Qwiic PIR Hookup Guide

Introduction

Passive Infrared (PIR) sensors are great for detecting motion in a specific area around the sensor. The SparkFun Qwiic PIR - 170uA (EKMC4607112K) and SparkFun Qwiic PIR - 1uA (EKMB1107112) use two versions of the EKM-series PIR sensors from Panasonic® to offer low profile motion-sensing options over I^2C for both battery powered and continuously powered applications.

SparkFun Qwiic PIR - 170uA (EKMC4607112K)

● SEN-17374

SparkFun Qwiic PIR - 1uA (EKMB1107112)
◉ SEN-17375

PIR sensors do not return specific distance data like distance sensors. Instead, the sensors measure IR light coming from objects in their detection area making them perfect for applications such as controlling power to devices like lights, cameras, screens, etc. automatically when motion is detected.

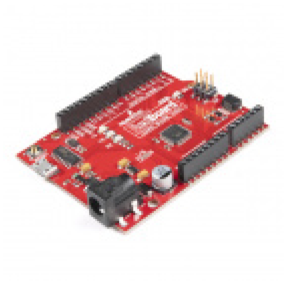
The Qwiic versions of these PIR breakouts feature an ATTiny84 with firmware that handles monitoring the sensor's output signal, debouncing that signal along with a configurable interrupt and translates it all to the I^{2}C interface; making it easy to add a PIR to an existing Qwiic/I^{2}C project.

If you would prefer to directly monitor the sensors' digital outputs, check out our standard breakouts of the 170uA PIR and 1uA PIR.

In this guide we'll cover the hardware present on the Qwiic PIRs, how to connect them to your circuit and we'll finish things up covering both the Arduino Library and Python Package we have written for these sensors along with the examples included in both code packages.

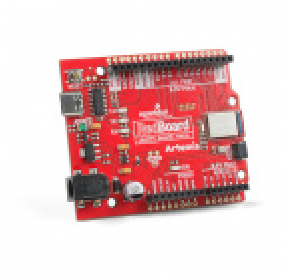
Required Materials

In order to follow along with this tutorial you'll need a few items along with your Qwiic PIR. First, you'll need a microcontroller to communicate with the board. Below are a few options that come Qwiic-enabled out of the box:

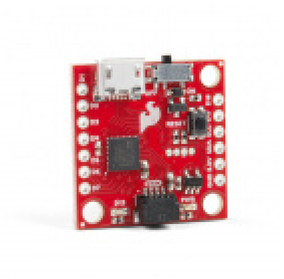


SparkFun RedBoard Qwiic
◉ DEV-15123

SparkFun Thing Plus - ESP32 WROOM
◉ WRL-15663

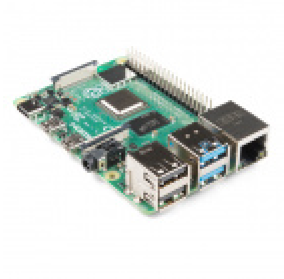


SparkFun RedBoard Artemis
◉ DEV-15444



SparkFun Qwiic Micro - SAMD21 Development Board
◉ DEV-15423

We also have a Python package available for the Qwiic PIRs so you can use a single-board computer (SBC) like a Raspberry Pi or Jetson Nano as your controller as well.



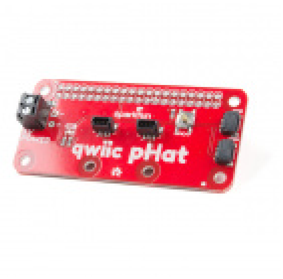
Raspberry Pi 4 Model B (2 GB)
○ DEV-15446

NVIDIA Jetson Nano Developer Kit (V3)
○ DEV-16271

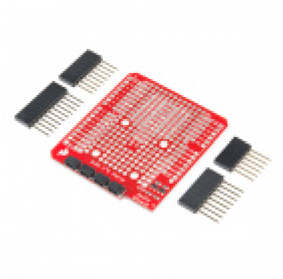
SparkFun DLI Kit for Jetson Nano
◉ KIT-16308

SparkFun Raspberry Pi 4 Desktop Kit - 4GB
○ KIT-16386

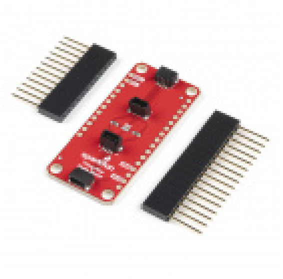
If your preferred microcontroller or SBC does not have a Qwiic connector, you can add one using one of the following products:



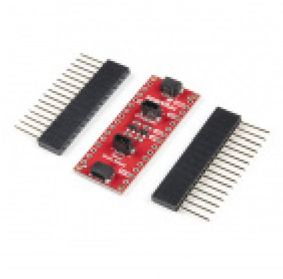
SparkFun Qwiic pHAT v2.0 for Raspberry Pi
◉ DEV-15945



SparkFun Qwiic Shield for Arduino
◉ DEV-14352



SparkFun Qwiic Shield for Thing Plus
◉ DEV-16790



SparkFun Qwiic Shield for Arduino Nano
◉ DEV-16789

Finally, you'll need at least one Qwiic cable to connect your Qwiic PIR to your microcontroller:

Flexible Qwiic Cable - 500mm

◉ PRT-17257

Qwiic Cable - 100mm

◉ PRT-14427

Flexible Qwiic Cable - 50mm

◉ PRT-17260

Qwiic Cable - 200mm

◉ PRT-14428

Suggested Reading

If you aren't familiar with the Qwiic system, we recommend reading here for an overview:

We would also recommend taking a look at the following tutorials if you aren't familiar with the concepts covered in them:

What is an Arduino?

What is this 'Arduino' thing anyway? This tutorials dives into what an Arduino is and along with Arduino projects and widgets.

I2C

An introduction to I2C, one of the main embedded communications protocols in use today.

Serial Terminal Basics
This tutorial will show you how to communicate with your serial devices using a variety of terminal emulator applications.

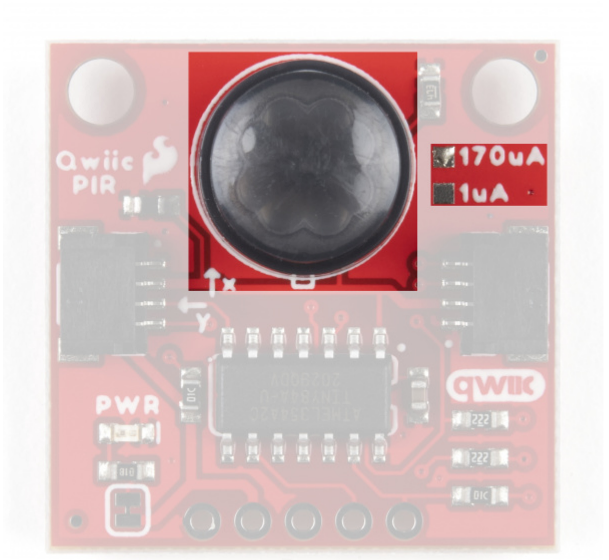
Raspberry Pi 4 Kit Hookup Guide
Guide for hooking up your Raspberry Pi 4 Model B basic, desktop, or hardware starter kit together.

Hardware Overview

In this section we'll cover the characteristics and features of the PIR sensors and other hardware included on the Qwiic PIR boards.

Panasonic EKM-Series PIR Sensors

The EKMC4607112K and EKMB1107112 from Panasonic are low-profile PIR sensors ideal for things like motion-activated lights, cameras or other electronics. Applications include automatic lighting for energy conservation, motion-activated security or trail cameras or maybe something fun like a homemade convenience store chime. The EKMC4607112K works best in a continuous power installation and has slightly better sensing performance than the EKMB1107112 which is best suited for battery and low-power installations.



The version (1μA or 170μA) is marked by one of two solder pads "East" of the PIR sensor.

Input voltage (normally **3.3V**) for the Qwiic PIR is provided either via the Qwiic connectors or through the **3.3V** pin on the PTH header. The Output (OUT) pin is connected to a digital pin on the ATTiny84. Take note that both versions of the Qwiic PIR share the same PCB design and the version (**1μA** or **170μA**) are marked by the solder pads "East" of the PIR sensor.

The two PIR sensors have very similar electrical and sensing characteristics with a few specific differences users will want to take note of prior to deciding which sensor best suits their needs. The tables below outline the Electrical and Detection Performance Characteristics to give users a basic overview. For a more detailed look at these two sensors, take a look at their respective specification sheets (EKMC4607112K & EKMB1107112) along with the Panasonic PIR Sensors - Product Brief (EKM-Series sensors are covered on page 8).

Electrical Characteristics							
		EKMC4607112K			**EKMB1107112**		
Characteristic	**Units**	**Min**	**Typ.**	**Max**	**Min**	**Typ.**	**Max**
Operating Voltage	VDC	3.0	-	6.0	2.3	-	4.0
Current Consumption (Sensor Only)	μA	-	170	300	-	1[1]	3
Output Current	μA	-	-	100	-	-	100
Output Voltage	VDC	VDD-0.5	-	-	VDD-0.5	-	-
Circuit Stability Time (when voltage is applied)	secs	-	-	30	-	25	210

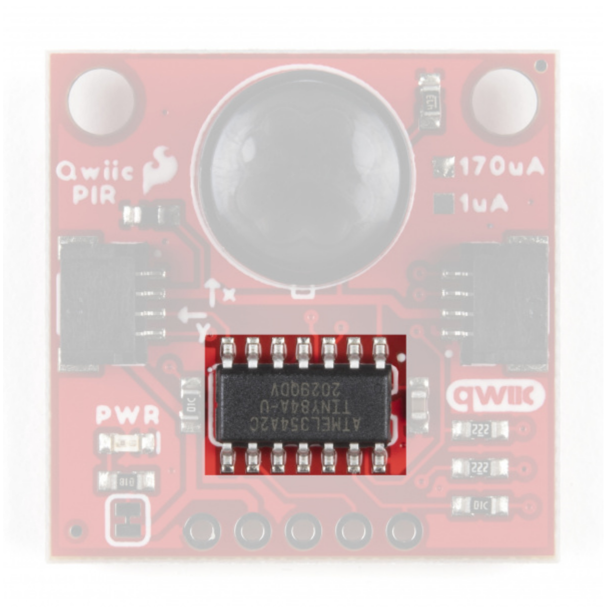
As mentioned above, the sensing performances of the PIR Sensors are very similar with a few notable differences. Also take note that PIR sensor performance can vary depending on the environment it is monitoring.

Detection Performance Characteristics					
	EKMC4607112K		**EKMB1107112**		**Notes**
	Temperature Difference	Value	Temperature Difference	Value	Target Conditions
Detection Range	8°C (14.4°F)	up to 7m	4°C (7.2°F)	up to 7m	1. Movement speed: 1 m/s 2. Target concept is human body (Object size:Around700×250mm)
	4°C (7.2°F)	up to 5m	2°C (3.6°F)	up to 5m	

[1] Note: Current consumption for the EKMB1107112 varies depending on the operating mode. Refer to section 4-4 of the EKMB Spec Sheet for specific values.

ATTiny84 IC

The ATTiny84 on this board comes pre-programmed with firmware to act as an intermediary between the PIR sensor and the microcontroller via Qwiic/I^{2}C. The firmware handles monitoring the raw sensor output to detect objects entering or leaving the sensing area, automatically debouncing the output, triggering an interrupt whenever motion is detected and even storing a queue of sensing events you can pull from and clear.



The default I^{2}C address for the Qwiic PIR is **0x12** but can be switched to **0x13** by adjusting the ADR jumper. Alternatively, users can alter the address to a custom value using functions from the libraries or direct writes to the address register. Software setting of the I^{2}C address is covered in more detail in the Arduino and Python library sections.

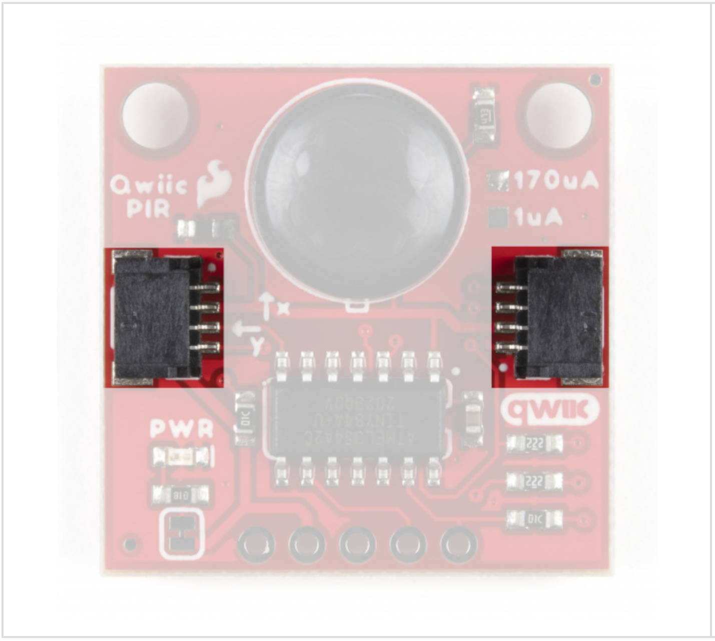
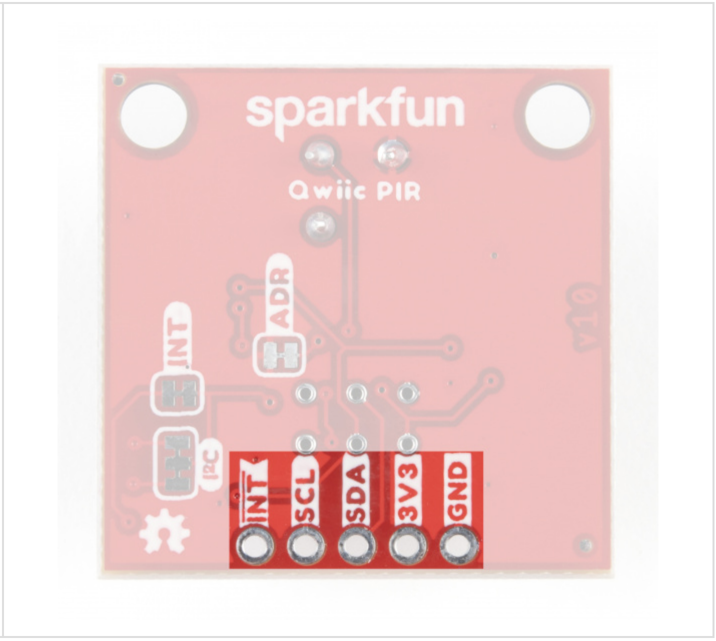
Finally, a 2x3 header broken out on the back of the board allows for programming the ATTiny84. This is primarily used for programming during manufacturing but can be used re-program the IC with custom firmware. You can download and modify the firmware from the Hardware GitHub Repository.

Need help re-programming your Qwiic PIR? Take a look through these tutorials for instructions and tips:

- Tiny AVR Programmer Hookup Guide
- Re-Programming the LilyTiny/LilyTwinkle

Qwiic/I^{2}C Interface and Interrupt Pin

The easiest way to use the Qwiic PIR is with the Qwiic connect system. Connect it to your microcontroller or SBC with a Qwiic Cable to start communicating with it via I^{2}C. For users who prefer a soldered connection, the I^{2}C pins for the Qwiic PIR are broken out to a standard 0.1"-spaced PTH header.

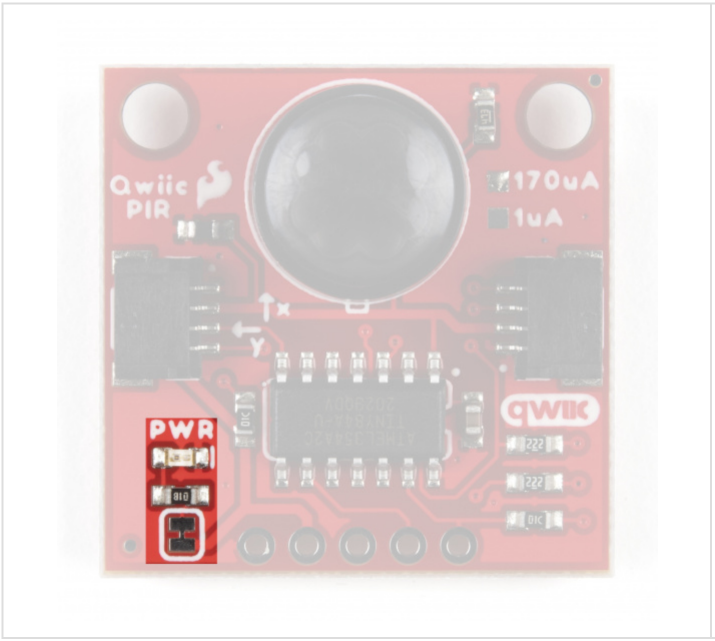
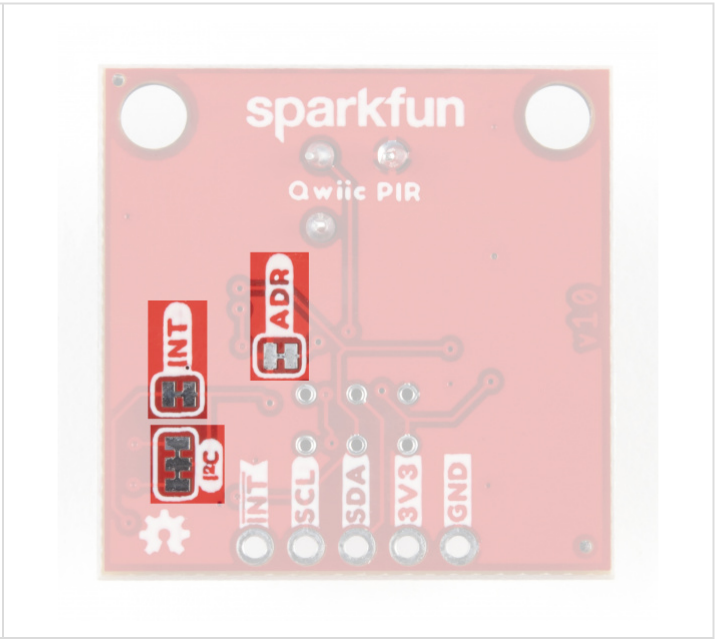
Having trouble viewing the detail in these photos? Click on them for a larger view.

We've also included a dedicated Interrupt pin users can connect to an interrupt-capable pin on their microcontroller to trigger interrupt events based on the activity detected by the Qwiic PIR. Read on to the Arduino and Python sections for more information on how to configure and use this pin.

Solder Jumpers

If you have never worked with solder jumpers and PCB traces before or would like a quick refresher, check out our How to Work with Solder Jumpers and PCB Traces tutorial for detailed instructions and tips.

There are four solder jumpers on the Qwiic PIR boards labeled **PWR**, **I2C**, **INT** and **ADR**. Let's briefly cover each jumper's functionality and default state.

Power Jumper

The Power (PWR) Jumper controls the power LED on the board and is **closed** by default. It ties the anode of the Power LED to **3.3V** via a **1KΩ** resistor. Open the jumper and disable the LED by severing the trace between the two pads. Disabling the Power LED helps to reduce the total current draw of the Qwiic PIR.

I^{2}C Jumper

The I^{2}C Jumper pulls the ATTiny84's SDA and SCL lines to **3.3V** via a pair of **2.2kΩ** resistors. The default state is **closed**. Open the jumper by severing the trace connecting the three pads to disable the pull-up resistors.

If you have more than one device on a single I^{2}C bus, it is recommended to only maintain a single pair of pullup resistors to avoid creating too strong of a parallel resistance. A strong parallel resistance can lead to communication issues on the bus. If you have multiple devices using a single set of pull-up resistors on your I^{2}C bus make sure all devices operate at the same logic level or are properly shifted to avoid damage to the device(s).

Interrupt Jumper

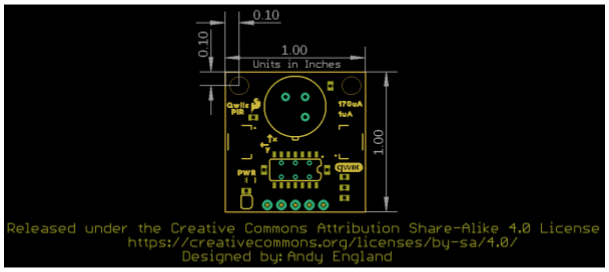
The Interrupt (INT) Jumper pulls the ATTiny84's Interrupt pin to **3.3V** via a **10kΩ** resistor. The default state of the Interrupt jumper is **closed**. The Interrupt Pin on the ATTiny84 is configured as an active LOW interrupt and must be pulled to logic HIGH (**3.3V**) to function properly. Disable the Interrupt pin by opening the jumper.

Address Jumper

The Address (ADR) Jumper sets the I^{2}C address of the ATTiny84. The jumper is **closed** by default. Open the jumper by severing the trace between the two pads to change the I^{2}C address from its default value (**0x12**) to **0x13**.

Board Dimensions

The Qwiic PIR matches the standard 1" x 1" (24.5mm x 24.5mm) form-factor for most Qwiic breakouts and has two mounting holes that fit a 4-40 screw.

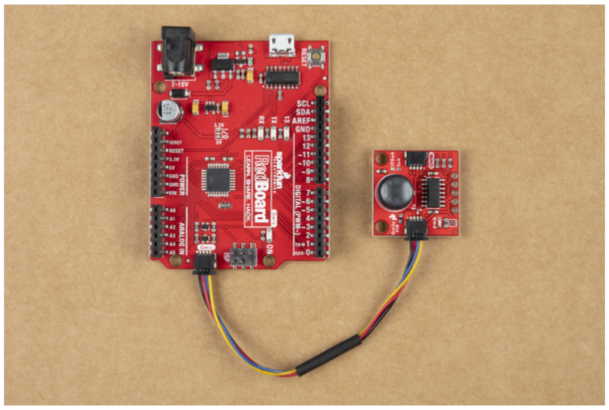

0.10
0.10
1.00
Units in Inches
1.00
Qwiic PIR
170uA
1uA
PWR
qwiic
Released under the Creative Commons Attribution Share-Alike 4.0 License
https://creativecommons.org/licenses/by-sa/4.0/
Designed by: Andy England

Hardware Assembly

With the Qwiic system, assembling your SparkFun Qwiic PIR incredibly easy. We'll cover the basics of hardware assembly here along with a couple of tips for using the Interrupt pin.

Connecting Qwiic Cables

Assuming you are using a Qwiic-enabled development board like the RedBoard Qwiic shown below or you have a Qwiic shield or Qwiic adapter attached to your development board or Raspberry Pi/SBC, all you need to do to connect the Qwiic PIR to your circuit is to plug one end of your Qwiic cable into the Qwiic PIR and the other end into the Qwiic connector on your development board/shield.



Alternatively, you can use one of our adapter cables (male and female) to convert the Qwiic system to a standard jumper wire assembly. If you use one of these adapter cables, make sure you match the signals correctly:

- **Black = GND**
- **Red = 3.3V**
- **Blue = SDA**
- **Yellow = SCL**

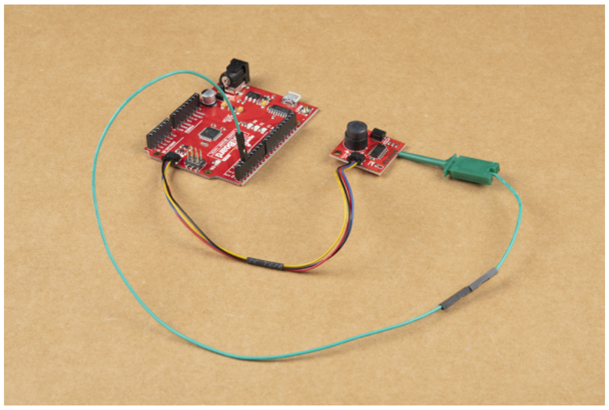
If you prefer to not use the Qwiic connectors you can solder wire or header pins to the PTH header to make your connections.

Connecting Everything Up

If you do not plan to use the Interrupt pin on the Qwiic PIR go ahead and connect your development board to your computer via USB or if you are using a Raspberry Pi or other SBC, power it up.

If you want to use the Interrupt pin, a bit more assembly is required. In order to use the Interrupt pin we need to either solder to it or, for quick prototyping, you can connect it to your development board/SBC using IC hooks like these that terminate in a standard male jumper wire connection.

After you have made your connection to the Qwiic PIR's Interrupt Pin, run that wire to a digital pin available for external interrupts. If you are not sure which pins are interrupt-capable, refer to your board's documentation for clarification. Since we're using the RedBoard Qwiic we can set up D2 or D3 as an interrupt pin. In the circuit below and the Interrupt Example in the Arduino library, the Interrupt pin is connected to D2:



Now that the Qwiic PIR circuit is fully assembled, connect the RedBoard to your PC via USB (or if using a Raspberry Pi or other SBC, power it on) and we can move on to uploading code and monitoring motion.

Qwiic PIR Arduino Library

Note: This example assumes you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please read our Installing an Arduino Library tutorial.

We've written an Arduino library to make it easy to get started and interact with the Qwiic PIR. Before we jump into the examples for reading data from the sensor, we need to install the library and we'll take a closer look at the available functions in the library. The Arduino Library Manager is the easiest way to install the library. Open the Library Manager, search for **"SparkFun Qwiic PIR Arduino Library"** and click the "Install" button to download the latest version. If you prefer manually installing the library from the GitHub repository, you can download it here:

DOWNLOAD THE SPARKFUN QWIIC PIR ARDUINO LIBRARY

Library Functions

The list below outlines all of the functions available in the SparkFun Qwiic PIR Arduino Library along with quick descriptions of what they do. The examples cover nearly all the functions on this list so refer to them to get started or for demonstrations on how to integrate them into your own code.

Class

In the global scope, construct your sensor object (such as `mySensor` or `myPIR`) without arguments.

- `QwiicPIR mySensor;`

Device Setup & Settings

- `bool begin(uint8_t address = DEFAULT_ADDRESS, TwoWire &wirePort = Wire);` - Initializes the Qwiic PIR on the I^{2}C bus. Users can specify an alternate address if it has been changed as well as choose which I^{2}C port used to communicate with the PIR.
- `bool isConnected();` - Returns true if the PIR will acknowledge over I^{2}C and false otherwise.
- `uint8_t deviceID();` - Returns the 8-bit device ID.
- `bool checkDeviceID();` - Returns true if the device ID matches the Qwiic PIR ID.
- `uint8_t getDeviceType();` - Returns `1` if a Qwiic PIR is attached to the bus and `0` if no device is attached.
- `uint16_t getFirmwareVersion();` - Returns the Qwiic PIR firmware version as a 16-bit integer. Major Revision Number = leftmost (high) byte. Minor Revision Number = rightmost (low) byte.
- `bool setI2Caddress(uint8_t address);` - Configures the attached Qwiic PIR to initialize to the bus using the specified address.
- `uint8_t getI2Caddress();` - Returns the I^{2}C address of the attached Qwiic PIR.

PIR Status and Debounce Configuration

These functions are the primary ways to read whether or not an object is detected in the sensing area as well as how to customize the debounce time from the PIR to reduce noise and false detections and how to manipulate the detection queues.

- `bool rawPIRReading();` - Returns `1` when PIR is outputting a signal, `0` when not. This is the raw output from the PIR with no debouncing.
- `bool objectDetected();` - Returns `1` if a debounced object detection event occurs in the sensing area. The debounce time `objectDetected();` waits for is set by the `setDebounceTime(uint16_t time);` .
- `bool ojbectRemoved()` ; - Returns `1` if a debounced object removal event occurs in the sensing area. The debounce time `objectRemoved();` waits for is set by the `setDebounceTime(uint16_t time);` .

- `uint16_t getDebounceTime();` - Returns the debounce time set for the PIR reading to settle (in milliseconds).
- `uint8_t setDebounceTime(uint16_t time);` Sets the time the Qwiic PIR waits for the raw reading from the sensor to settle. The default value for debounce time is 750ms.

Interrupt Status and Configuration

- `uint8_t enableInterrupt();` - When called, the interrupt pin is configured to trigger on all PIR events (detection & removal).
- `uint8_t disableInterrupt();` - When called, the interrupt pin is no longer configured to trigger on PIR events.
- `bool available();` - Returns the `eventAvailable` bit.
- `uint8_t clearEventBits();` - Sets `objectDetected` , `objectRemoved` and `eventAvailable` bits to zero.
- `uint8_t resetInterruptConfig();` - Resets any configured interrupt functions to defaults (OFF).

Queue Manipulation

- `bool isDetectedQueueFull();` - Returns true if queue of object detections timestamps is *full* and false if not. This queue stores ten timestamp values.
- `bool isDetectedQueueEmpty();` - Returns true if the queue of object detections timestamps is *empty* and false otherwise.
- `unsigned long timeSinceLastDetect();` - Returns the time (in milliseconds) since the `objectDetected();` function last returned `1` .
- `unsigned long timeSinceFirstDetect();` - Returns the time (in milliseconds) for the oldest value stored in the Detected Queue.
- `unsigned long popDetectedQueue();` - Returns the oldest value stored in the Detected Queue and removes it.
- `bool isRemovedQueueFull();` - Returns true if the object removals queue is *full* and false if not. This queue stores ten timestamp values.
- `bool isRemovedQueueEmpty();` - Returns true if the object removals queue is *empty* and false otherwise.
- `unsigned long timeSinceLastRemove();` - Returns the time (in milliseconds) since the `objectRemoved();` function last returned `1` .
- `unsigned long timeSinceFirstRemove();` - Returns the time (in milliseconds) for the oldest value stored in the Removed Queue.
- `unsigned long popRemovedQueue();` - Returns the oldest value stored in the Removed Queue and removes it.

Arduino Examples

The SparkFun Qwiic PIR Arduino Library includes five examples to help users get started with the board and library. In this section we'll go over a few of the examples and highlight how they work.

Example 1 - Print Raw PIR Status

The first example demonstrates how to set up the Qwiic PIR on your I^{2}C bus and then retrieve raw data readings from the Qwiic PIR using the `rawPIRReading();` function. Open the first example by navigating to **File > Examples > SparkFun Qwiic PIR Arduino Library > Example1_PrintRawPIRStatus**. Next, open the Tools menu and select your board (in our case, Arduino Uno) and the correct Port your board enumerated on. Upload the code and open your serial monitor with the baud set to **115200**.

The example first sets up the Qwiic PIR object and debounce time (in milliseconds):

```
QwiicPIR pir;

#define DEBOUNCE_TIME 750
```

Next it initializes the sensor and then waits 30 seconds for the PIR to warm up and stabilize.

```
if (pir.begin() == false) {
    Serial.println("Device did not acknowledge! Freezing.");
    while (1);
  }

  Serial.println("PIR acknowledged. Waiting 30 Seconds while PIR warms up");
  for (uint8_t seconds = 0; seconds < 30; seconds++)
  {
    Serial.println(seconds);
    delay(1000);
  }
```

The code will freeze if the Qwiic PIR does not acknowledge on the I^{2}C bus at the default address. A bad connection or a different I^{2}C address are the most common causes of this failure.

Once the PIR warms up the main loop checks whether `rawPIRReading();` returns `TRUE` or `FALSE` and waits to update again for the value set for `DEBOUNCE_TIME` . The code prints out objects detected or removed via serial. Take note when using the `rawPIRReading();` any debouncing of the signal must be done manually.

Example 2 - Print PIR Status

The second example is very similar to the first but uses the `objectDetected();` and `objectRemoved();` functions instead of `rawPIRReading();` . The primary difference between these functions is where debouncing the PIR signal happens. Instead of manually debouncing the PIR signal each time it occurs, the `objectDetected();` and `objectRemoved();` functions refer to the value stored for `setDebounceTime(uint16_t time);` and will automatically debounce the signal for that time. The default value for `setDebounceTime();` is 750ms.

Open the example, upload it and open a serial terminal set to **115200** baud. After initializing the sensor and waiting for 30 seconds for the PIR to warm up, the code will start polling the PIR for events and prints what they are over serial:

```
if (pir.available()) {
    if (pir.objectDetected()) {
        Serial.println("Object Detected");
    }
    if (pir.objectRemoved()) {
        Serial.println("Object Removed");
    }
    pir.clearEventBits();
}
```

Example 3 - Queue Usage

The third example included with the library shows how to read and manipulate the Object Detected and Object Removed queues. After uploading the example, open a serial terminal with the baud set again to **115200**. After initializing the Qwiic PIR, the main loop checks if either the Detected or Removed queues have values stored for either time (in seconds) since first detect/remove or time since last detect/remove and prints them over serial. If no values are present in either queue, the code prints out which queue is empty:

```
if(pir.isDetectedQueueEmpty() == false) {
    Serial.print(pir.timeSinceLastDetect()/1000.0);
    Serial.print("s since PIR detect   ");
    Serial.print(pir.timeSinceFirstDetect()/1000.0);
    Serial.print("s since PIR detect   ");
}

if(pir.isDetectedQueueEmpty() == true) {
    Serial.print("PIR Detected Queue is empty! ");
}

if(pir.isRemovedQueueEmpty() == false) {
    Serial.print(pir.timeSinceLastRemove()/1000.0);
    Serial.print("s since PIR remove   ");
    Serial.print(pir.timeSinceFirstRemove()/1000.0);
    Serial.print("s since PIR remove   ");
}
if(pir.isRemovedQueueEmpty() == true) {
    Serial.print("  PIR Removed Queue is empty!");
}
```

Along with printing values from each queue, this example also shows how to manipulate and pop values from any queue:

```
if(Serial.available()) {

    uint8_t data = Serial.read();
    if(data == 'd' || data == 'D') {
        pir.popDetectedQueue();
        Serial.println("Popped DetectedQueue!");
    }

    if(data == 'r' || data == 'R') {
        pir.popRemovedQueue();
        Serial.println("Popped RemovedQueue!");
    }
}
delay(20);
```

With a serial terminal open, send the letter "D" (capitalized or not) to pop a value off the Detected Queue. Similarly, send the letter "R" to pop a value off the Removed Queue.

Example 4 - External Interrupt

Example 4 - ExtInterrupt demonstrates how to use the Interrupt pin on the Qwiic PIR.

Along with setting up the Qwiic PIR in the global class, the code defines the interrupt pin and creates an interrupt flag:

```
int interruptPin = 2;

bool interruptEntered = false;
```

Adjust the value for your interrupt pin as needed. This example assumes a SparkFun RedBoard/Arduino Uno is used and sets D2 as the interrupt pin. In the setup, the code initializes the Interrupt pin as an input and attaches it as a falling-edge interrupt to a custom function called `pirHandler` :

```
pinMode(interruptPin, INPUT);
attachInterrupt(digitalPinToInterrupt(interruptPin), pirHandler, FALLING);
```

After initializing the Qwiic PIR on the bus and waiting for 30 seconds for the PIR to warm up, the code calls the `enableInterrupt();` and `clearEventBits();` functions to tell the Qwiic PIR to trigger the pin on object events and clears any event bits to toggle the Interrupt pin `HIGH` :

```
pir.enableInterrupt();
pir.clearEventBits();
```

The main loop checks for motion events and if any are detected it will fire the interrupt pin:

```
void loop() {
  if (interruptEntered)
  {
    if (pir.objectDetected()) {
      Serial.println("Object Detected");
    }
    pir.clearEventBits();
    interruptEntered = false;
    delay(10);
  }
}

void pirHandler()
{
  interruptEntered = true;
}
```

From here, you can modify this example or insert it into more complex code to trigger whatever interrupt event you would like using the Qwiic PIR.

Qwiic PIR Python Package

Note: This tutorial assumes you are using the latest version of Python 3. If this is your first time using Python or I^{2}C hardware on a Raspberry Pi these tutorials will help you get started:

- Python Programming with the Raspberry Pi
- Raspberry Pi SPI and I2C Tutorial

We've written a Python package for the Qwiic PIR for users who prefer to use something like a Raspberry Pi with their sensor. The package can be installed with the all inclusive SparkFun Qwiic Python package or independently.

We recommend installing the entire SparkFun Qwiic Package as it also installs the required I^{2}C driver.

Note: Don't forget to double check that the hardware I^{2}C connection is enabled on your Raspberry Pi or other single board computer. Make sure to reboot your Pi after enabling the I^{2}C bus for changes to take effect.

SparkFun Qwiic Package

This repository is hosted on PyPi as the `sparkfun-qwiic` package. On systems that support PyPi installation via `pip3` (use `pip` for Python 2) is simple using the following commands:

For **all users** (Note: the user must have **sudo** privileges):

```
sudo pip3 install sparkfun-qwiic
```

For the **current user**:

```
pip3 install sparkfun-qwiic
```

Independent Qwiic PIR Py Package Installation

If you prefer to only install the Qwiic PIR package, you can download the `sparkfun-qwiic-pir` Python package hosted by PyPi via `pip3` following the instructions below. Alternatively, if you prefer to manually download and build the libraries you can grab them from the Qwiic PIR Py GitHub Repository or by clicking the button below:

DOWNLOAD THE QWIIC PIR PY REPOSITORY

PyPi Installation

This repository is hosted on PyPi as the `sparkfun-qwiic-PIR` package. On systems that support PyPi, install the `sparkfun-qwiic-PIR` package via `pip3` (use `pip` for Python 2) using the following commands:

For **all users** (Note: the user must have **sudo** privileges):

```
sudo pip3 install sparkfun-qwiic-PIR
```

For the **current user**:

```
pip3 install sparkfun-qwiic-PIR
```

Local Installation

To install, make sure the `setuptools` package is installed on the system.

Direct installation at the command line (use `python` for Python 2):

```
python3 setup.py install
```

To build a package for use with `pip3` :

```
python3 setup.py sdist
```

A package file is built and placed in a subdirectory called dist. This package file can be installed using `pip3` .

```
cd dist
pip3 install sparkfun_qwiic_PIR-<version>.tar.gz
```

Qwiic PIR Python Package Operation

Let's take a quick look at the functions available in the Python package. For more details on how the package works, take a look at the source code and package documentation hosted on ReadTheDocs.

Dependencies

This Python package has a few dependencies in the code, listed below:

```
import time
import sys
```

Default Variables

```
# Define the device name and I2C addresses. These are set in the class definition
# as class variables, making them available without having to create a class instance.
# This allows higher level logic to rapidly create an index of qwiic devices at runtime.

# This is the name of the device
_DEFAULT_NAME = "Qwiic PIR"

# Some devices have  multiple available addresses - this is a list of these addresses.
# NOTE: The first address in this list is considered the default I2C address for the
# device.
_AVAILABLE_I2C_ADDRESS = [0x12]
```

Note: This package differs from other SparkFun packages as the register values are declared in the object class.

```
# Constructor
device_name = _DEFAULT_NAME
available_addresses = _AVAILABLE_I2C_ADDRESS

# Device ID for all Qwiic PIRs
DEV_ID = 0x72

# Registers
ID = 0x00
FIRMWARE_MINOR = 0x01
FIRMWARE_MAJOR = 0x02
EVENT_STATUS = 0x03
INTERRUPT_CONFIG = 0x04
EVENT_DEBOUNCE_TIME = 0x05
DETECTED_QUEUE_STATUS = 0x07
DETECTED_QUEUE_FRONT = 0x08
DETECTED_QUEUE_BACK = 0x0C
REMOVED_QUEUE_STATUS = 0x10
REMOVED_QUEUE_FRONT = 0x11
REMOVED_QUEUE_BACK = 0x15
I2C_ADDRESS = 0x19

# Status Flags
eventAvailable = 0
objectRemove = 0
objectDetect = 0
rawObjectDetected = 0

# Interrupt Configuration Flags
interruptEnable = 0

# Queue Status Flags
popRequest = 0
isEmpty = 0
isFull = 0
```

Class

`QwiicPIR()` or **`QwiicPIR(address)`**

This Python package operates as a class object, allowing new instances of that type to be made. An `__init__()` constructor is used that creates a connection to an I^{2}C device over the I^{2}C bus using the default or specified I^{2}C address.

Note: If the Qwiic PIR's address has been altered from the default (**0x12**), create the Qwiic PIR object with the new address. For example, if the address jumper is opened create the Qwiic PIR object using this format: `QwiicPIR(0x13)`.

The Constructor

A constructor is a special kind of method used to initialize (assign values to) the data members needed by the object when it is created.

`__init__(address=None, i2c_driver=None):`

Input: value
: The value of the device address. If not defined, the Python package will use the default I^{2}C address (**0x12**) stored under `_AVAILABLE_I2C_ADDRESS` variable.

Input: *i2c_driver*
: Loads the specified I^{2}C driver; by default the Qwiic I^{2}C driver is used: `qwiic_i2c.getI2CDriver()`. Users should use the default I^{2}C driver and leave this field blank.

Functions

A function is an attribute of the class, which defines a method for instances of that class. In simple terms, they are objects for the operations (or methods) of the class. For a complete list of all the available functions, head over to the API Reference page of ReadtheDocs for the Qwiic PIR Py Python Package.

Upgrading the Package

Updating the installed packages has to be done individually for each package (i.e. sub-modules and dependencies won't update automatically and must be updated manually). For the `sparkfun-qwiic-pir` Python package, use the following command (use `pip` for Python 2):

For **all users** (note: the user must have **sudo** privileges):

```
sudo pip3 install --upgrade sparkfun-qwiic-pir
```

For the **current user**:

```
pip3 install --upgrade sparkfun-qwiic-pir
```

Python Examples

The SparkFun Qwiic PIR Python Library includes four examples to help users get started with the board and library. In this section we'll go over the examples and highlight how they work.

To use the examples, open them from the Python library's location or copy the code into your preferred Python interpreter.

Example 1 - Simple Example (Raw PIR Readings)

The first example shows how to set up the Qwiic PIR on the I^{2}C bus and retrieve raw data from the PIR's output signal. Because we are reading raw PIR output the code manually sets up a debounce time (in milliseconds) to wait for the PIR output signal to stabilize. Adjust the debounce time by changing this value:

```
debounce_time = .20
```

The main example loop sets up the PIR object, attempts to initialize it on the I^{2}C bus and, if successful, waits 30 seconds for the PIR to stabilize:

```
def run_example():


        print("\nSparkFun Qwiic PIR Example1\n")
        my_PIR = qwiic_pir.QwiicPIR()

        if my_PIR.begin() == False:
                print("The Qwiic PIR isn't connected to the system. Please check your connectio
n", \ file=sys.stderr)

                return
        print ("Waiting 30 seconds for PIR to stabilize")
        for i in range(0,30):
                print(i)
                time.sleep(1)

        print("Device Stable")
```

Once the PIR has initialized and stabilized, the code begins to take readings and print out whether an object was detected or removed, pausing for the value set for `debounce_time` after each reading:

```
while True:
        if my_PIR.raw_reading() is True:
                print("Object Detected")
        else:
                print("Object Removed")
        time.sleep(debounce_time)
```

Example 2 - Debounced Readings

The second example demonstrates how to read the debounced output signals from the Qwiic PIR using the `object_detected()` and `object_removed()` functions. Just like with the Arduino library, the Qwiic PIR Python Library includes functions for both raw PIR readings as well as automatically debounced readings.

Using the `object_detected()/removed()` functions allows you to set a debounce time with the `set_debounce_time()` function and the PIR will always wait for that amount of time before taking another reading. The default value for `set_debounce_time()` is 750ms.

The code initializes the Qwiic PIR on the bus and waits for 30 seconds for the PIR to stabilize prior to taking readings for object detections:

```
while True:
        if my_PIR.available() is True:
                if my_PIR.object_detected():
                        print("Object Detected")
                if my_PIR.object_removed():
                        print("Object Removed")
                my_PIR.clear_event_bits()
        time.sleep(1)
```

Example 3 - Queue Usage

The third example shows how to read values stored for the Object Detected and Object Removed queues. After initializing the sensor and waiting for it to stabilize the code then prints out values stored in Detected Queue & Removed Queue for both time (in seconds) since the last (most recent) object detection or removal event as well as the time since the oldest stored object detection or removal event. If either queue is empty, the code prints out which queue is empty.

```
while True:
        if my_PIR.is_detected_queue_empty() is False:
                last_detect = my_PIR.time_since_last_detect() / 1000.0
                first_detect =  my_PIR.time_since_first_detect() / 1000.0
                print(last_detect)
                print("s since last PIR detect   ")
                print(first_detect)
                print("s since first PIR detect   ")
        else:
                print("Detected queue is empty")

        if my_PIR.is_removed_queue_empty() is False:
                last_remove = my_PIR.time_since_last_remove() / 1000.0
                first_remove =  my_PIR.time_since_first_remove() / 1000.0
                print(last_remove)
                print("s since last PIR detect   ")
                print(first_remove)
                print("s since first PIR detect   ")
        else:
                print("Removed queue is empty")
```

Example 4 - Pop Queue

Example 4 demonstrates how to pop values from both the `object_detected()` and `object_removed()` queues by sending the appropriate characters over serial.

Just like the other examples, the Qwiic PIR is initialized on the I^{2}C bus and the code waits for 30 seconds for the PIR to stabilize it's readings. After waiting, the code prints out to enter either "d" or "r" to pop values from either the detected (d) or removed (r) queues:

```
while True:
    print("\mType 'd' to pop from the detected queue.")
    val = raw_input("Type 'r' to pop from the removed queue: ")
    # If the character is 'd' or 'D', then pop a value off the detected queue
    if val == 'd' or val == 'D':
        print("\nPopped detected queue! The first timestamp in detected queue was: ")
        print(str(my_PIR.pop_detected_queue() / 1000.0))

    # If the character is 'r' or 'R', then pop a value off the removed queue
    if val == 'r' or val == 'R':
        print("\nPopped removed queue! The first timestamp in removed queue was: ")
        print(str(my_PIR.pop_removed_queue90 / 1000.0))

    time.sleep(debounce_time)
```

If the correct value is entered the code prints out over serial the respective queue has been popped and prints out the timestamp for the removed value in seconds.

Troubleshooting

Assembling and testing the Qwiic PIR is fairly straight-forward but in case you run into any issues we've outlined a few tips and tricks for testing the PIR here.

Detection Area/Field of View

The effective detection area of both the EKMC4607112K and EKMB1107112 is dependent on a variety of factors. The specifications for measurement range are based on a target concept (area of ~700×250mm) of a human body moving across two detection zones at a speed of 1m/s. The PIR senses objects best when moving across two detection zones on the horizontal (X) or vertical (Y) axes. The PIR may struggle to detect objects moving away or toward the PIR (along the Z axis) unless they also move along the other two axes.

Also note that background IR radiation can influence the PIR's ability to detect an object. The PIR can detect objects with a larger temperature difference from the background at a larger range. Refer back to the Hardware Overview section for range specifications at different temperature differences.

Take these detection factors into consideration when selecting the mounting position of your Qwiic PIR. Section 4-7 of the sensors' spec sheets (EKMC4607112K and EKMB1107112) show diagrams for optimal sensor placement and object motion for sensing performance.

Qwiic PIR Not Recognized on I^{2}C Bus

The examples included in both the Arduino and Python libraries will freeze if the Qwiic PIR does not acknowledge on the I^{2}C bus. The most common cause of this is a poor or incomplete connection either using the Qwiic connectors or PTH header. Check your Qwiic cables for a secure connection to the Qwiic connectors or for damage. If using the PTH header, double check your solder joints and wires to make sure they are complete and secure.

Another common cause for this error is if the Qwiic PIR is set to an alternate address. The examples assume the PIR uses the default I^{2}C address (**0x12**) and will freeze if the code is not adjusted to reflect a change in address. For example, if the ADR jumper is set to switch the address to **0x13**, the `begin();` function in the Arduino Library can be adjusted as follows:

```
pir.begin(0x13);
```

Similarly, the Qwiic PIR object can be created in the Python package at an alternate address using the following code:

```
QwiicPIR(0x13)
```

Finally, Raspberry Pi users encountering this error should check to make sure the I^{2}C bus is enabled. Refer to this section of our Raspberry Pi SPI & I^{2}C tutorial for detailed instructions on enabling the bus.

General Troubleshooting

If you need technical assistance and more information on this or another SparkFun product that is not working as you expected, we recommend heading on over to the SparkFun Technical Assistance page for some initial troubleshooting:

SPARKFUN TECHNICAL ASSISTANCE PAGE

If you don't find what you need there, the SparkFun Forums are a great place to find and ask for help. If this is your first visit, you'll need to create a Forum Account to search product forums and post questions.

Resources and Going Further

For more information about the SparkFun Qwiic PIR boards, take a look at theses resources:

- Schematic (PDF)
- Eagle Files (ZIP)
- Dimensional Drawing (PNG)
- Panasonic PIR Motion Sensors - Product Brief
- EKMB110711x Spec Sheet
- EKMC460711xK Spec Sheet
- SparkFun Qwiic PIR Arduino Library
- Qwiic PIR Python Library
- GitHub Hardware Repo
- Qwiic Landing Page

Looking for some project inspiration to use your Qwiic PIR in? Check out these resources:

- PIR "Doorbell" Gag Project
- PIR Primer and PIR Project Blog Post

Vernier Photogate

Vernier Photogate Timer -- using the Serial Enabled LCD Kit.

Photon Remote Water Level Sensor

Learn how to build a remote water level sensor for a water storage tank and how to automate a pump based off the readings!

Are You Okay? Widget

Use an Electric Imp and accelerometer to create an "Are You OK" widget. A cozy piece of technology your friend or loved one can nudge to let you know they're OK from half-a-world away.

Boss Alarm

Build a Boss Alarm that alerts you of anyone walking into your office and automatically changes your computer screen.