Cpp attachInterrupt to class function help [solved]

SomeFixItDude

Jun '14

I am pretty new to cpp and find I am fumbling through it, but need some help and or explanation. I am trying to make a class that in the constructor attaches a member function to an interrupt. Can I do this?? I am sure I am just missing something fundamentally about pointers or something. Here is my example code:

help.ino

```
#include "foo.h"

foo myFoo = foo(D0);

void setup() {

}

void loop() {

}
```

foo.h

```
#pragma once
#include "application.h"

class foo {
    public:
        foo(int channelA);
        void doChannelA();
        volatile int position = 0;
};
```

foo.cpp

```
#include "foo.h"

foo::foo(int channelA) {
    pinMode(channelA, INPUT);

    // Setup interrupts
    attachInterrupt(channelA, doChannelA, RISING);
};

void foo::doChannelA() {
    position++;
}
```

No Joy message:

```
/foo.cpp:7:49: error: cannot convert 'foo::doChannelA' from type 'void (foo::)()' to type '
```

Thanks Again for anyone's time for the help,
Brian

dpursell Jun '14

Hi Brian,

Using C++ in embedded environments is an area that interests me quite a bit, so I will probably ramble a bit. If you just want a quick solution feel free to skip to the end of this post 😄

The problem is that the attachInterrupt() function requires a non-member function, since it doesn't have any way to know what foo object you want to call your doChannelA() function on (recall that class member functions are always called on an object, e.g. fooObject.doChannelA()). This leaves us with the choice of a static class function or a global function, neither of which require an object to act on.

I greatly prefer static class functions when possible, but to make them clean, the interrupt function would have to provide a user parameter so that the interrupt provides the object we want to call doChannelA() on. It would look something like this:

*** Not a real solution, just wishful thinking ***

Foo class declaration (side note - standard C++ style is to capitalize class names)

```
class Foo {
 public:
  Foo(int channelA);
  void doChannelA();
  static void channelAInterruptDispatch(void* userData);
  volatile int position = 0;
};
```

Foo class implementation

```
Foo::Foo(int channelA) {
  pinMode(channelA, INPUT);

  // Here we attach our interrupt AND provide user data that the
  // interrupt will give back to us whenever it fires. Unfortunately
  // this is unavailable in the Spark library (probably for
  // compatibility with Arduino, but I'm not sure)
  attachInterrupt(channelA, &Foo::channelAInterruptDispatch, this, RISING);
}

void Foo::doChannelA() {
  position++;
}

// static interrupt function
void Foo::channelAInterruptDispatch(void* userData) {
  // We tell the compiler that we are SURE the userData parameter
  // is in fact a Foo object and it lets us call the actual
  // doChannelA interrupt function
  static_cast<Foo*>(userData)->doChannelA();
}
```

However, since the interrupt functions don't allow for a custom parameter to be passed in and out, we are forced to create a unique global function (or a unique static class function) for each interrupt we want to attach. So here's my solution I use in Spark, but I'm not happy with it. If anyone has a better solution I would be very interested to see it!

*** Actual Solution ***

Main file:

```
#include "foo.h"

// Interrupt dispatch forward declaration
```

Jun 2014

1 / 6
Jun 2014

May 2015

```
void myFooDispatch();

// Declare the Foo object and pass in our global dispatch function
Foo myFoo = foo(D0, &myFooDispatch);

void setup() {
}

void loop() {
}

void myFooDispatch() {
  myFoo.doChannelA();
}
```

Jun 2014

1 / 6
Jun 2014

Foo class declaration

```
class Foo {
 public:
  Foo(int channelA, voidFuncPtr interruptDispatch);
  void doChannelA();
  volatile int position = 0;
};
```

May 2015

Foo class implementation

```
Foo::Foo(int channelA, voidFuncPtr interruptDispatch) {
  pinMode(channelA, INPUT);
  attachInterrupt(channelA, interruptDispatch, RISING);
}

void Foo::doChannelA() {
  position++;
}
```

Fair warning: I haven't tried compiling the code listed above, so I may have made some typos and there may be an #include "application.h" needed in foo.h.

Let me know if this doesn't work for you and I'll spend a little more time on it,
David

P.S. What I would actually love most of all is if there were an abstract GpioInterruptInterface class that just provides a pure virtual function: void onInterrupt(int pin). Then the attachInterrupt() function could just take a GpioInterruptInterface pointer as a parameter, the interrupt would call the virtual function on the object, polymorphism does its magic, and no static/global functions are needed at all! But I understand the desire not to force C++ on people who don't want it 😄

SomeFixItDude Jun '14

Thank you @dpursell for the explanation and example. That makes sense and I'll give the code example a go. I am a little disappointed I need to have a static/global function to make it work. Makes me feel like my class is not so stand-alone as I would like it. Defining an interface and virtual function that we could override sounds like the nicer solution. Again thanks for the explanation. 😃

---

paulsoulsby May '15

Hi David,
I was just looking at your code for the ideal solution to the problem, which is to have an attachInterrupt function that takes the class too:

```
attachInterrupt(channelA, &Foo::channelAInterruptDispatch, this, RISING);
```

What would be the source code for this alternative attachInterrupt function? How would you declare **this** in attachInterrupt, as it could be coming from any class?

cheers
Paul

Jun 2014

---

mdma **Particle Senior Embedded Engineer** May '15

The 0.4.0 firmware (available now on the photon, and in a few weeks on the core) supports registering `std::function` instances as the handler so class callbacks are fully supported! 🙂

1 / 6
Jun 2014

SomeFixItDude May '15

That's great! Thanks @mdma.

May 2015