

```
void Mode0::handleCVInput ( )  [private]
```

block of code is here to handle inputs from the CV Input Jacks. It doesn't do anything now but is here for future use.

#### 4.17.2.3 handleDoublePress()

```
void Mode0::handleDoublePress ( )  [override], [virtual]
```

Handle double press. Default behavior is to enter or exit tempo selection mode.

Implements Mode.

#### 4.17.2.4 handleEncoder()

```
void Mode0::handleEncoder ( )  [private]
```

Detects the direction of the encoder and updates the selected gate or division based on the direction.

#### 4.17.2.5 handleLongPress()

```
void Mode0::handleLongPress ( )  [override], [virtual]
```

Long press is used by modeSelector, so don't use that here.

Implements Mode.

#### 4.17.2.6 handlePressReleased()

```
void Mode0::handlePressReleased ( )  [override], [virtual]
```

Mode 0 specific press released handling

Implements Mode.

#### 4.17.2.7 handleResetButton()

```
void Mode0::handleResetButton (
            ResetButton::ButtonState buttonState )  [private]
```

This block of code is used to handle reset button presses. It is called by the update method.

**Parameters**

| | |
|---|---|
| *buttonState* | |

#### 4.17.2.8 handleResetDoublePress()

```
void Mode0::handleResetDoublePress ( )  [override], [virtual]
```

Handle reset double press. Default behavior is to reset the clock so it can be synced with an external clock.

Implements Mode.

#### 4.17.2.9 handleResetLongPress()

```
void Mode0::handleResetLongPress ( )  [override], [virtual]
```

Does nothing yet but it could. :)

Implements Mode.

#### 4.17.2.10 handleResetPressReleased()

```
void Mode0::handleResetPressReleased ( )  [override], [virtual]
```

Does nothing yet but it could. :)

Implements Mode.

#### 4.17.2.11 handleResetSinglePress()

```
void Mode0::handleResetSinglePress ( )  [override], [virtual]
```

Handle reset single press. Default behavior is to reset the selected gate so it can by synced with the clock.

Implements Mode.

#### 4.17.2.12 handleSelectionStates()

```
void Mode0::handleSelectionStates ( )  [override], [virtual]
```

Handle selection states. Default behavior is to handle tempo selection.

Implements Mode.

#### 4.17.2.13 handleSinglePress()

```
void Mode0::handleSinglePress ( )  [override], [virtual]
```

Handle single press. Default behavior is to toggle between division selection and gate selection.

If in division selection update the division for the selected gate

Toggle between division selection and gate selection

Implements Mode.

#### 4.17.2.14 handleTempoSelection()

```
void Mode0::handleTempoSelection ( )  [private]
```

Handle tempo selection. Default behavior is to increase or decrease the tempo based on the encoder direction.

If externalTempo, exit external tempo mode and increase the tempo

Enter external tempo mode when the tempo reaches the minimum

#### 4.17.2.15 setDefaultDivisionIndex()

```
void Mode0::setDefaultDivisionIndex ( )
```

Set the default division index based on the internal PPQN value, only used by the constructor to avoid compile errors.

#### 4.17.2.16 setDivisionPPQN()

```
void Mode0::setDivisionPPQN (
            int ppqn )
```

#### 4.17.2.17 setup()

```
void Mode0::setup ( )  [override], [virtual]
```

Setup and teardown methods are meant to be called when Mode selector switches modes. This is where you can put code that should only run once when the mode is switched to. It is configured to run once when the mode is switched to and once when the mode is switched from.

Implements Mode.

#### 4.17.2.18 teardown()

```
void Mode0::teardown ( )  [override], [virtual]
```

This block of code is executed once whenever we switch modes. The code here is intended to be cleanup code. This is where you can put code that should only run once when the mode is switched from.

Implements Mode.

#### 4.17.2.19 update()

```
void Mode0::update ( )  [override], [virtual]
```

The update method is meant to be called every loop iteration. This is where you can put code that should run every loop iteration.

Implements Mode.

### 4.17.3 Member Data Documentation

#### 4.17.3.1 clock

```
EurorackClock& Mode0::clock  [private]
```

#### 4.17.3.2 divisionIndex

```
int Mode0::divisionIndex = 24  [private]
```

#### 4.17.3.3 doublePressHandled

```
bool Mode0::doublePressHandled = false  [private]
```

#### 4.17.3.4 doubleResetPressHandled

```
bool Mode0::doubleResetPressHandled = false  [private]
```

#### 4.17.3.5 encoder

```
Encoder& Mode0::encoder  [private]
```

#### 4.17.3.6 externalTempo

```
bool Mode0::externalTempo = false  [private]
```

#### 4.17.3.7 gates

```
Gates& Mode0::gates  [private]
```

#### 4.17.3.8 inDivisionSelection

```
bool Mode0::inDivisionSelection = false  [private]
```

#### 4.17.3.9 inputHandler

```
InputHandler& Mode0::inputHandler  [private]
```

#### 4.17.3.10 ledController

```
LEDController& Mode0::ledController  [private]
```

#### 4.17.3.11 maxTempo

```
const int Mode0::maxTempo = 340  [private]
```

#### 4.17.3.12 midiHandler

```
MIDIHandler& Mode0::midiHandler  [private]
```

#### 4.17.3.13 minTempo

```
const int Mode0::minTempo = 20  [private]
```

#### 4.17.3.14 resetButton

```
ResetButton& Mode0::resetButton  [private]
```

#### 4.17.3.15 selectedGate

```
int Mode0::selectedGate = 0  [private]
```

#### 4.17.3.16 selectingTempo

```
bool Mode0::selectingTempo = false  [private]
```

#### 4.17.3.17 singlePressHandled

```
bool Mode0::singlePressHandled = false  [private]
```

#### 4.17.3.18 singleResetPressHandled

```
bool Mode0::singleResetPressHandled = false  [private]
```

#### 4.17.3.19 stateManager

```
StateManager& Mode0::stateManager  [private]
```

#### 4.17.3.20 tempoIncrement

```
int Mode0::tempoIncrement = 1  [private]
```

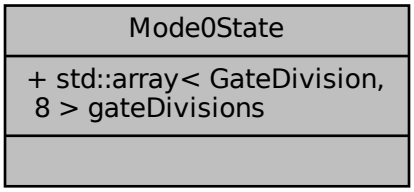
The documentation for this class was generated from the following files:

- include/Mode0.h
- src/Mode0.cpp

## 4.18 Mode0State Struct Reference

```
#include <AppState.h>
```

Collaboration diagram for Mode0State:




### Public Attributes

- std::array< GateDivision, 8 > gateDivisions

### 4.18.1 Member Data Documentation

#### 4.18.1.1 gateDivisions

```
std::array<GateDivision, 8> Mode0State::gateDivisions
```

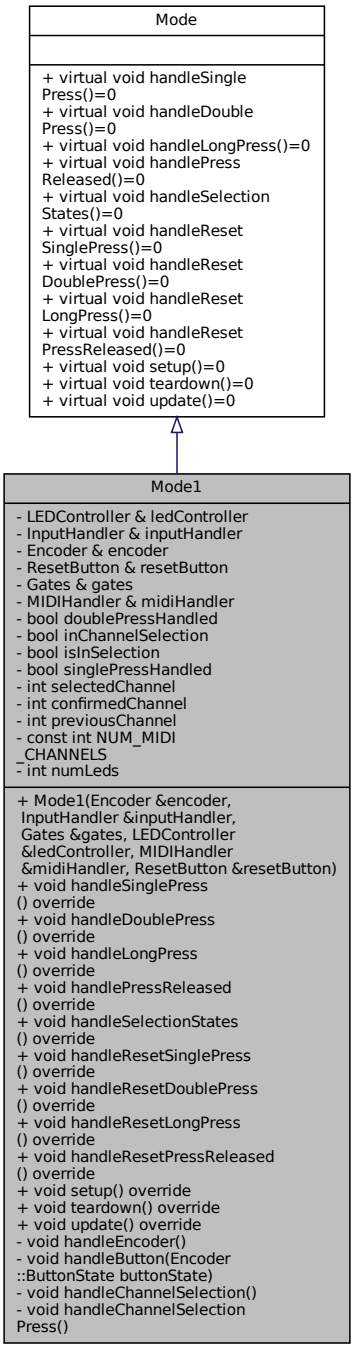
The documentation for this struct was generated from the following file:

- include/AppState.h

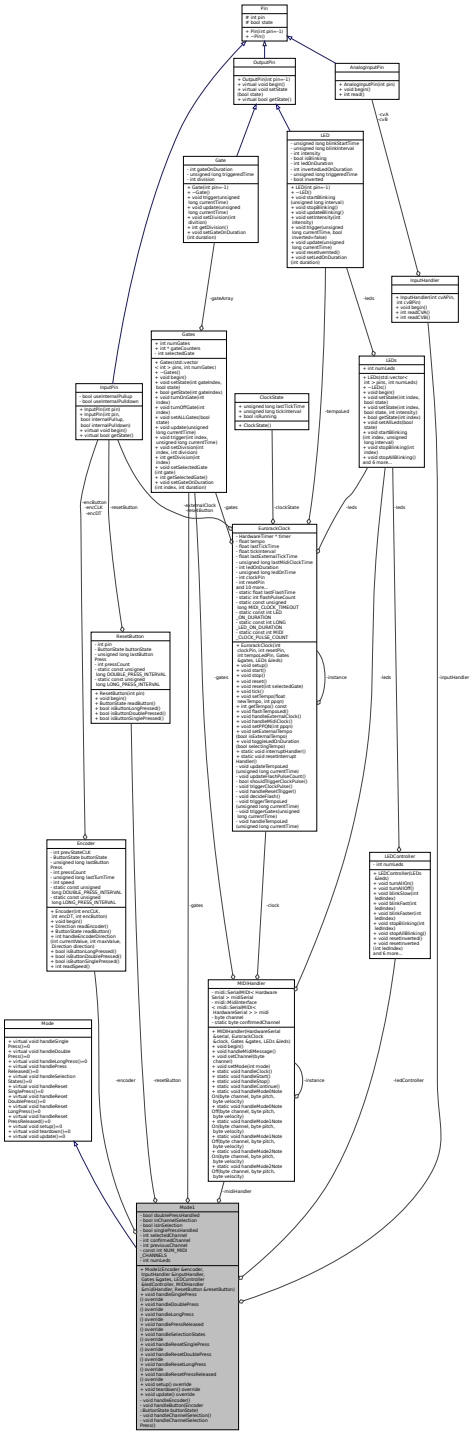
## 4.19 Mode1 Class Reference

`#include <Mode1.h>`

Inheritance diagram for Mode1:



Collaboration diagram for Mode1:



## Public Member Functions

- Mode1 (Encoder &encoder, InputHandler &inputHandler, Gates &gates, LEDController &ledController, MIDIHandler &midiHandler, ResetButton &resetButton)
- void handleSinglePress () override
- void handleDoublePress () override
- void handleLongPress () override

- void handlePressReleased () override
- void handleSelectionStates () override
- void handleResetSinglePress () override
- void handleResetDoublePress () override
- void handleResetLongPress () override
- void handleResetPressReleased () override
- void setup () override
- void teardown () override
- void update () override

### Private Member Functions

- void handleEncoder ()
- void handleButton (Encoder::ButtonState buttonState)
- void handleChannelSelection ()
- void handleChannelSelectionPress ()

### Private Attributes

- LEDController & ledController
- InputHandler & inputHandler
- Encoder & encoder
- ResetButton & resetButton
- Gates & gates
- MIDIHandler & midiHandler
- bool doublePressHandled = false
- bool inChannelSelection = false
- bool isInSelection = false
- bool singlePressHandled = false
- int selectedChannel = 9
- int confirmedChannel = 9
- int previousChannel = -1
- const int NUM_MIDI_CHANNELS = 16
- int numLeds = 8

## 4.19.1 Constructor & Destructor Documentation

#### 4.19.1.1 Mode1()

```
Mode1::Mode1 (
            Encoder & encoder,
            InputHandler & inputHandler,
            Gates & gates,
            LEDController & ledController,
            MIDIHandler & midiHandler,
            ResetButton & resetButton )
```

## 4.19.2 Member Function Documentation

### 4.19.2.1 handleButton()

```
void Mode1::handleButton (
            Encoder::ButtonState buttonState )  [private]
```

### 4.19.2.2 handleChannelSelection()

```
void Mode1::handleChannelSelection ( )  [private]
```

### 4.19.2.3 handleChannelSelectionPress()

```
void Mode1::handleChannelSelectionPress ( )  [private]
```

### 4.19.2.4 handleDoublePress()

```
void Mode1::handleDoublePress ( )  [override], [virtual]
```

Implements Mode.

### 4.19.2.5 handleEncoder()

```
void Mode1::handleEncoder ( )  [private]
```

### 4.19.2.6 handleLongPress()

```
void Mode1::handleLongPress ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.7 handlePressReleased()

```
void Mode1::handlePressReleased ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.8 handleResetDoublePress()

```
void Mode1::handleResetDoublePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.9 handleResetLongPress()

```
void Mode1::handleResetLongPress ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.10 handleResetPressReleased()

```
void Mode1::handleResetPressReleased ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.11 handleResetSinglePress()

```
void Mode1::handleResetSinglePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.12 handleSelectionStates()

```
void Mode1::handleSelectionStates ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.13 handleSinglePress()

```
void Mode1::handleSinglePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.14 setup()

```
void Mode1::setup ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.15 teardown()

```
void Mode1::teardown ( )  [override], [virtual]
```

Implements Mode.

#### 4.19.2.16 update()

```
void Mode1::update ( )  [override], [virtual]
```

Implements Mode.

### 4.19.3 Member Data Documentation

#### 4.19.3.1 confirmedChannel

```
int Mode1::confirmedChannel = 9  [private]
```

#### 4.19.3.2 doublePressHandled

```
bool Mode1::doublePressHandled = false  [private]
```

#### 4.19.3.3 encoder

```
Encoder& Mode1::encoder  [private]
```

#### 4.19.3.4 gates

```
Gates& Mode1::gates  [private]
```

#### 4.19.3.5 inChannelSelection

```
bool Mode1::inChannelSelection = false  [private]
```

#### 4.19.3.6 inputHandler

```
InputHandler& Mode1::inputHandler  [private]
```

#### 4.19.3.7 isInSelection

```
bool Mode1::isInSelection = false  [private]
```

#### 4.19.3.8 ledController

```
LEDController& Mode1::ledController  [private]
```

#### 4.19.3.9 midiHandler

```
MIDIHandler& Mode1::midiHandler  [private]
```

#### 4.19.3.10 NUM_MIDI_CHANNELS

```
const int Mode1::NUM_MIDI_CHANNELS = 16  [private]
```

#### 4.19.3.11 numLeds

```
int Mode1::numLeds = 8  [private]
```

#### 4.19.3.12 previousChannel

```
int Mode1::previousChannel = -1  [private]
```

#### 4.19.3.13 resetButton

```
ResetButton& Mode1::resetButton  [private]
```

#### 4.19.3.14 selectedChannel

```
int Mode1::selectedChannel = 9  [private]
```

#### 4.19.3.15 singlePressHandled

```
bool Mode1::singlePressHandled = false  [private]
```

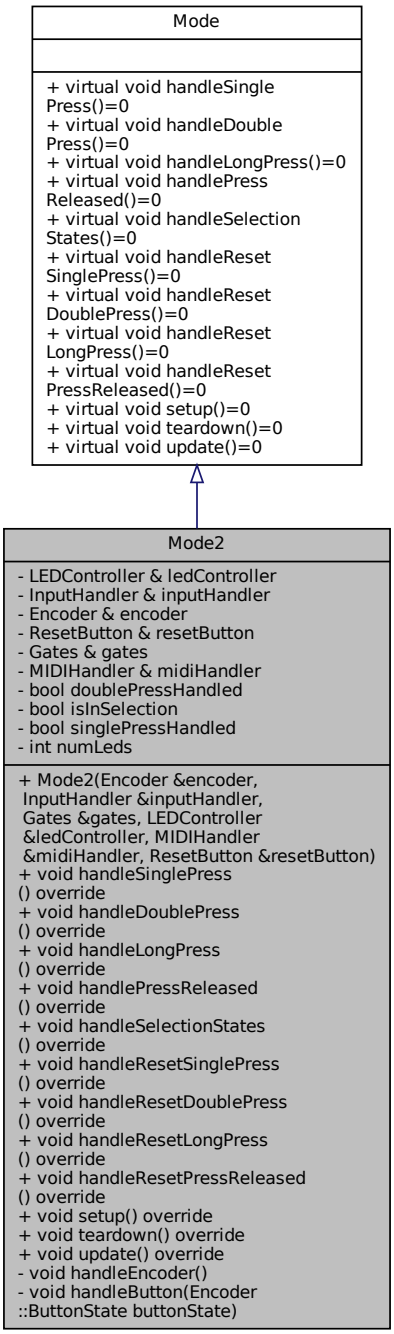
The documentation for this class was generated from the following files:

- include/Mode1.h
- src/Mode1.cpp

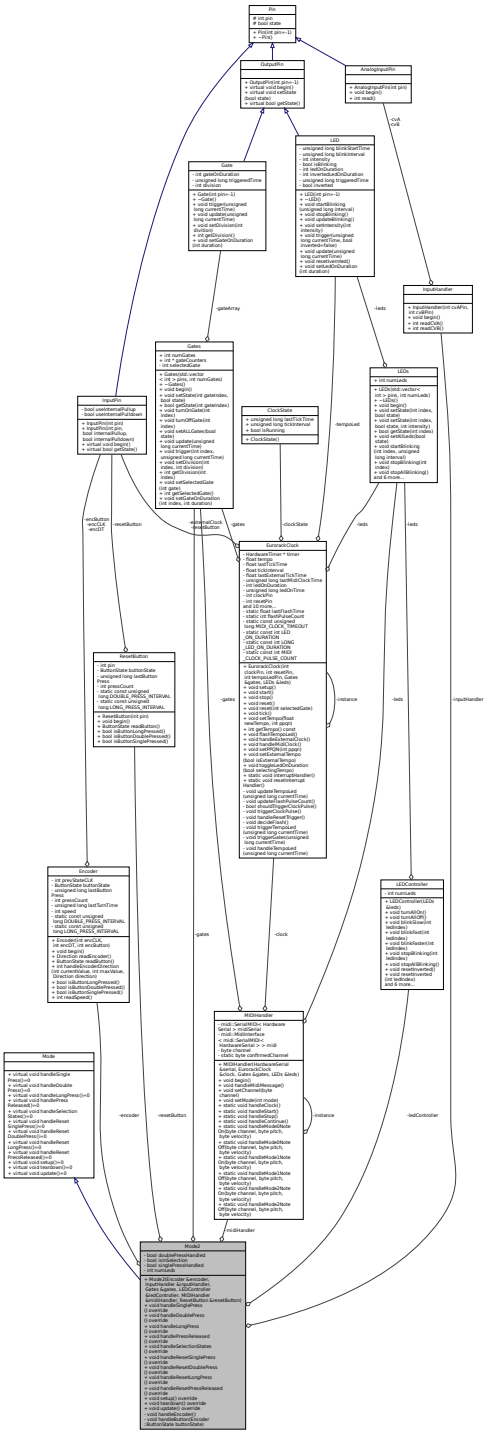
## 4.20 Mode2 Class Reference

`#include <Mode2.h>`

Inheritance diagram for Mode2:



Collaboration diagram for Mode2:



## Public Member Functions

- Mode2 (Encoder &encoder, InputHandler &inputHandler, Gates &gates, LEDController &ledController, MIDIHandler &midiHandler, ResetButton &resetButton)
- void handleSinglePress () override
- void handleDoublePress () override
- void handleLongPress () override

- void handlePressReleased () override
- void handleSelectionStates () override
- void handleResetSinglePress () override
- void handleResetDoublePress () override
- void handleResetLongPress () override
- void handleResetPressReleased () override
- void setup () override
- void teardown () override
- void update () override

### Private Member Functions

- void handleEncoder ()
- void handleButton (Encoder::ButtonState buttonState)

### Private Attributes

- LEDController & ledController
- InputHandler & inputHandler
- Encoder & encoder
- ResetButton & resetButton
- Gates & gates
- MIDIHandler & midiHandler
- bool doublePressHandled = false
- bool isInSelection = false
- bool singlePressHandled = false
- int numLeds = 8

## 4.20.1 Constructor & Destructor Documentation

#### 4.20.1.1 Mode2()

```
Mode2::Mode2 (
            Encoder & encoder,
            InputHandler & inputHandler,
            Gates & gates,
            LEDController & ledController,
            MIDIHandler & midiHandler,
            ResetButton & resetButton )
```

## 4.20.2 Member Function Documentation

#### 4.20.2.1 handleButton()

```
void Mode2::handleButton (
            Encoder::ButtonState buttonState )  [private]
```

#### 4.20.2.2 handleDoublePress()

```
void Mode2::handleDoublePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.3 handleEncoder()

```
void Mode2::handleEncoder ( )  [private]
```

#### 4.20.2.4 handleLongPress()

```
void Mode2::handleLongPress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.5 handlePressReleased()

```
void Mode2::handlePressReleased ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.6 handleResetDoublePress()

```
void Mode2::handleResetDoublePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.7 handleResetLongPress()

```
void Mode2::handleResetLongPress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.8 handleResetPressReleased()

```
void Mode2::handleResetPressReleased ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.9 handleResetSinglePress()

```
void Mode2::handleResetSinglePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.10 handleSelectionStates()

```
void Mode2::handleSelectionStates ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.11 handleSinglePress()

```
void Mode2::handleSinglePress ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.12 setup()

```
void Mode2::setup ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.13 teardown()

```
void Mode2::teardown ( )  [override], [virtual]
```

Implements Mode.

#### 4.20.2.14 update()

```
void Mode2::update ( )  [override], [virtual]
```

Implements Mode.

### 4.20.3 Member Data Documentation

#### 4.20.3.1 doublePressHandled

```
bool Mode2::doublePressHandled = false  [private]
```

#### 4.20.3.2 encoder

```
Encoder& Mode2::encoder  [private]
```

#### 4.20.3.3 gates

```
Gates& Mode2::gates  [private]
```

#### 4.20.3.4 inputHandler

```
InputHandler& Mode2::inputHandler  [private]
```

#### 4.20.3.5 isInSelection

```
bool Mode2::isInSelection = false  [private]
```

#### 4.20.3.6 ledController

```
LEDController& Mode2::ledController  [private]
```

#### 4.20.3.7 midiHandler

```
MIDIHandler& Mode2::midiHandler  [private]
```

#### 4.20.3.8 numLeds

```
int Mode2::numLeds = 8  [private]
```

#### 4.20.3.9 resetButton

```
ResetButton& Mode2::resetButton  [private]
```

#### 4.20.3.10 singlePressHandled

```
bool Mode2::singlePressHandled = false  [private]
```

The documentation for this class was generated from the following files:

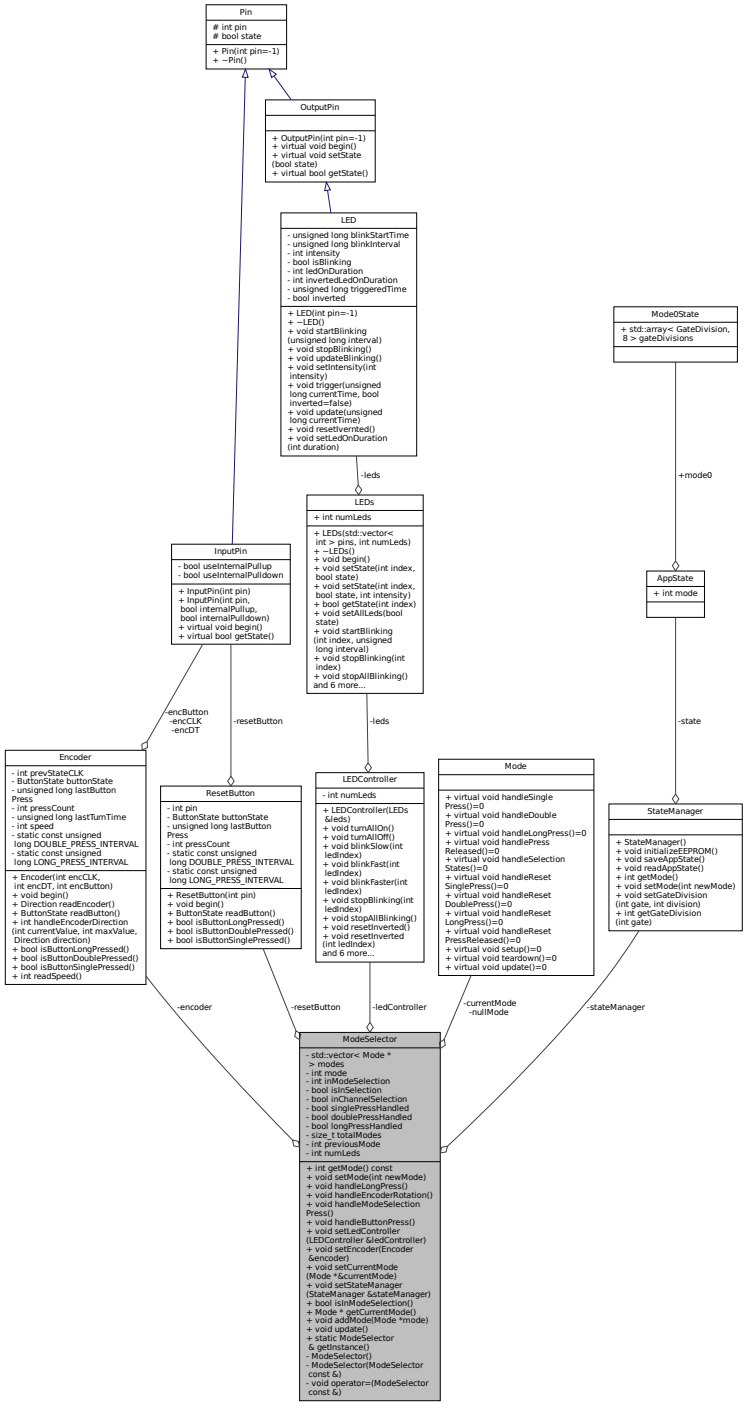
- include/Mode2.h
- src/Mode2.cpp

# 4.21 ModeSelector Class Reference

Mode Selector Singleton. This class is responsible for managing the different modes of the device. It provides methods to add modes, set the current mode, and handle mode selection.

`#include <ModeSelector.h>`

Collaboration diagram for ModeSelector:



### Public Member Functions

- int getMode () const
- void setMode (int newMode)
- void handleLongPress ()
- void handleEncoderRotation ()
- void handleModeSelectionPress ()
- void handleButtonPress ()
- void setLedController (LEDController &ledController)
- void setEncoder (Encoder &encoder)
- void setCurrentMode (Mode *&currentMode)
- void setStateManager (StateManager &stateManager)
- bool isInModeSelection ()
- Mode * getCurrentMode ()
- void addMode (Mode *mode)
- void update ()

### Static Public Member Functions

- static ModeSelector & getInstance ()

### Private Member Functions

- ModeSelector ()

  *Constructor is private.*
- ModeSelector (ModeSelector const &)
- void operator= (ModeSelector const &)

### Private Attributes

- std::vector< Mode * > modes
- Mode * nullMode = nullptr
- Mode *& currentMode
- int mode
- int inModeSelection = false
- LEDController * ledController
- Encoder * encoder
- StateManager * stateManager
- ResetButton * resetButton
- bool isInSelection
- bool inChannelSelection
- bool singlePressHandled
- bool doublePressHandled
- bool longPressHandled
- size_t totalModes = modes.size()
- int previousMode = -1
- int numLeds

### 4.21.1 Detailed Description

Mode Selector Singleton. This class is responsible for managing the different modes of the device. It provides methods to add modes, set the current mode, and handle mode selection.

## 4.21.2 Constructor & Destructor Documentation

### 4.21.2.1 ModeSelector() [1/2]

```
ModeSelector::ModeSelector ( )  [private]
```

Constructor is private.

### 4.21.2.2 ModeSelector() [2/2]

```
ModeSelector::ModeSelector (
            ModeSelector const &  )  [private]
```

## 4.21.3 Member Function Documentation

### 4.21.3.1 addMode()

```
void ModeSelector::addMode (
            Mode * mode )
```

### 4.21.3.2 getCurrentMode()

```
Mode * ModeSelector::getCurrentMode ( )
```

### 4.21.3.3 getInstance()

```
ModeSelector & ModeSelector::getInstance ( )  [static]
```

### 4.21.3.4 getMode()

```
int ModeSelector::getMode ( ) const
```

#### 4.21.3.5 handleButtonPress()

```
void ModeSelector::handleButtonPress ( )
```

#### 4.21.3.6 handleEncoderRotation()

```
void ModeSelector::handleEncoderRotation ( )
```

#### 4.21.3.7 handleLongPress()

```
void ModeSelector::handleLongPress ( )
```

#### 4.21.3.8 handleModeSelectionPress()

```
void ModeSelector::handleModeSelectionPress ( )
```

#### 4.21.3.9 isInModeSelection()

```
bool ModeSelector::isInModeSelection ( )
```

#### 4.21.3.10 operator=()

```
void ModeSelector::operator= (
            ModeSelector const &  )  [private]
```

#### 4.21.3.11 setCurrentMode()

```
void ModeSelector::setCurrentMode (
            Mode *& currentMode )
```

#### 4.21.3.12 setEncoder()

```
void ModeSelector::setEncoder (
            Encoder & encoder )
```

#### 4.21.3.13 setLedController()

```
void ModeSelector::setLedController (
            LEDController & ledController )
```

#### 4.21.3.14 setMode()

```
void ModeSelector::setMode (
            int newMode )
```

#### 4.21.3.15 setStateManager()

```
void ModeSelector::setStateManager (
            StateManager & stateManager )
```

#### 4.21.3.16 update()

```
void ModeSelector::update ( )
```

### 4.21.4 Member Data Documentation

#### 4.21.4.1 currentMode

```
Mode*& ModeSelector::currentMode  [private]
```

#### 4.21.4.2 doublePressHandled

```
bool ModeSelector::doublePressHandled  [private]
```

#### 4.21.4.3 encoder

```
Encoder* ModeSelector::encoder  [private]
```

#### 4.21.4.4 inChannelSelection

```
bool ModeSelector::inChannelSelection  [private]
```

#### 4.21.4.5 inModeSelection

```
int ModeSelector::inModeSelection = false  [private]
```

#### 4.21.4.6 isInSelection

```
bool ModeSelector::isInSelection  [private]
```

#### 4.21.4.7 ledController

```
LEDController* ModeSelector::ledController  [private]
```

#### 4.21.4.8 longPressHandled

```
bool ModeSelector::longPressHandled  [private]
```

#### 4.21.4.9 mode

```
int ModeSelector::mode  [private]
```

#### 4.21.4.10 modes

```
std::vector<Mode*> ModeSelector::modes  [private]
```

#### 4.21.4.11 nullMode

```
Mode* ModeSelector::nullMode = nullptr  [private]
```

#### 4.21.4.12 numLeds

```
int ModeSelector::numLeds  [private]
```

#### 4.21.4.13 previousMode

```
int ModeSelector::previousMode = -1  [private]
```

#### 4.21.4.14 resetButton

```
ResetButton* ModeSelector::resetButton  [private]
```

#### 4.21.4.15 singlePressHandled

```
bool ModeSelector::singlePressHandled  [private]
```

#### 4.21.4.16 stateManager

```
StateManager* ModeSelector::stateManager  [private]
```

#### 4.21.4.17 totalModes

```
size_t ModeSelector::totalModes = modes.size()  [private]
```

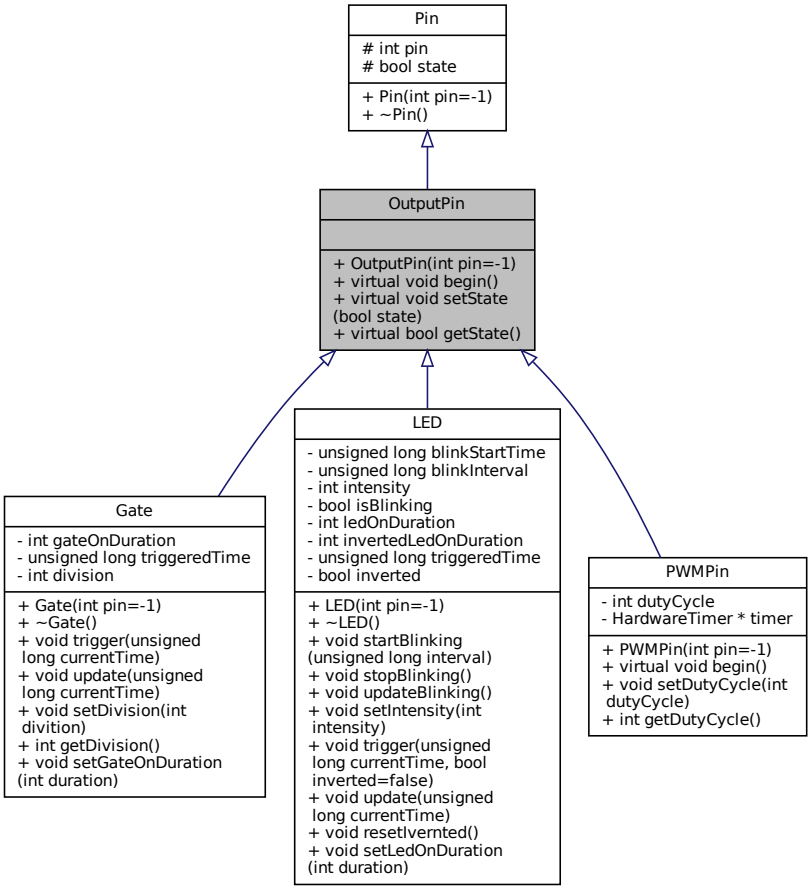
The documentation for this class was generated from the following files:

- include/ModeSelector.h
- src/ModeSelector.cpp

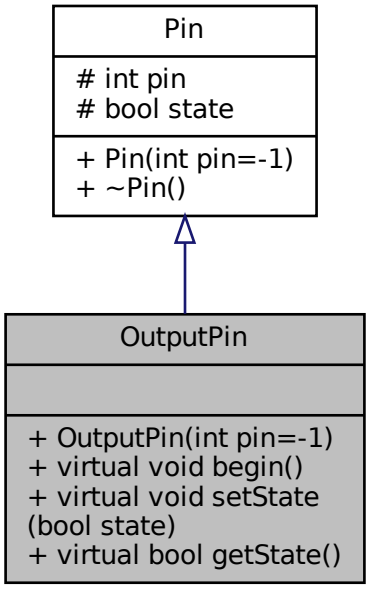
## 4.22 OutputPin Class Reference

`#include <Pin.h>`

Inheritance diagram for OutputPin:



Collaboration diagram for OutputPin:




## Public Member Functions

- OutputPin (int pin=-1)
- virtual void begin ()
- virtual void setState (bool state)
- virtual bool getState ()

## Additional Inherited Members

### 4.22.1 Constructor & Destructor Documentation

#### 4.22.1.1 OutputPin()

```
OutputPin::OutputPin (
            int pin = -1 )
```

### 4.22.2 Member Function Documentation

#### 4.22.2.1 begin()

```
void OutputPin::begin ( )  [virtual]
```

Reimplemented in PWMPin.

#### 4.22.2.2 getState()

```
bool OutputPin::getState ( )  [virtual]
```

#### 4.22.2.3 setState()

```
void OutputPin::setState (
            bool state )  [virtual]
```

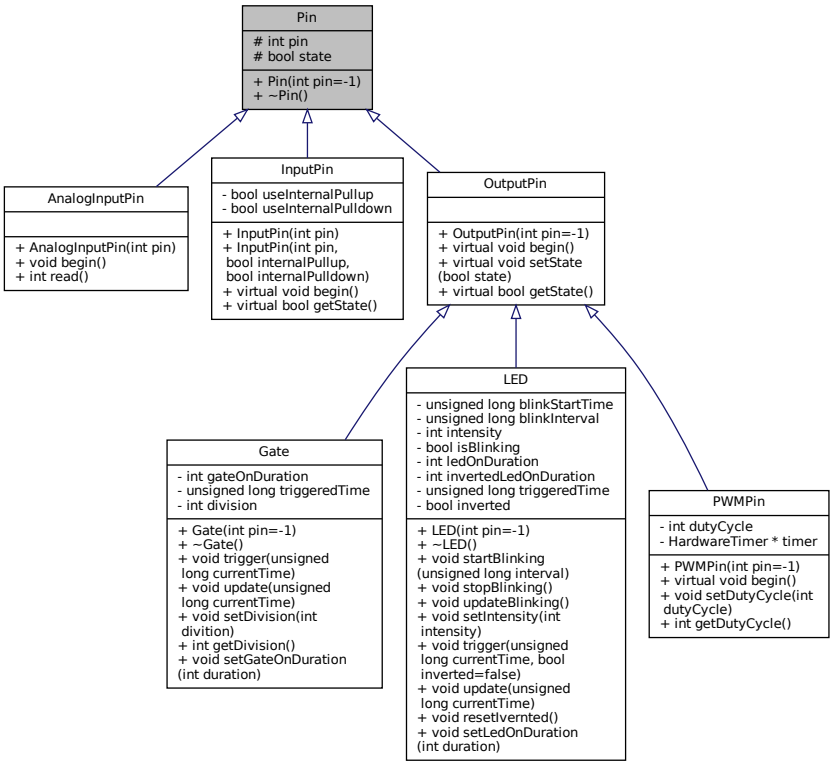
The documentation for this class was generated from the following files:

- include/Pin.h
- src/Pin.cpp

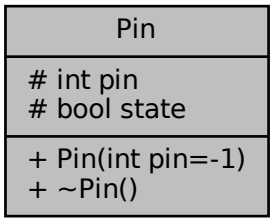
## 4.23 Pin Class Reference

```
#include <Pin.h>
```

Inheritance diagram for Pin:



Collaboration diagram for Pin:




## Public Member Functions

- Pin (int pin=-1)
- ~Pin ()

## Protected Attributes

- int pin
- bool state

### 4.23.1 Constructor & Destructor Documentation

#### 4.23.1.1 Pin()

```
Pin::Pin (
            int pin = -1 )
```

#### 4.23.1.2 ~Pin()

```
Pin::~Pin ( )
```

### 4.23.2 Member Data Documentation

#### 4.23.2.1 pin

```
int Pin::pin  [protected]
```

#### 4.23.2.2 state

```
bool Pin::state  [protected]
```

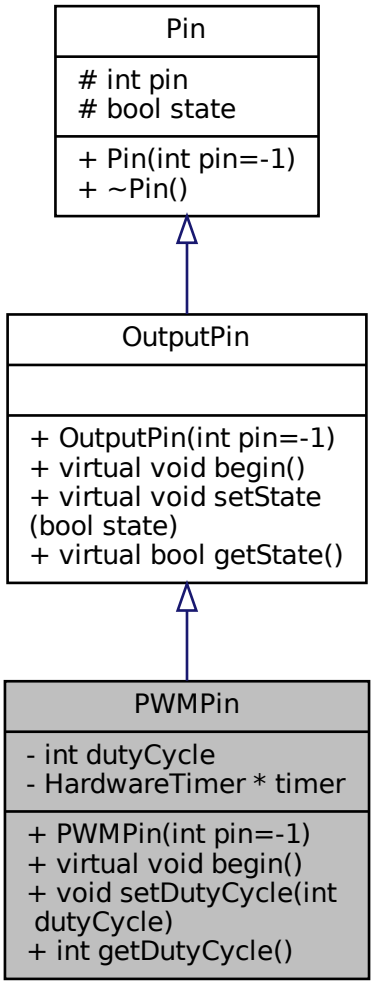
The documentation for this class was generated from the following files:

- include/Pin.h
- src/Pin.cpp

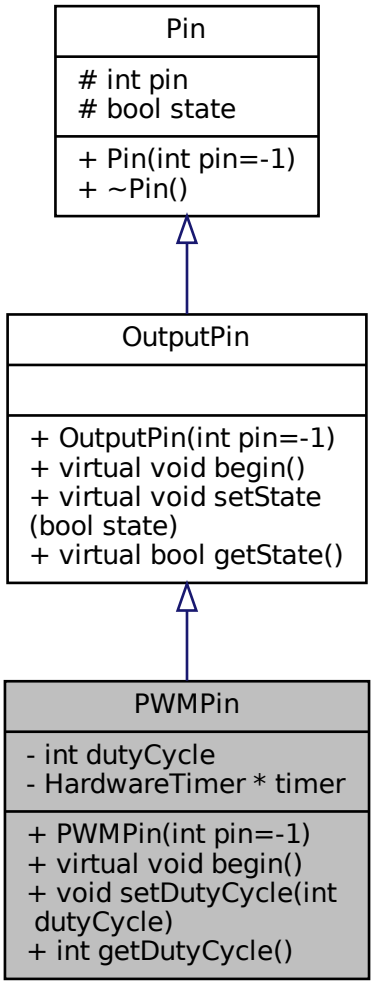
## 4.24 PWMPin Class Reference

```
#include <Pin.h>
```

Inheritance diagram for PWMPin:



Collaboration diagram for PWMPin:




## Public Member Functions

- PWMPin (int pin=-1)
- virtual void begin ()
- void setDutyCycle (int dutyCycle)
- int getDutyCycle ()

## Private Attributes

- int dutyCycle
- HardwareTimer * timer

## Additional Inherited Members

### 4.24.1 Constructor & Destructor Documentation

#### 4.24.1.1 PWMPin()

```
PWMPin::PWMPin (
            int pin = -1 )
```

### 4.24.2 Member Function Documentation

#### 4.24.2.1 begin()

```
void PWMPin::begin ( )  [virtual]
```

Reimplemented from OutputPin.

#### 4.24.2.2 getDutyCycle()

```
int PWMPin::getDutyCycle ( )
```

#### 4.24.2.3 setDutyCycle()

```
void PWMPin::setDutyCycle (
            int dutyCycle )
```

### 4.24.3 Member Data Documentation

#### 4.24.3.1 dutyCycle

```
int PWMPin::dutyCycle  [private]
```

#### 4.24.3.2 timer

```
HardwareTimer* PWMPin::timer  [private]
```

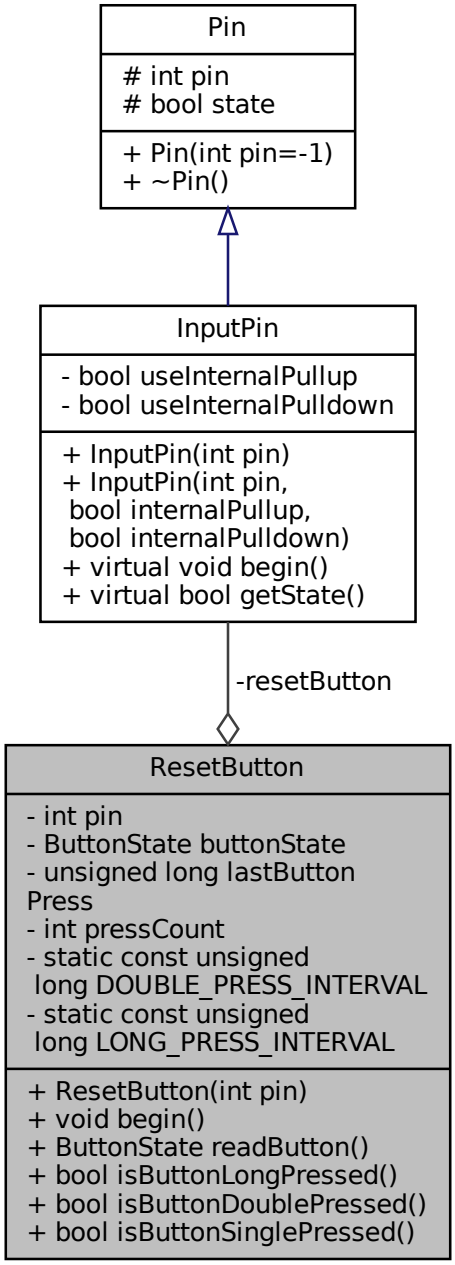
The documentation for this class was generated from the following files:

- include/Pin.h
- src/Pin.cpp

## 4.25 ResetButton Class Reference

```
#include <ResetButton.h>
```

Collaboration diagram for ResetButton:




## Public Types

- enum ButtonState { OPEN , PRESSED }

## Public Member Functions

- ResetButton (int pin)

- void begin ()
- ButtonState readButton ()
- bool isButtonLongPressed ()
- bool isButtonDoublePressed ()
- bool isButtonSinglePressed ()

## Private Attributes

- int pin
- InputPin resetButton
- ButtonState buttonState
- unsigned long lastButtonPress
- int pressCount

## Static Private Attributes

- static const unsigned long DOUBLE_PRESS_INTERVAL = 500
- static const unsigned long LONG_PRESS_INTERVAL = 1000

### 4.25.1 Member Enumeration Documentation

#### 4.25.1.1 ButtonState

```
enum ResetButton::ButtonState
```

**Enumerator**

| | |
|---|---|
| OPEN | |
| PRESSED | |

### 4.25.2 Constructor & Destructor Documentation

#### 4.25.2.1 ResetButton()

```
ResetButton::ResetButton (
            int pin )
```

### 4.25.3 Member Function Documentation

#### 4.25.3.1 begin()

```
void ResetButton::begin ( )
```

#### 4.25.3.2 isButtonDoublePressed()

```
bool ResetButton::isButtonDoublePressed ( )
```

#### 4.25.3.3 isButtonLongPressed()

```
bool ResetButton::isButtonLongPressed ( )
```

#### 4.25.3.4 isButtonSinglePressed()

```
bool ResetButton::isButtonSinglePressed ( )
```

#### 4.25.3.5 readButton()

```
ResetButton::ButtonState ResetButton::readButton ( )
```

### 4.25.4 Member Data Documentation

#### 4.25.4.1 buttonState

```
ButtonState ResetButton::buttonState  [private]
```

#### 4.25.4.2 DOUBLE_PRESS_INTERVAL

```
const unsigned long ResetButton::DOUBLE_PRESS_INTERVAL = 500  [static], [private]
```

#### 4.25.4.3 lastButtonPress

```
unsigned long ResetButton::lastButtonPress  [private]
```

#### 4.25.4.4 LONG_PRESS_INTERVAL

```
const unsigned long ResetButton::LONG_PRESS_INTERVAL = 1000  [static], [private]
```

#### 4.25.4.5 pin

```
int ResetButton::pin  [private]
```

#### 4.25.4.6 pressCount

```
int ResetButton::pressCount  [private]
```

#### 4.25.4.7 resetButton

```
InputPin ResetButton::resetButton  [private]
```

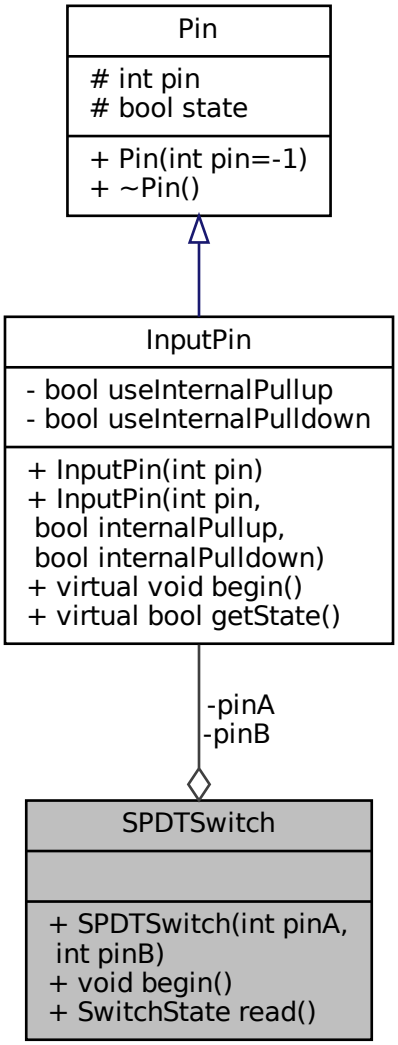
The documentation for this class was generated from the following files:

- include/ResetButton.h
- src/ResetButton.cpp

## 4.26 SPDTSwitch Class Reference

`#include <SPDTSwitch.h>`

Collaboration diagram for SPDTSwitch:




### Public Member Functions

- SPDTSwitch (int pinA, int pinB)
- void begin ()
- SwitchState read ()

### Private Attributes

- InputPin pinA
- InputPin pinB

### 4.26.1 Constructor & Destructor Documentation

#### 4.26.1.1 SPDTSwitch()

```
SPDTSwitch::SPDTSwitch (
            int pinA,
            int pinB )
```

### 4.26.2 Member Function Documentation

#### 4.26.2.1 begin()

```
void SPDTSwitch::begin ( )
```

#### 4.26.2.2 read()

```
SwitchState SPDTSwitch::read ( )
```

### 4.26.3 Member Data Documentation

#### 4.26.3.1 pinA

```
InputPin SPDTSwitch::pinA  [private]
```

#### 4.26.3.2 pinB

```
InputPin SPDTSwitch::pinB  [private]
```

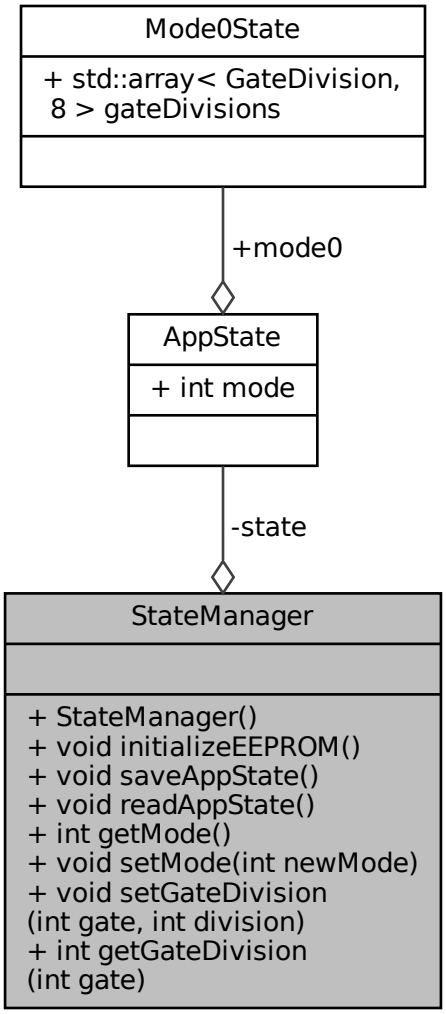
The documentation for this class was generated from the following files:

- include/SPDTSwitch.h
- src/SPDTSwitch.cpp

## 4.27 StateManager Class Reference

`#include <StateManager.h>`

Collaboration diagram for StateManager:




### Public Member Functions

- StateManager ()
- void initializeEEPROM ()

  *Initializes the EEPROM memory with the default AppState values if the EEPROM has not been initialized.*
- void saveAppState ()

  *Saves the current AppState object 'state' to the EEPROM memory.*
- void readAppState ()

  *Reads the AppState object 'state' from the EEPROM memory.*

- int getMode ()

  *Returns the current mode stored in the AppState object 'state'.*
- void setMode (int newMode)

  *Sets the current mode in the AppState object 'state'.*
- void setGateDivision (int gate, int division)

  *Sets the gate division for a specific gate in the AppState object 'state'.*
- int getGateDivision (int gate)

  *Returns the gate division for a specific gate from the AppState object 'state'.*

## Private Attributes

- AppState state

### 4.27.1 Constructor & Destructor Documentation

#### 4.27.1.1 StateManager()

```
StateManager::StateManager ( )
```

**Parameters**

| | |
|---|---|
| *state* | - The AppState object to be saved to the EEPROM |

Empty constructor - we will initialize the AppState object in the setup() function this way we can print debug messages.

### 4.27.2 Member Function Documentation

#### 4.27.2.1 getGateDivision()

```
int StateManager::getGateDivision (
            int gate )
```

Returns the gate division for a specific gate from the AppState object 'state'.

**Parameters**

| | |
|---|---|
| *gate* | - The gate to get the division for |

**Returns**

int - The division for the gate

#### 4.27.2.2 getMode()

```
int StateManager::getMode ( )
```

Returns the current mode stored in the AppState object 'state'.

**Returns**

int - The current mode

#### 4.27.2.3 initializeEEPROM()

```
void StateManager::initializeEEPROM ( )
```

Initializes the EEPROM memory with the default AppState values if the EEPROM has not been initialized.

Read the current state from the EEPROM

Set the mode to 0 by default.

Save the default state to the EEPROM

#### 4.27.2.4 readAppState()

```
void StateManager::readAppState ( )
```

Reads the AppState object 'state' from the EEPROM memory.

By using get we don't have to read each byte individually

#### 4.27.2.5 saveAppState()

```
void StateManager::saveAppState ( )
```

Saves the current AppState object 'state' to the EEPROM memory.

By using put we don't have to write each byte individually

#### 4.27.2.6 setGateDivision()

```
void StateManager::setGateDivision (
            int gate,
            int division )
```

Sets the gate division for a specific gate in the AppState object 'state'.

**Parameters**

| | |
|---|---|
| *gate* | - The gate to set the division for |
| *division* | - The division to set |

#### 4.27.2.7 setMode()

```
void StateManager::setMode (
            int newMode )
```

Sets the current mode in the AppState object 'state'.

**Parameters**

| | |
|---|---|
| *newMode* | - The new mode to set |

Save the new mode to the EEPROM

### 4.27.3 Member Data Documentation

#### 4.27.3.1 state

```
AppState StateManager::state  [private]
```

The documentation for this class was generated from the following files:

- include/StateManager.h
- src/StateManager.cpp

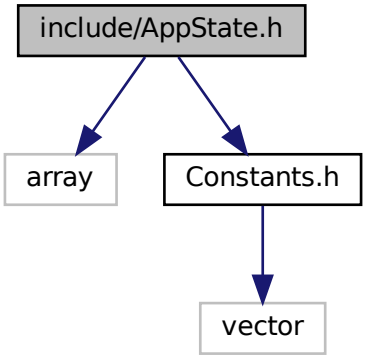
# Chapter 5

# File Documentation

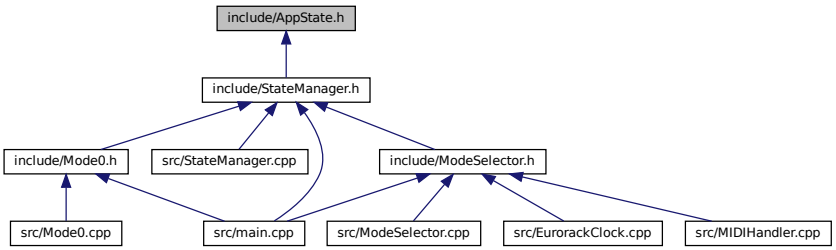
## 5.1 include/AppState.h File Reference

```
#include <array>
#include "Constants.h"
```

Include dependency graph for AppState.h:




This graph shows which files directly or indirectly include this file:



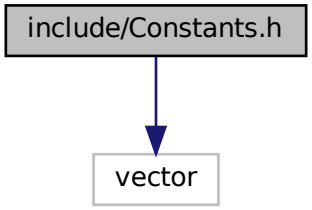
### Classes

- struct GateDivision

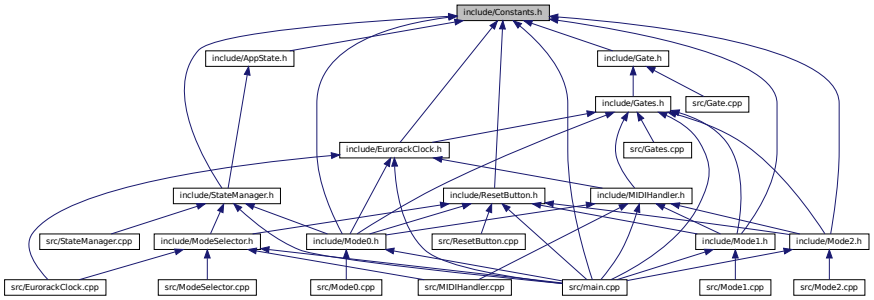
  *This is a global struct that holds the state of the application. It mainly holds items that need to persist after a power cycle. The object is initialized managed by the StateManager class.*
- struct Mode0State
- struct AppState

## 5.2 include/Constants.h File Reference

`#include <vector>`
Include dependency graph for Constants.h:




This graph shows which files directly or indirectly include this file:




### Variables

- std::vector< int > musicalIntervals

  *Pulses per quarter note.*
- const int musicalIntervalsSize
- unsigned char internalPPQN

  *Last flash time.*

### 5.2.1 Variable Documentation

#### 5.2.1.1 internalPPQN

```
unsigned char internalPPQN  [extern]
```

Last flash time.

#### 5.2.1.2 musicalIntervals

```
std::vector<int> musicalIntervals  [extern]
```

Pulses per quarter note.

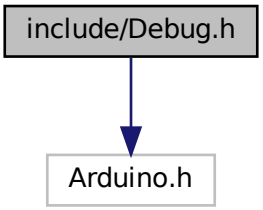
#### 5.2.1.3 musicalIntervalsSize

```
const int musicalIntervalsSize  [extern]
```

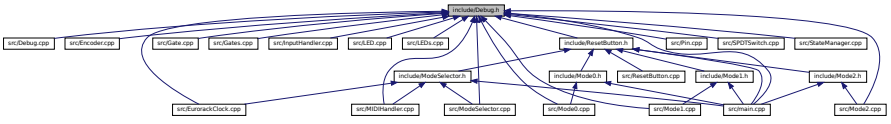
## 5.3 include/Debug.h File Reference

```
#include <Arduino.h>
```

Include dependency graph for Debug.h:




This graph shows which files directly or indirectly include this file:



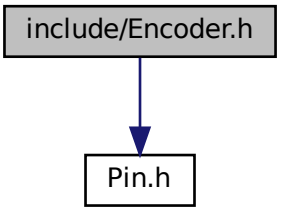
### Classes

- class Debug

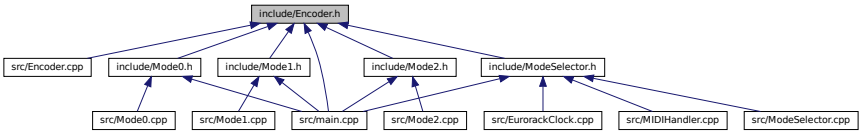
## 5.4 include/Encoder.h File Reference

```
#include "Pin.h"
```

Include dependency graph for Encoder.h:




This graph shows which files directly or indirectly include this file:




### Classes

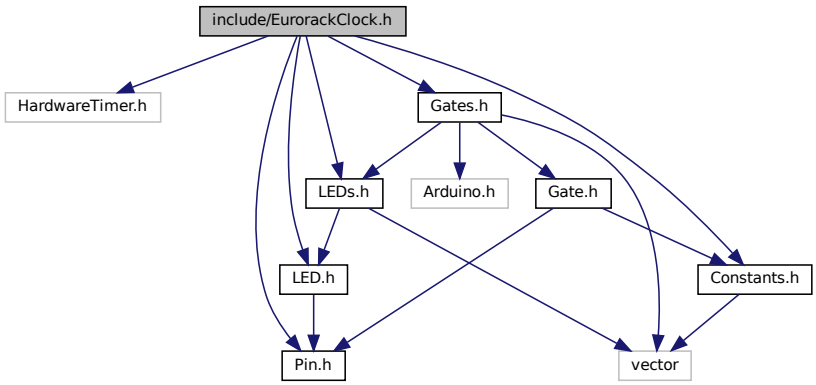
- class Encoder

## 5.5 include/EurorackClock.h File Reference

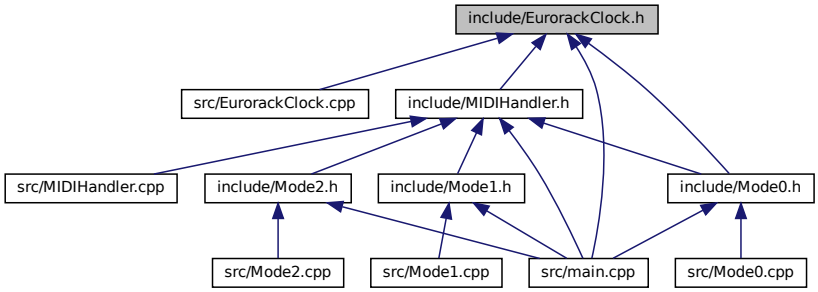
```
#include <HardwareTimer.h>
#include "LED.h"
#include "Pin.h"
#include "Gates.h"
#include "LEDs.h"
```

```
#include "Constants.h"
```

Include dependency graph for EurorackClock.h:




This graph shows which files directly or indirectly include this file:



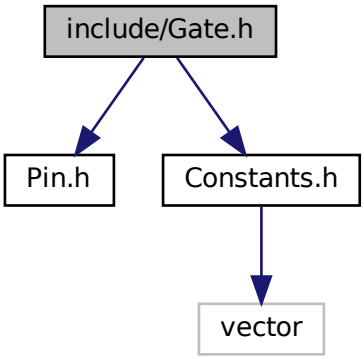

## Classes

- struct ClockState
- class EurorackClock

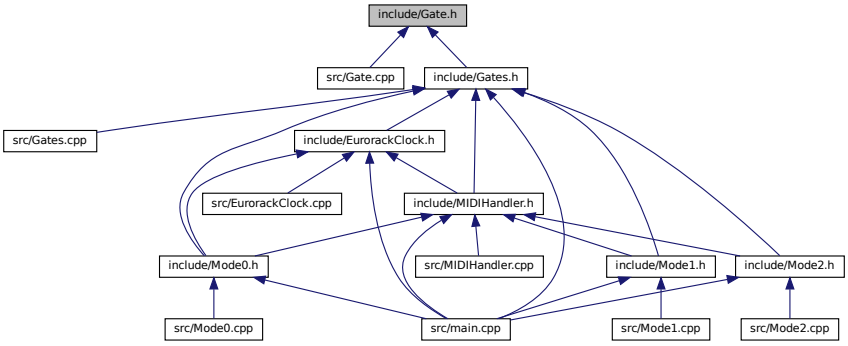
# 5.6 include/Gate.h File Reference

```
#include "Pin.h"
#include "Constants.h"
```

Include dependency graph for Gate.h:




This graph shows which files directly or indirectly include this file:




### Classes

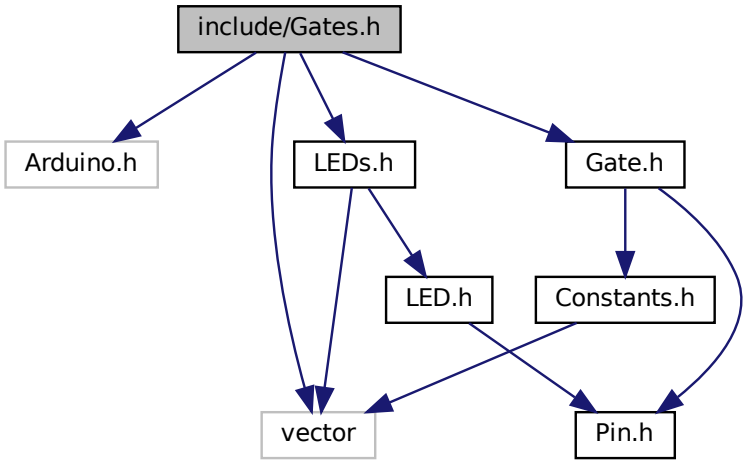
- class Gate

## 5.7 include/Gates.h File Reference

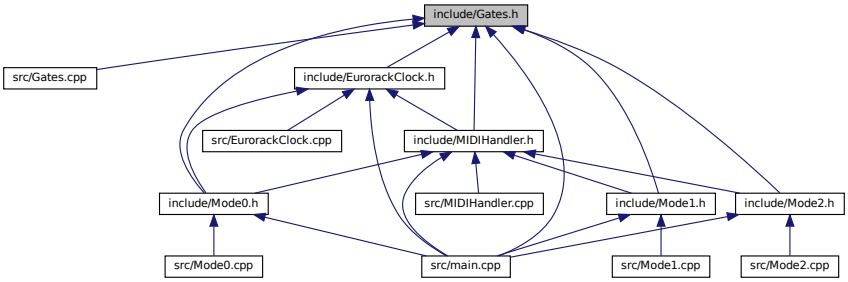
```
#include <Arduino.h>
#include "Gate.h"
#include "LEDs.h"
```

```
#include <vector>
```

Include dependency graph for Gates.h:




This graph shows which files directly or indirectly include this file:



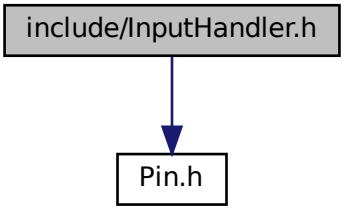

### Classes

- class Gates

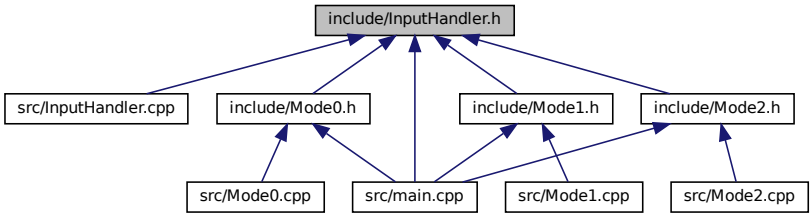
## 5.8 include/InputHandler.h File Reference

```
#include "Pin.h"
```

Include dependency graph for InputHandler.h:




This graph shows which files directly or indirectly include this file:



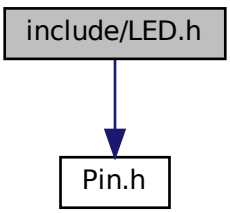

### Classes

- class InputHandler

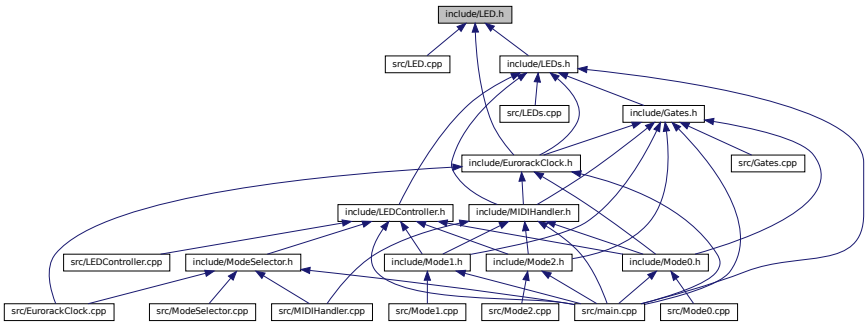
## 5.9 include/LED.h File Reference

```
#include "Pin.h"
```

Include dependency graph for LED.h:



This graph shows which files directly or indirectly include this file:



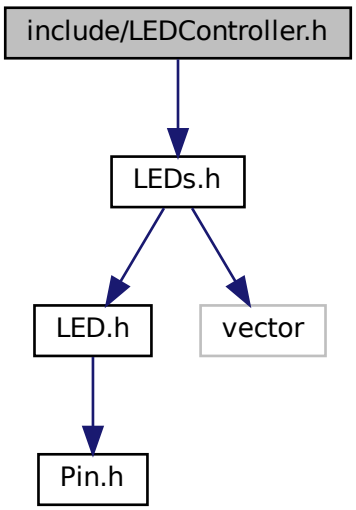

## Classes

- class LED

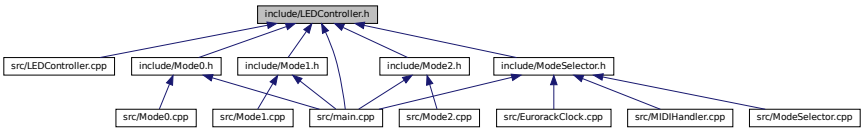
# 5.10 include/LEDController.h File Reference

```
#include "LEDs.h"
```

Include dependency graph for LEDController.h:



This graph shows which files directly or indirectly include this file:



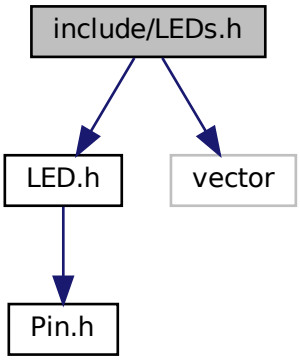

### Classes

- class LEDController

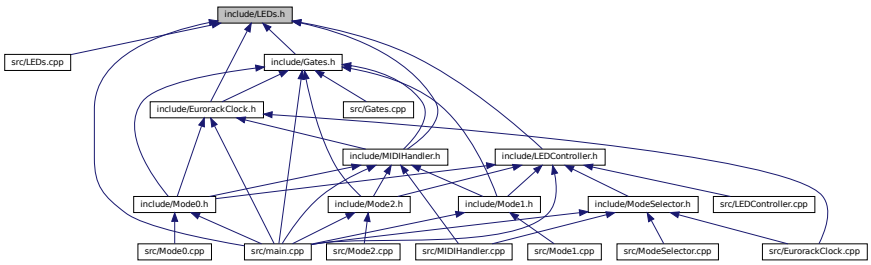
## 5.11 include/LEDs.h File Reference

```
#include "LED.h"
#include <vector>
```

Include dependency graph for LEDs.h:




This graph shows which files directly or indirectly include this file:



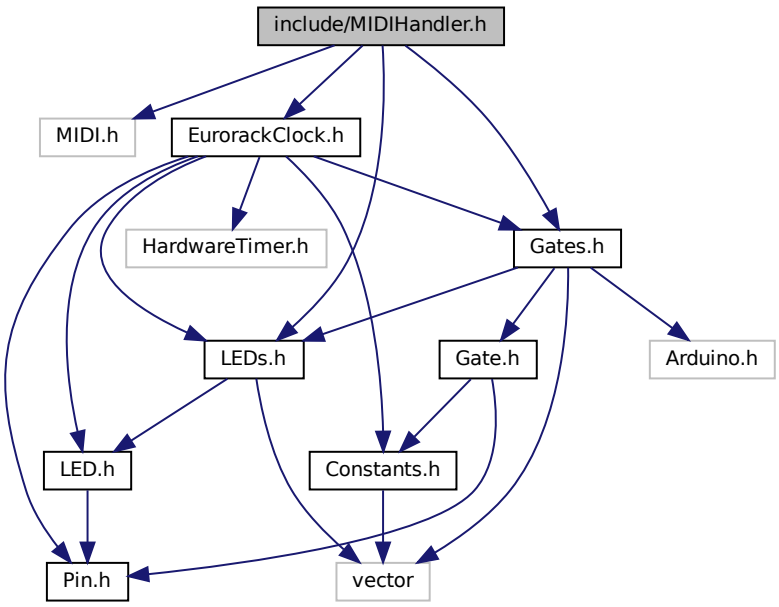
### Classes

- class LEDs

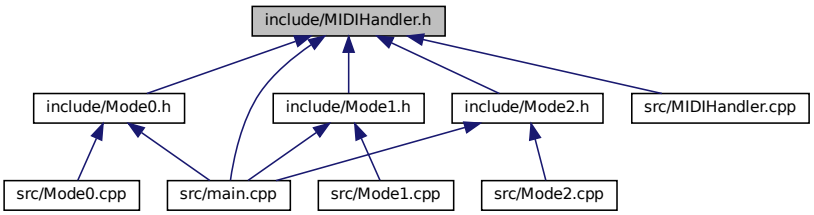
## 5.12 include/MIDIHandler.h File Reference

```
#include <MIDI.h>
#include "EurorackClock.h"
#include "Gates.h"
#include "LEDs.h"
```

Include dependency graph for MIDIHandler.h:




This graph shows which files directly or indirectly include this file:

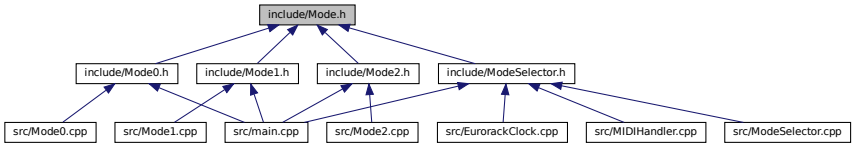



### Classes

- class MIDIHandler

## 5.13 include/Mode.h File Reference

This graph shows which files directly or indirectly include this file:




### Classes

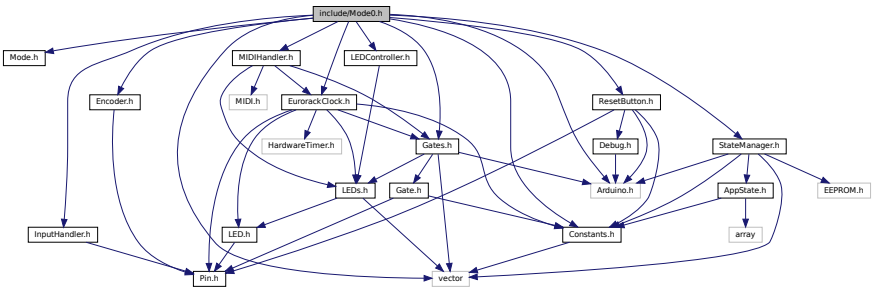
- class Mode

## 5.14 include/Mode0.h File Reference

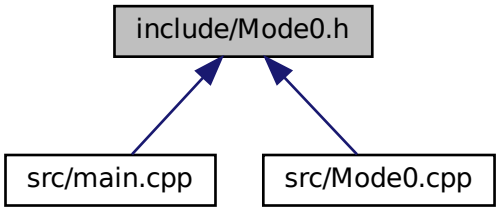
This mode is the main mode for the Eurorack Clock module.

```
#include "Mode.h"
#include "Encoder.h"
#include "Gates.h"
#include "LEDController.h"
#include "EurorackClock.h"
#include "MIDIHandler.h"
#include "Constants.h"
#include "ResetButton.h"
#include "InputHandler.h"
#include <vector>
#include <Arduino.h>
#include "StateManager.h"
```

Include dependency graph for Mode0.h:



This graph shows which files directly or indirectly include this file:




## Classes

- class Mode0

### 5.14.1 Detailed Description

This mode is the main mode for the Eurorack Clock module.

In this mode, the user can set the tempo, select the division of the clock signal, and select the gate output. It works with the Encoder, Gates, LEDController, MIDIHandler, ResetButton, and EurorackClock classes.

This mode utilizes an internal clock and can be synchronized with an external clock signal as well as MIDI clock. When the mode is active, the user can set the tempo by turning the encoder knob. The tempo can be set between 20 and 340 BPM. This is done by turning the encoder knob to the left to decrease the tempo or to the right to increase the tempo when in tempo selection mode.

Tempo selection mode is activated by pressing the encoder knob twice in quick succession. Then to exit this mode the user can press the encoder knob twice again.

The user can also select the division of the clock signal for each gate output. The division can be set to 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, or 256 PPQN. This is done by first selecting the gate output by rotating the encoder. The selected gate output will be indicated by the LED corresponding to the gate output. Then the user can press the encoder knob to enter the division selection mode. The division can be set by rotating the encoder knob. Pese the encoder knob again to exit the division selection mode.

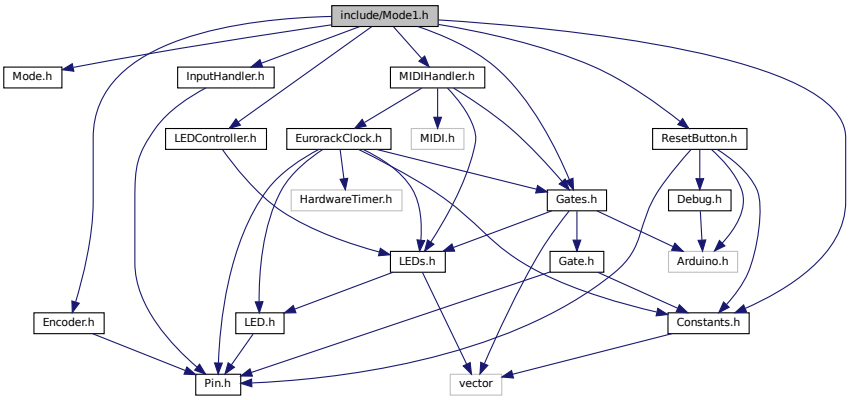
The internal clock is done by using the EurorackClock class. The clock signal is sent to the gate outputs using the Gates class. It's all complicated stuff but I'm working on making it easier to understand. The MIDIHandler class is used to handle MIDI clock signals. The LEDController class is used to control the LEDs on the module.

TODO: The internal clock works with a PPQN of 24 by default. This can be changed by pressing the reset button and rotating the encoder knob to select the desired PPQN.

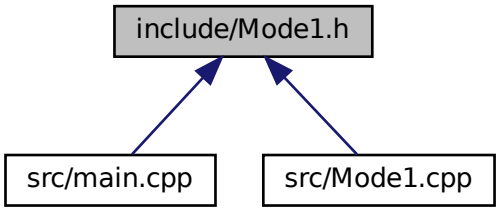
## 5.15 include/Mode1.h File Reference

```
#include "Mode.h"
#include "Encoder.h"
#include "Gates.h"
#include "LEDController.h"
#include "MIDIHandler.h"
#include "Constants.h"
#include "ResetButton.h"
#include "InputHandler.h"
```

Include dependency graph for Mode1.h:




This graph shows which files directly or indirectly include this file:




### Classes

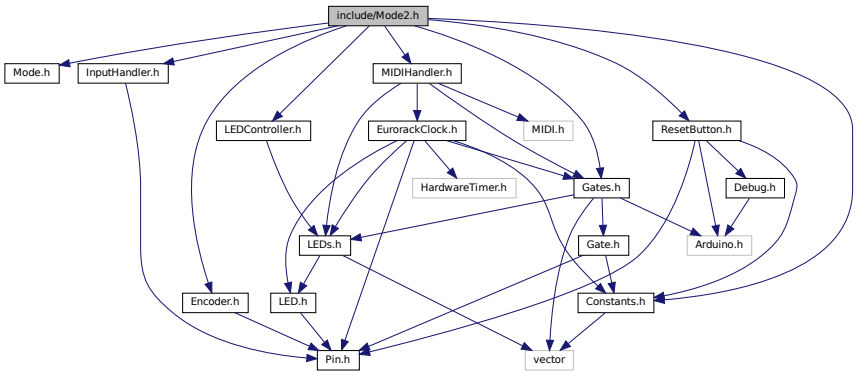
- class Mode1

## 5.16 include/Mode2.h File Reference

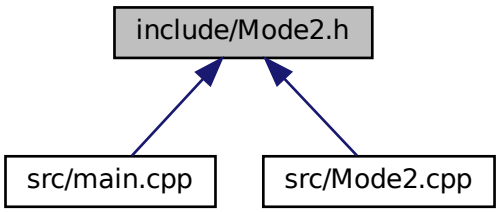
```
#include "Mode.h"
#include "LEDController.h"
```

```
#include "Encoder.h"
#include "Gates.h"
#include "MIDIHandler.h"
#include "Constants.h"
#include "InputHandler.h"
#include "ResetButton.h"
```

Include dependency graph for Mode2.h:




This graph shows which files directly or indirectly include this file:




## Classes

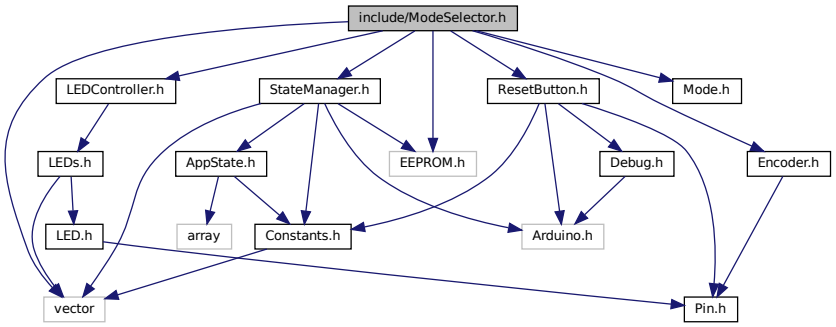
- class Mode2

# 5.17 include/ModeSelector.h File Reference

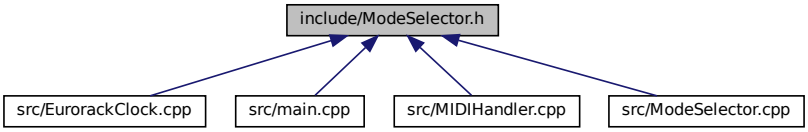
```
#include <vector>
#include <EEPROM.h>
#include "LEDController.h"
#include "Encoder.h"
#include "Mode.h"
#include "ResetButton.h"
```

```
#include "StateManager.h"
```

Include dependency graph for ModeSelector.h:




This graph shows which files directly or indirectly include this file:



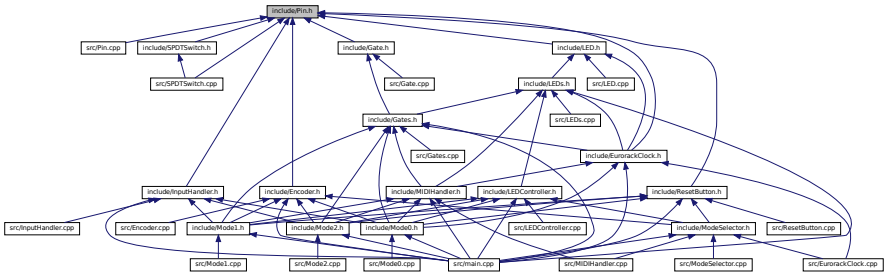

## Classes

- class ModeSelector

  *Mode* Selector Singleton. This class is responsible for managing the different modes of the device. It provides methods to add modes, set the current mode, and handle mode selection.

# 5.18 include/Pin.h File Reference

This graph shows which files directly or indirectly include this file:



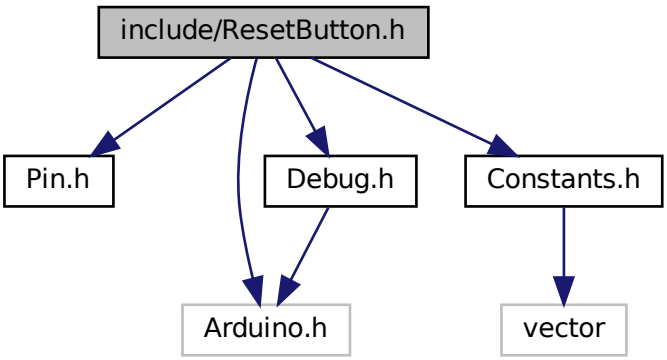
### Classes

- class Pin
- class InputPin
- class AnalogInputPin
- class OutputPin
- class PWMPin

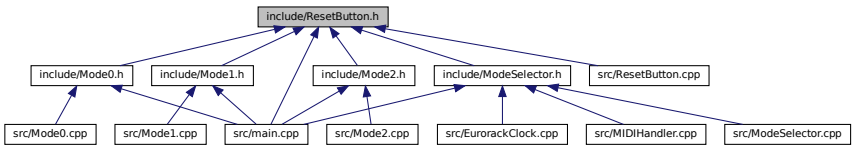
## 5.19 include/ResetButton.h File Reference

```
#include "Pin.h"
#include <Arduino.h>
#include "Debug.h"
#include "Constants.h"
```

Include dependency graph for ResetButton.h:




This graph shows which files directly or indirectly include this file:



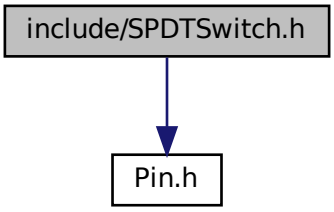

### Classes

- class ResetButton

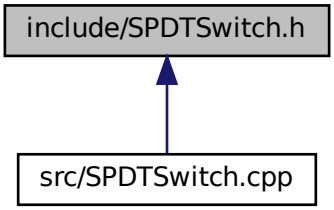
## 5.20 include/SPDTSwitch.h File Reference

```
#include "Pin.h"
```

Include dependency graph for SPDTSwitch.h:




This graph shows which files directly or indirectly include this file:




### Classes

- class SPDTSwitch

### Enumerations

- enum SwitchState { NEUTRAL , STATE_A , STATE_B }

### 5.20.1 Enumeration Type Documentation

#### 5.20.1.1 SwitchState

```
enum SwitchState
```

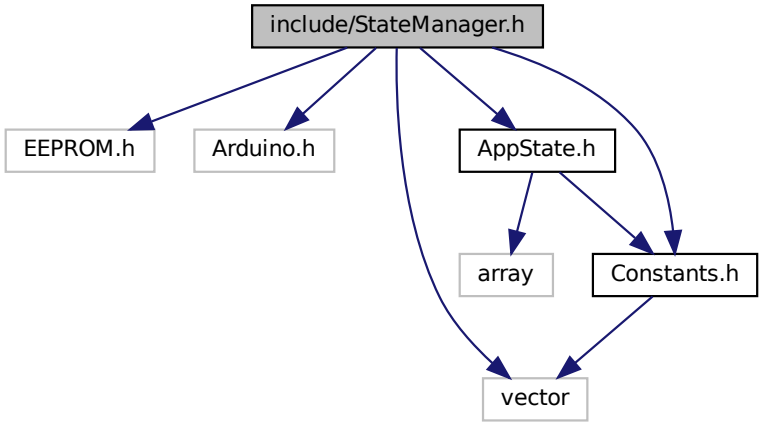
**Enumerator**

| | |
|---|---|
| NEUTRAL | |
| STATE_A | |
| STATE_B | |

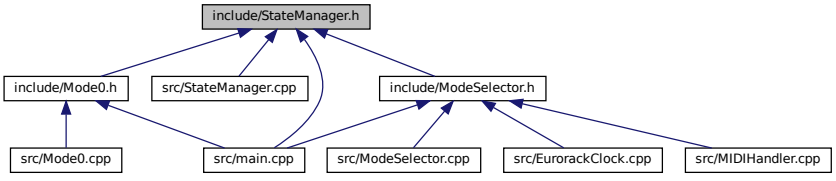
## 5.21 include/StateManager.h File Reference

```
#include <EEPROM.h>
#include <Arduino.h>
#include <vector>
#include "AppState.h"
#include "Constants.h"
```

Include dependency graph for StateManager.h:




This graph shows which files directly or indirectly include this file:



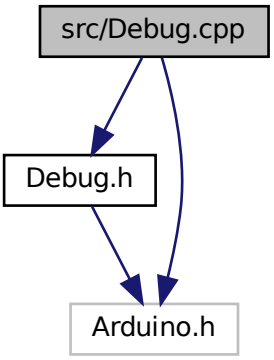

### Classes

- class StateManager

## 5.22 src/Debug.cpp File Reference

```
#include "Debug.h"
#include <Arduino.h>
```

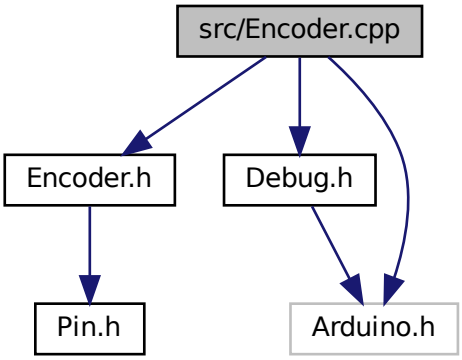
Include dependency graph for Debug.cpp:




## 5.23 src/Encoder.cpp File Reference

```
#include "Encoder.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for Encoder.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.23.1 Macro Definition Documentation

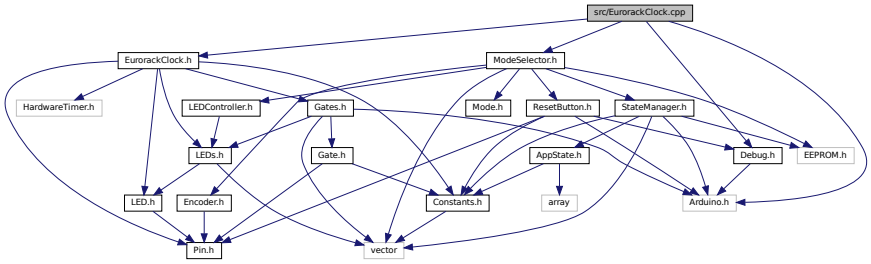
#### 5.23.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
            message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.24 src/EurorackClock.cpp File Reference

```
#include "EurorackClock.h"
#include "Debug.h"
#include <Arduino.h>
#include "ModeSelector.h"
```

Include dependency graph for EurorackClock.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.24.1 Macro Definition Documentation

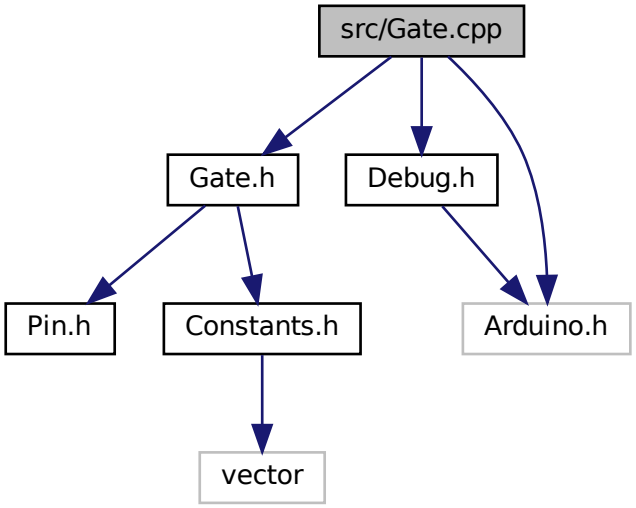
#### 5.24.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
            message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.25 src/Gate.cpp File Reference

```
#include "Gate.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for Gate.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.25.1 Macro Definition Documentation

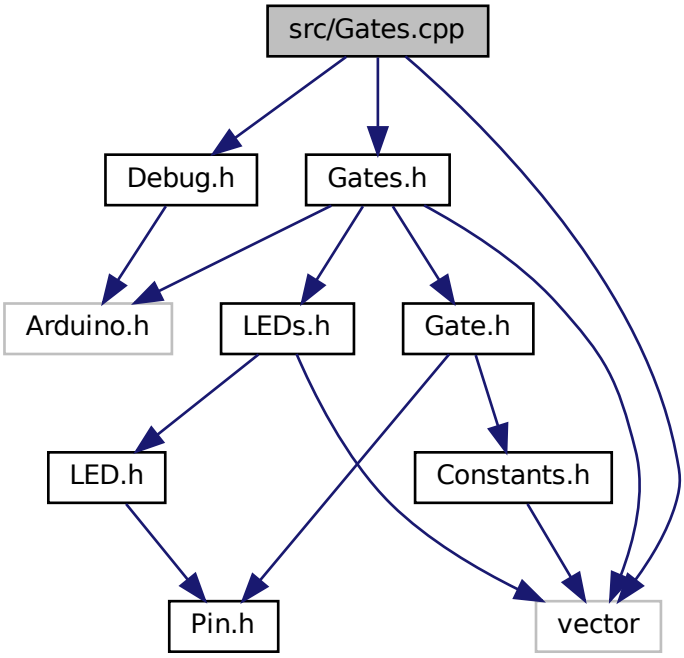
#### 5.25.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.26 src/Gates.cpp File Reference

```
#include "Gates.h"
#include "Debug.h"
#include <vector>
```

Include dependency graph for Gates.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.26.1 Macro Definition Documentation

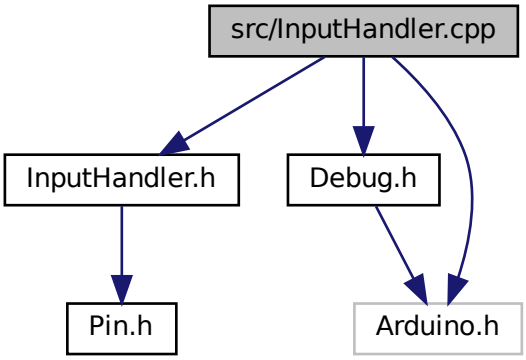
#### 5.26.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.27 src/InputHandler.cpp File Reference

```
#include "InputHandler.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for InputHandler.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.27.1 Macro Definition Documentation

#### 5.27.1.1 DEBUG_PRINT

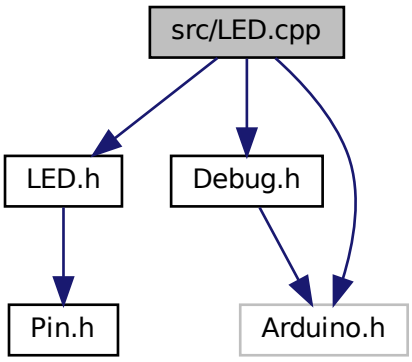
```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.28 src/LED.cpp File Reference

```
#include "LED.h"
#include "Debug.h"
```

```
#include <Arduino.h>
```

Include dependency graph for LED.cpp:




## Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.28.1 Macro Definition Documentation

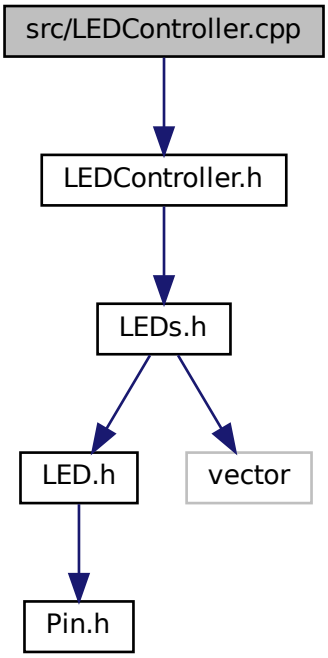
#### 5.28.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.29 src/LEDController.cpp File Reference

```
#include "LEDController.h"
```

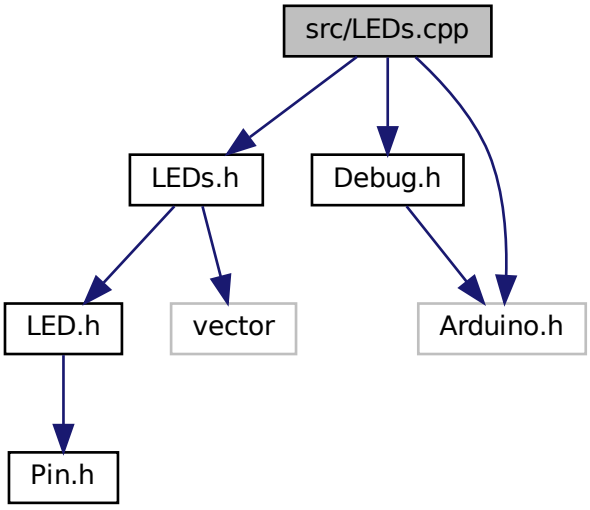
Include dependency graph for LEDController.cpp:




## 5.30 src/LEDs.cpp File Reference

```
#include "LEDs.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for LEDs.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.30.1 Macro Definition Documentation

#### 5.30.1.1 DEBUG_PRINT

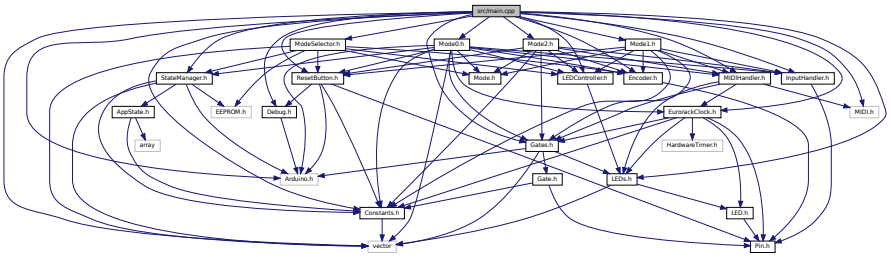
```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.31 src/main.cpp File Reference

```
#include <Arduino.h>
#include <MIDI.h>
#include <vector>
#include "Gates.h"
#include "ModeSelector.h"
#include "LEDs.h"
#include "Debug.h"
#include "Encoder.h"
#include "MIDIHandler.h"
```

```
#include "EurorackClock.h"
#include "Constants.h"
#include "Mode0.h"
#include "Mode1.h"
#include "Mode2.h"
#include "LEDController.h"
#include "ResetButton.h"
#include "InputHandler.h"
#include "StateManager.h"
```

Include dependency graph for main.cpp:




## Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))
- #define RX_PIN PA3
- #define TX_PIN PA2
- #define ENCODER_PINA PB13
- #define ENCODER_PINB PB14
- #define ENCODER_BUTTON PB12
- #define CLOCK_PIN PB10
- #define RESET_PIN PB11
- #define RESET_BUTTON PB15
- #define TEMPO_LED PA8
- #define CV_A_PIN PA4
- #define CV_B_PIN PA5

## Functions

- void setup ()

  *Instance of Mode2 class.*
- void loop ()

  *Main loop function for the Arduino sketch.*

## Variables

- std::vector< int > pins = {PA15, PB3, PB4, PB5, PB6, PB7, PB8, PB9}
- const int numPins = pins.size()

  *Example pins for gates.*
- Gates gates = Gates(pins, numPins)

  *Number of gate pins.*
- std::vector< int > ledPins = {PA12, PA11, PB1, PB0, PA7, PA6, PA1, PA0}

*Create an instance of Gates.*

- int numLedPins = ledPins.size()

  *Placeholder pin numbers for LEDs.*
- LEDs leds = LEDs(ledPins, numLedPins)

  *Number of LED pins.*
- int encCLKPin = ENCODER_PINA

  *Create an instance of LEDs.*
- int encDTPin = ENCODER_PINB

  *Encoder CLK pin.*
- int encButtonPin = ENCODER_BUTTON

  *Encoder DT pin.*
- bool inModeSelection = false

  *Encoder button pin.*
- int intensity = 255

  *Flag for mode selection.*
- bool isInSelection = false

  *Default intensity for LEDs.*
- unsigned long lastFlashTime = 0

  *Flag to prevent multiple presses from being handled.*
- unsigned char internalPPQN = 24

  *Last flash time.*
- std::vector< int > musicalIntervals = {1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 72, 96, 128, 144, 192, 288, 384, 576, 768, 1152, 1536}

  *Pulses per quarter note.*
- const int musicalIntervalsSize = musicalIntervals.size()
- int total_pages = 16 / leds.numLeds

  *Size of musical intervals array.*
- int min_intensity = 64

  *Calculate total pages based on number of LEDs.*
- int intensity_step = (255 - min_intensity) / (total_pages - 1)

  *Set minimum intensity to 25% (64 out of 255)*
- StateManager stateManager = StateManager()

  *Calculate intensity step.*
- Encoder encoder = Encoder(encCLKPin, encDTPin, encButtonPin)

  *Instance of the StateManager class used to manage state of the device in EEPROM.*
- ResetButton resetButton = ResetButton(RESET_BUTTON)

  *Instance of the Encoder class.*
- LEDController ledController (leds)

  *Instance of the ResetButton class.*
- EurorackClock clock (CLOCK_PIN, RESET_PIN, TEMPO_LED, gates, leds)

  *Instance of the LEDController class.*
- MIDIHandler midiHandler (Serial2, clock, gates, leds)

  *Instance of the EurorackClock class.*
- InputHandler inputHandler = InputHandler(CV_A_PIN, CV_B_PIN)

  *Instance of the MIDIHandler class.*
- ModeSelector & modeSelector = ModeSelector::getInstance()

  *Instance of the InputHandler class.*
- Mode * currentMode = nullptr

  *Instance of the ModeSelector class.*
- Mode * previousMode = nullptr

  *Pointer to the current mode.*

- Mode0 mode0 (stateManager, encoder, inputHandler, gates, ledController, midiHandler, resetButton, clock)
  *Pointer to the previous mode.*
- Mode1 mode1 (encoder, inputHandler, gates, ledController, midiHandler, resetButton)
  *Instance of Mode0 class.*
- Mode2 mode2 (encoder, inputHandler, gates, ledController, midiHandler, resetButton)
  *Instance of Mode1 class.*

### 5.31.1 Macro Definition Documentation

#### 5.31.1.1 CLOCK_PIN

```
#define CLOCK_PIN PB10
```

#### 5.31.1.2 CV_A_PIN

```
#define CV_A_PIN PA4
```

#### 5.31.1.3 CV_B_PIN

```
#define CV_B_PIN PA5
```

#### 5.31.1.4 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

#### 5.31.1.5 ENCODER_BUTTON

```
#define ENCODER_BUTTON PB12
```

#### 5.31.1.6 ENCODER_PINA

```
#define ENCODER_PINA PB13
```

#### 5.31.1.7 ENCODER_PINB

```
#define ENCODER_PINB PB14
```

#### 5.31.1.8 RESET_BUTTON

```
#define RESET_BUTTON PB15
```

#### 5.31.1.9 RESET_PIN

```
#define RESET_PIN PB11
```

#### 5.31.1.10 RX_PIN

```
#define RX_PIN PA3
```

#### 5.31.1.11 TEMPO_LED

```
#define TEMPO_LED PA8
```

#### 5.31.1.12 TX_PIN

```
#define TX_PIN PA2
```

### 5.31.2 Function Documentation

#### 5.31.2.1 loop()

```
void loop ( )
```

Main loop function for the Arduino sketch.

This function is called repeatedly as long as the Arduino is powered on. It contains the main logic of the sketch. Update the ModeSelector

Update the LEDController's blinking status

If not in mode selection

Update the current mode

If in mode selection

Set the previous mode to the current mode

Get the new current mode from the ModeSelector

Only run once when the mode is switched to

Teardown the current mode

Setup the new current mode

Set the previous mode to the current mode

#### 5.31.2.2 setup()

```
void setup ( )
```

Instance of Mode2 class.

Setup function for the Arduino sketch.

This function is called once when the sketch starts. It is used to initialize variables, input and output pin modes, and start using libraries. Initialize the debug settings

Enable debugging

Enable to clear and reset EEPROM.

Initialize serial communication

Print debug message

Set the RESET_BUTTON pin to INPUT_PULLDOWN mode

Initialize the MIDIHandler

Set the MIDIHandler to listen to all channels

Start the clock

Set the tempo to 120 BPM with internal 4 PPQN

Initialize the EEPROM with default values

Add Mode0 to the ModeSelector

Add Mode1 to the ModeSelector

Add Mode2 to the ModeSelector

Set the LEDController for the ModeSelector

Set the Encoder for the ModeSelector

Set the StateManager for the ModeSelector

Set the current mode for the ModeSelector

Get the current mode from the ModeSelector

Run the setup function for the current mode

Initialize LED pins

Initialize gate pins

Initialize encoder pins

### 5.31.3 Variable Documentation

#### 5.31.3.1 clock

```
EurorackClock clock(CLOCK_PIN, RESET_PIN, TEMPO_LED, gates, leds) (
             CLOCK_PIN ,
             RESET_PIN ,
             TEMPO_LED ,
             gates ,
             leds  )
```

Instance of the LEDController class.

#### 5.31.3.2 currentMode

```
Mode* currentMode = nullptr
```

Instance of the ModeSelector class.

#### 5.31.3.3 encButtonPin

```
int encButtonPin = ENCODER_BUTTON
```

Encoder DT pin.

#### 5.31.3.4 encCLKPin

```
int encCLKPin = ENCODER_PINA
```

Create an instance of LEDs.

#### 5.31.3.5 encDTPin

```
int encDTPin = ENCODER_PINB
```

Encoder CLK pin.

#### 5.31.3.6 encoder

```
Encoder encoder = Encoder(encCLKPin, encDTPin, encButtonPin)
```

Instance of the StateManager class used to manage state of the device in EEPROM.

#### 5.31.3.7 gates

```
Gates gates = Gates(pins, numPins)
```

Number of gate pins.

#### 5.31.3.8 inModeSelection

```
bool inModeSelection = false
```

Encoder button pin.

#### 5.31.3.9 inputHandler

```
InputHandler inputHandler = InputHandler(CV_A_PIN, CV_B_PIN)
```

Instance of the MIDIHandler class.

#### 5.31.3.10 intensity

```
int intensity = 255
```

Flag for mode selection.

#### 5.31.3.11 intensity_step

```
int intensity_step = (255 - min_intensity) / (total_pages - 1)
```

Set minimum intensity to 25% (64 out of 255)

#### 5.31.3.12 internalPPQN

```
unsigned char internalPPQN = 24
```

Last flash time.

#### 5.31.3.13 isInSelection

```
bool isInSelection = false
```

Default intensity for LEDs.

#### 5.31.3.14 lastFlashTime

```
unsigned long lastFlashTime = 0
```

Flag to prevent multiple presses from being handled.

#### 5.31.3.15 ledController

```
LEDController ledController(leds) (
             leds  )
```

Instance of the ResetButton class.

#### 5.31.3.16 ledPins

```
std::vector<int> ledPins = {PA12, PA11, PB1, PB0, PA7, PA6, PA1, PA0}
```

Create an instance of Gates.

#### 5.31.3.17 leds

```
LEDs leds = LEDs(ledPins, numLedPins)
```

Number of LED pins.

#### 5.31.3.18 midiHandler

```
MIDIHandler midiHandler(Serial2, clock, gates, leds) (
             Serial2 ,
             clock ,
             gates ,
             leds  )
```

Instance of the EurorackClock class.

#### 5.31.3.19 min_intensity

```
int min_intensity = 64
```

Calculate total pages based on number of LEDs.

#### 5.31.3.20 mode0

```
Mode0 mode0(stateManager, encoder, inputHandler, gates, ledController, midiHandler, resetButton,
clock) (
            stateManager ,
            encoder ,
            inputHandler ,
            gates ,
            ledController ,
            midiHandler ,
            resetButton ,
            clock  )
```

Pointer to the previous mode.

#### 5.31.3.21 mode1

```
Mode1 mode1(encoder, inputHandler, gates, ledController, midiHandler, resetButton) (
            encoder ,
            inputHandler ,
            gates ,
            ledController ,
            midiHandler ,
            resetButton  )
```

Instance of Mode0 class.

#### 5.31.3.22 mode2

```
Mode2 mode2(encoder, inputHandler, gates, ledController, midiHandler, resetButton) (
            encoder ,
            inputHandler ,
            gates ,
            ledController ,
            midiHandler ,
            resetButton  )
```

Instance of Mode1 class.

#### 5.31.3.23 modeSelector

```
ModeSelector& modeSelector = ModeSelector::getInstance()
```

Instance of the InputHandler class.

#### 5.31.3.24 musicalIntervals

```
std::vector<int> musicalIntervals = {1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 72, 96, 128,
144, 192, 288, 384, 576, 768, 1152, 1536}
```

Pulses per quarter note.

#### 5.31.3.25 musicalIntervalsSize

```
const int musicalIntervalsSize = musicalIntervals.size()
```

#### 5.31.3.26 numLedPins

```
int numLedPins = ledPins.size()
```

Placeholder pin numbers for LEDs.

#### 5.31.3.27 numPins

```
const int numPins = pins.size()
```

Example pins for gates.

#### 5.31.3.28 pins

```
std::vector<int> pins = {PA15, PB3, PB4, PB5, PB6, PB7, PB8, PB9}
```

#### 5.31.3.29 previousMode

```
Mode* previousMode = nullptr
```

Pointer to the current mode.

#### 5.31.3.30 resetButton

```
ResetButton resetButton = ResetButton(RESET_BUTTON)
```

Instance of the Encoder class.

#### 5.31.3.31 stateManager

```
StateManager stateManager = StateManager()
```

Calculate intensity step.

#### 5.31.3.32 total_pages

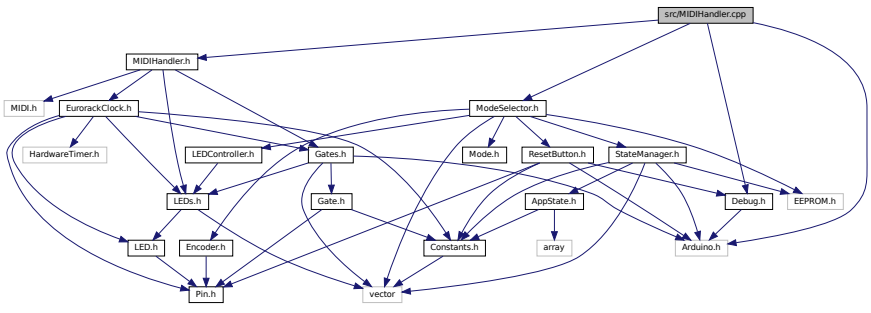
```
int total_pages = 16 / leds.numLeds
```

Size of musical intervals array.

## 5.32 src/MIDIHandler.cpp File Reference

```
#include "MIDIHandler.h"
#include "Debug.h"
#include <Arduino.h>
#include "ModeSelector.h"
```

Include dependency graph for MIDIHandler.cpp:




## Macros

- #define DEBUG_PRINT(message)

### Variables

- bool isInSelection

  *Default intensity for LEDs.*

## 5.32.1 Macro Definition Documentation

#### 5.32.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
              message )
```

**Value:**

```
    { \
    Debug::print(__FILE__, __LINE__, __func__, String(message)); \
    Serial.flush(); \
}
```

## 5.32.2 Variable Documentation

#### 5.32.2.1 isInSelection

```
bool isInSelection  [extern]
```

Default intensity for LEDs.

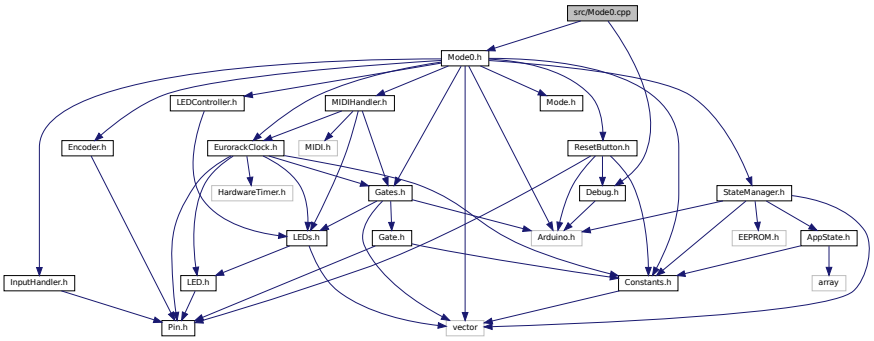
# 5.33 src/Mode.cpp File Reference

# 5.34 src/Mode0.cpp File Reference

Implementation file for Mode0, Please see Mode0.h for more information.

```
#include "Mode0.h"
#include "Debug.h"
```

Include dependency graph for Mode0.cpp:



## Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.34.1 Detailed Description

Implementation file for Mode0, Please see Mode0.h for more information.

### 5.34.2 Macro Definition Documentation

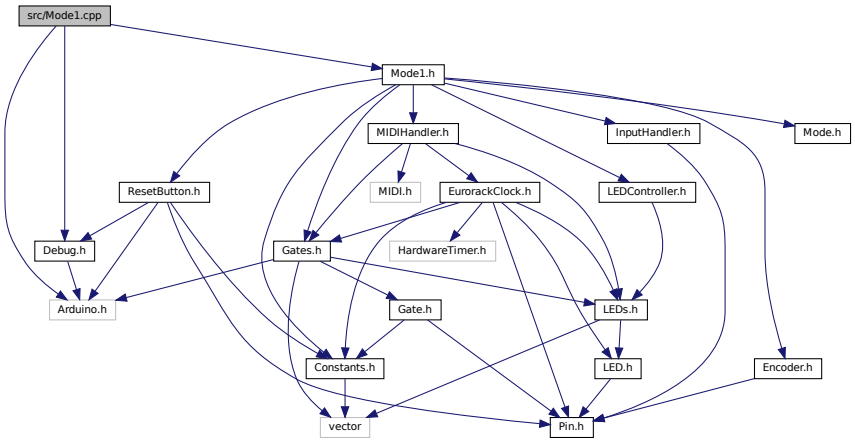
#### 5.34.2.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.35 src/Mode1.cpp File Reference

```
#include "Mode1.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for Mode1.cpp:




## Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.35.1 Macro Definition Documentation

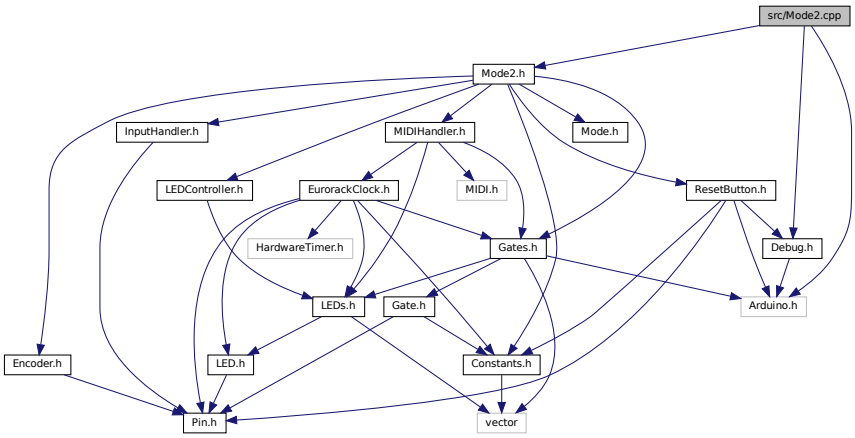
#### 5.35.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.36 src/Mode2.cpp File Reference

```
#include "Mode2.h"
#include "Debug.h"
#include <Arduino.h>
```

Include dependency graph for Mode2.cpp:




### Macros

- #define DEBUG_PRINT(message)

### 5.36.1 Macro Definition Documentation

#### 5.36.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message )
```

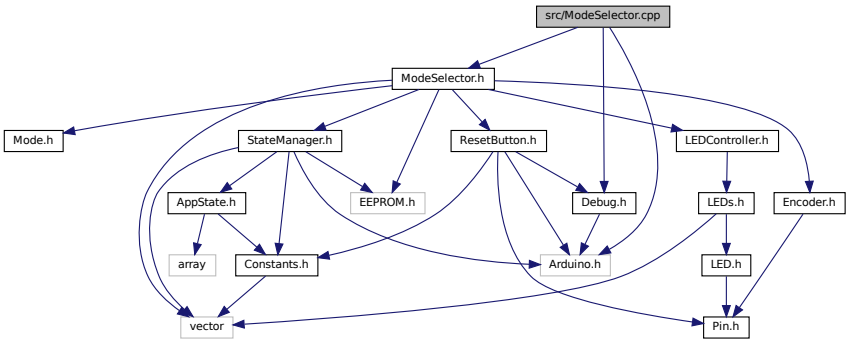
**Value:**

```
    { \
    Debug::print(__FILE__, __LINE__, __func__, String(message)); \
    Serial.flush(); \
}
```

## 5.37 src/ModeSelector.cpp File Reference

```
#include "ModeSelector.h"
#include <Arduino.h>
#include "Debug.h"
```

Include dependency graph for ModeSelector.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.37.1 Macro Definition Documentation

#### 5.37.1.1 DEBUG_PRINT

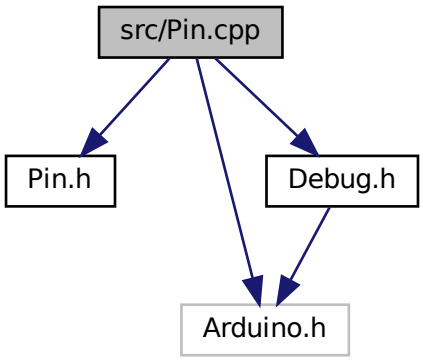
```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.38 src/Pin.cpp File Reference

```
#include "Pin.h"
#include <Arduino.h>
```

```
#include "Debug.h"
```

Include dependency graph for Pin.cpp:




## Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.38.1 Macro Definition Documentation

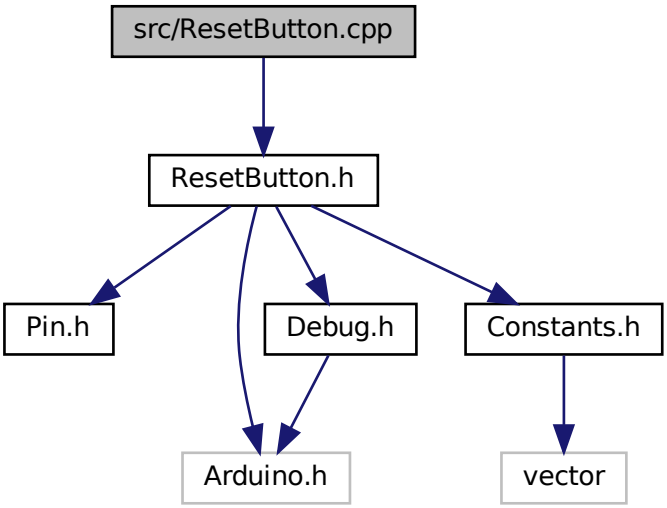
#### 5.38.1.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
             message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

## 5.39 src/ResetButton.cpp File Reference

```
#include "ResetButton.h"
```

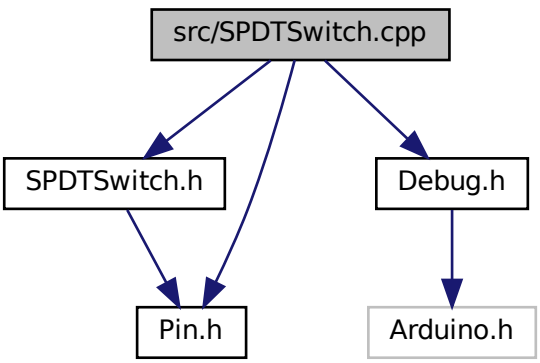
Include dependency graph for ResetButton.cpp:




## 5.40 src/SPDTSwitch.cpp File Reference

```
#include "SPDTSwitch.h"
#include "Pin.h"
#include "Debug.h"
```

Include dependency graph for SPDTSwitch.cpp:

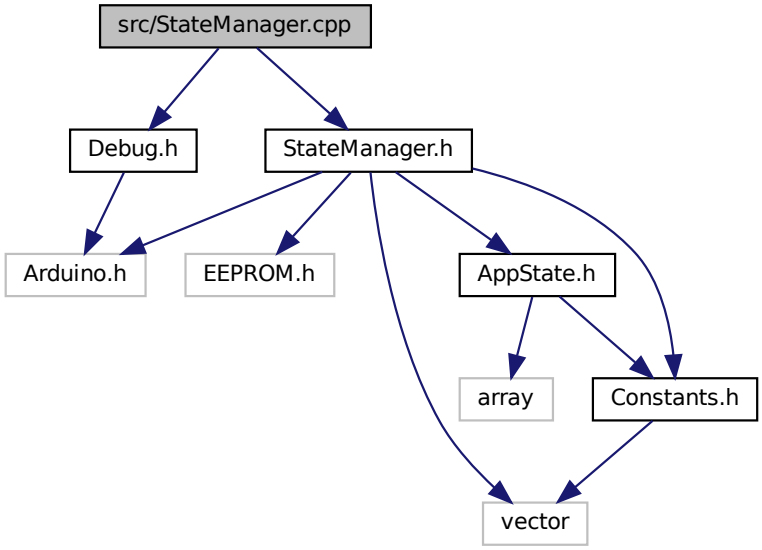


## 5.41 src/StateManager.cpp File Reference

"This class manages reading and writing state to the EEPROM memory."

```
#include "StateManager.h"
#include "Debug.h"
```

Include dependency graph for StateManager.cpp:




### Macros

- #define DEBUG_PRINT(message) Debug::print(__FILE__, __LINE__, __func__, String(message))

### 5.41.1 Detailed Description

"This class manages reading and writing state to the EEPROM memory."

### 5.41.2 Macro Definition Documentation

#### 5.41.2.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
            message ) Debug::print(__FILE__, __LINE__, __func__, String(message))
```

Debug macro