Objetivo del proyecto: construir prototipo para la Medición de nivel de tanque abierto

Water is a precious resource in many parts of the world and many people rely on water tanks to supplement their water supply by storing collected rainwater or water pumped from a well or bore. But how do you measure how full a tank is? Tanks are constructed of opaque material to prevent algae growth and are often kept closed up to prevent mosquito infestation or access by rodents, so it’s inconvenient to physically look inside. And besides, having a way to measure tank depth electronically opens up a world of possibilities, such as automatic control of pumps to fill tanks when they get low or to disable irrigation systems when not enough water is available. The obvious way to measure tank depth is by placing a series of conductive pickups at various heights inside the tank and measure the resistance between them. For example, you could attach ten exposed terminals at equal intervals on a length of material such as PVC tubing, and insert it into a tank vertically to measure the depth in 10 percent increments simply by reading the resistance between the bottom terminal and each of the terminals above it. The downside to this approach, though, is that you’ll need to do a lot of wiring and you’ll also need to read an analog input for every individual terminal. Also keep in mind that most Arduino designs have no more than six analog inputs. There are ways around this, such as using a resistor ladder, but it can still end up being quite time-consuming and still give very poor resolution.

This project works a little differently. It uses a device called a pressure transducer to measure the water pressure at the bottom of the tank, and from that measurement it calculates how full the tank is. Water pressure increases by about 9.8kPa per meter of depth (varying slightly depending on the purity of the water and other factors), so a full tank 1.5 meters tall will have a pressure at the bottom of about 14.7kPa above ambient atmospheric pressure. The “above ambient atmospheric pressure” part is important: it’s not enough to simply measure the pressure at the bottom of the tank using a single-ended “absolute” pressure transducer because varying climate conditions will alter the reading as the air pressure changes. That’s why this project uses a “differential” pressure transducer that has two inlets. By leaving one inlet open to the atmosphere and connecting the other to the bottom of the tank, the transducer will output the difference between the two. This approach provides automatic compensation for varying air pressure and giving a constant reading for constant depth even if the weather changes.

The Arduino then reads the output of the transducer and reports the depth of the tank.

In this project we will use an Ethernet shield so the Arduino can connect to an online datalogging service called Pachube (www.pachube.com) to generate graphs of water depth over time, but you could also have it make decisions based on water depth to control items such as a water pump or irrigation system solenoid. You can see the required parts in Figure 12-1 and the complete schematic in Figure 12-2.

**Parts Required**

1 Arduino Duemilanove, Arduino Pro, Seeeduino, or equivalent

1 Ethernet shield or WiShield

1 Prototyping shield

1 MPX2010DP or MPX2053DP differential pressure transducer (see text)

1 LM324 op-amp

1 14-pin IC socket

4 1K resistors

3 22K resistors

1 1K multiturn variable resistor

1 10K multiturn variable resistor

2 10nF MKT capacitors (typically marked “103”)

1 100nF MKT capacitor (typically marked “104”)

10cm of 4mm tubing (commonly used for aquarium air pumps)

3 4mm internal-diameter cable glands (typically sold as “3 to 6.5mm”)

1 small weatherproof PVC box (we used 64mm 58mm 35mm)

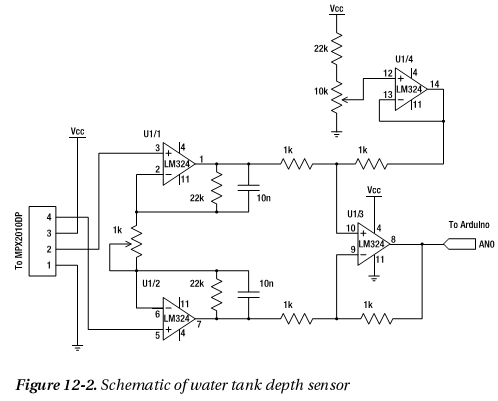
1 medium weatherproof PVC box with O-ring seal (we used 115mm 90mm

55mm)

2 meters flexible four-conductor wire (such as security cable or telephone wire)

Codigo del programa disponible en: [www.practicalarduino.com/projects/water-tank-depth-sensor](http://www.practicalarduino.com/projects/water-tank-depth-sensor).





**Instructions**

If you come from a software background and haven’t done much work with electronics before, this circuit can look a little intimidating, particularly as it deals with analog voltage levels rather than nice, neat, “on or off” digital levels.

The key to understanding the circuit is to temporarily ignore the section in the top right of the schematic shown in Figure 12-2 and just look at the overall symmetry of the central section, starting from the MPX2010DP differential pressure transducer on the left.

Before getting into the details, though, you need to consider the requirements for the pressure transducer. The MPX2010DP is fairly commonly available, but is only rated to 10kPa, equivalent to the pressure from just over a meter of water depth. In testing we’ve used it in tanks up to two meters deep and it’s performed fine, but keep in mind that doing so is pushing the transducer beyond the manufacturer’s specifications, and if you need to measure depth of a larger tank you may need to use a part with a higher pressure rating. For example, at the high end of the scale, the MPX2053DP is a similar part that is rated to 50kPa—equivalent to 5m of water depth. The transducer has two ports protruding from it and will generate a tiny voltage differential between its outputs that is proportional to the difference in pressure between those ports. If the pressure applied to both ports is the same, the sensor will remain in equilibrium and the voltage on both outputs will be about half of the supply voltage, or about 2.5V in our case. The difference between them will, therefore, be zero. If the pressure applied to port 1 is higher than the pressure applied to port 2, the voltage on pin 2 will rise and the voltage on pin 4 will fall. The variation is tiny, though: a pressure differential of 10kPa will cause a voltage differential of only about 12.5mV.

That’s too small to be measured reliably by an analog input on an Arduino, so the rest of the circuit is dedicated to amplifying that voltage differential up to a higher level.

The two transducer outputs are fed into the inputs of “operational amplifiers,” more commonly referred to simply as “op-amps.” Op-amps are perhaps one of the most commonly manufactured nonpassive electronic devices ever, and they work by taking the difference between two inputs and outputting a signal that is an amplification of that difference. As such, we could have just used a single op-amp and fed both outputs from the transducer into it, but that wouldn’t have provided enough gain to raise the signal to a level that the Arduino can read reliably. To overcome this we use a compound amplifier circuit that amplifies each of the outputs individually, driving the high signal further above the 2.5V reference point and the low signal further below 2.5V. The result is that the voltage difference between pins 1 and 7 on the outputs of op-amps 1 and 2 will be greater than the voltage difference between pins 1 and 7 on the outputs of op-amps 1 and 2 will be greater than the voltage difference

between the transducer outputs. The combination of the 22K resistors and 10nF capacitors linking the output of op-amps 1 and 2 back to their inputs provide “negative feedback,” with a rise in output level decreasing the gain to maintain stability. It helps the amplifier maintain a steady state and may look counterintuitive, but this is a very common configuration for op-amps and you’ll see it in many analog circuits. For more explanation of negative feedback op-amp circuits, see the Wikipedia page at en.wikipedia.org/wiki/Operational\_amplifier.The 1K multiturn variable resistor joining the negative op-amp inputs together also controls the gain so that the overall effectiveness of the amplification circuit can be altered to best suit the characteristics of the transducer. Most variable resistors are single-turn, and adjust from 0R to their maximum rating through only about 270 degrees of rotation. That doesn’t provide enough sensitivity when tuning an op-amp circuit, though, so we’ve specified multiturn variableresistors that are commonly available in either 10-turn or 20-turn versions. Multiturn variable resistors have a tiny threaded rod inside to gear down the turns you apply and provide very fine control of their resistance.

The outputs from the op-amps in the center of the schematic shown in Figure 12-2 then pass through a pair of 1K resistors and into the two inputs of op-amp 3, which once again amplifies the difference to drive an even greater variation on its output on pin 8. The result is that tiny variations in voltage across the transducer outputs on the extreme left of the circuit cause much larger voltage variations at the output on the far right where it will be connected to an Arduino analog input. That’s not quite the whole story, though. The section of the circuit we’ve been ignoring up in the top right also comes into play, biasing the input on pin 12 of op-amp 4, which in turn biases the input on pin 10 of op-amp 3. When tuning the circuit, the 10K variable resistor is adjusted until the output of op-amp 4 at pin 14 is about 1V when the pressure difference on the transducer is zero. This provides a base level of 1V to the Arduino input when the tank is empty.

Having set the bias for an empty reading, the 1K variable resistor controlling the gain on the first two op-amps is adjusted until the output to the Arduino is about 3V when port 1 of the transducer is exposed to the pressure at the bottom of a full tank. The circuit, therefore, has an output voltage that swings linearly between about 1V for an empty tank and 3V for a full tank, a variation of 2V.

So why bother with the bias to pull the “empty” value up to 1V and limit the gain to set the full value at 3V? Why not remove the “empty” bias and increase the amplification to provide a wider swing from 0V to 5V on the output and take advantage of the full input range of the Arduino’s analog to digital converters?

The reason is that the performance characteristics of an op-amp typically don’t allow it to provide a full rail-to-rail swing, and even if they could do so the linearity of the output could be compromised toward the edge of the range. Limiting it to a 1V-to-3V swing is well within the performance rating of the LM324 op-amp we’re using, though, and avoids the non-linear regions toward the edges.

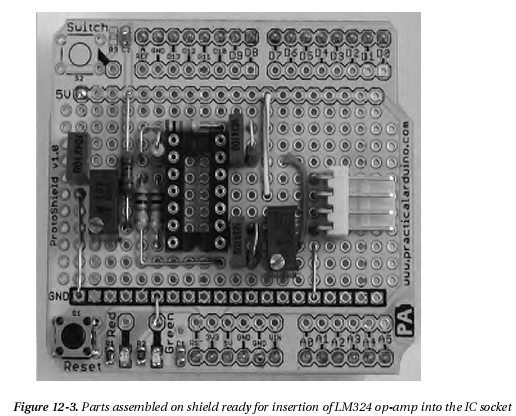
Looking at the schematic in Figure 12-2 you may be a little confused by the fact that it shows four op-amps, while looking at the photos there is only one integrated circuit. That’s because the LM324 is a quad package with four op-amps inside one physical chip, all sharing common power and ground connections. Perfect for this application. What you see in the schematic is the logical representation of the four op-amps contained in the package because as far as the circuit is concerned they are separate devices. The fact that they’re in the same physical package is just extra convenience for us when it comes to assembly.

**Assemble the Shield**

Begin by fitting the 14-pin IC socket, noting the direction of the orientation marker: the little notch in one end. That notch indicates which end of the chip is pin 1, which in the photo shown in Figure 12-3 is in the bottom right corner of the IC socket. The rest of the pins are numbered counterclockwise up the right side to 7 in the top right, 8 in the top left, and down to 14 in the bottom left with the socket oriented as shown in the photo. Note that this is rotated 180 degrees compared to most IC datasheets, which typically show the orientation marker at the top with pin 1 on the top left. In this case, though, it made sense to mount the chip upside down so the supporting parts are nearer the appropriate IC pins.

Next insert the resistors immediately adjacent to the socket, bending the leads underneath the shield to form jumpers to adjacent pads where necessary. The pair of 10nF MKT capacitors (probably marked “103K”) just to the right of the socket go in next, and then the resistors are mounted vertically next to them. Then fit the 10K multiturn variable resistor on the left (marked “103”), and the 100nF MKT capacitor next to it (marked “104K”) along with the jumpers that link them to ground and +5V. The 1K multiturn variable resistor and associated links go on the right.

For convenience when assembling and testing the system we fitted a 4-pin oriented male header for the cable to the pressure transducer. If you prefer, you can solder a lead for the transducer directly to the board.

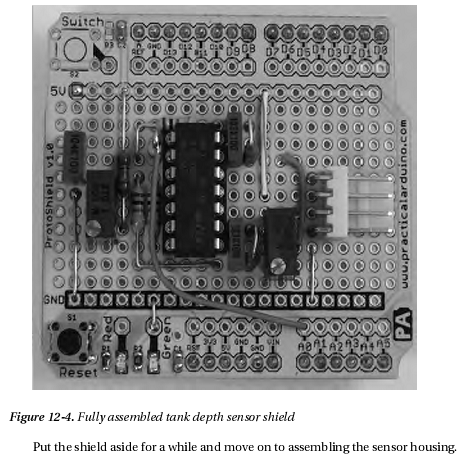


That’s all the hard assembly work done. The only thing left to do on the prototyping shield is to

install a jumper lead that connects pin 8 of the op-amp to analog input 0 on the Arduino. The 1K resistor mounted on the end connected between pins 8 and 9 provides a very convenient mounting point for one end of the jumper lead, which then stretches across the board to analog input 0. Connecting the op-amp output to the analog input with a jumper lead like this makes it easy later if you want to switch to a different input so you can stack multiple shields.

Finally, insert the LM324 op-amp into the socket. Inserting ICs can be tricky at first because the legs always spread out wider than the holes in the socket. You can buy special tools for inserting ICs into sockets but they’re really not necessary on a small IC like an LM234: it’s just as easy to bend the pins in a little until they align with the holes in the socket. Try putting the IC down on your workbench on one edge, grip it on the ends, and very carefully push down to bend the pins in. Turn the IC over and do the same to the other side so it will fit into the socket neatly. Be cautious in applying pressure to the pins, but don’t be afraid. The worst that can happen is you bend them too far and have to bend them back.

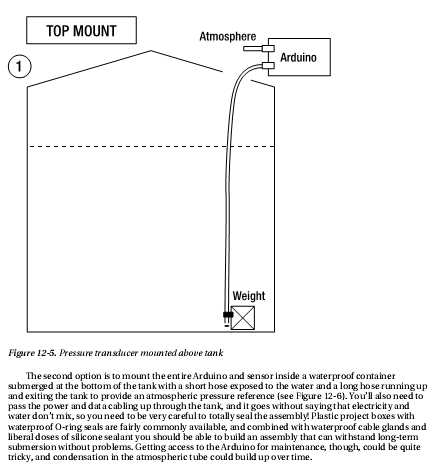
You’ll also notice a short link near the bottom left of the board to connect one of the LEDs provided on this particular prototyping shield to ground. The other end of the LED is connected to +5V via a 1K current-limiting resistor. That LED is included just to show when the shield is powered up and is entirely optional, so it doesn’t appear in the circuit diagram.

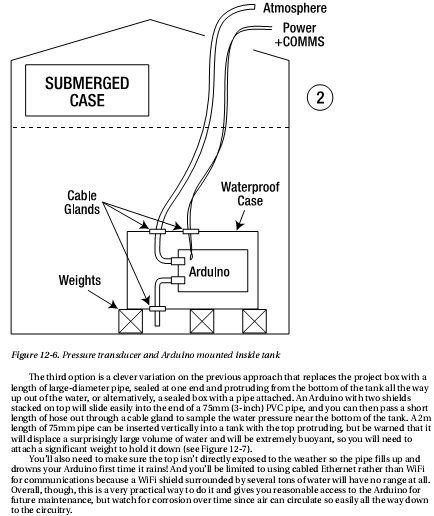


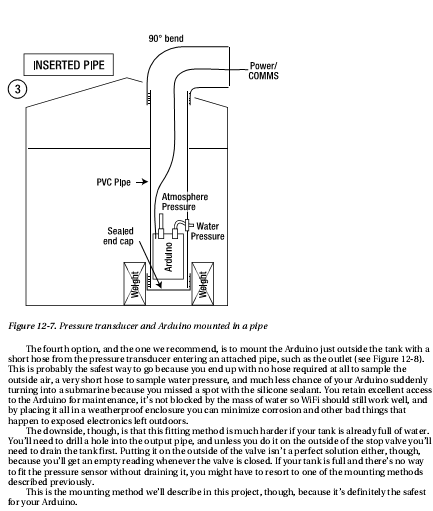
**Determine Sensor Installation Method**

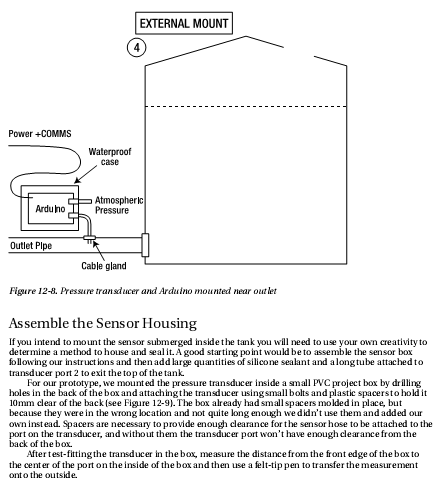
Although the basic circuit will be the same regardless of how you mount it, this project has several options regarding how you expose the transducer to the pressure at the bottom of the tank while also exposing it to atmospheric pressure outside the tank. Before starting construction of the sensor assembly you need to think about how it will be connected.

There are four basic options for installation. The first, and most obvious, is to fit the Arduino and shield into a weatherproof case mounted just above the tank with one transducer hose exposed to the atmosphere and the other running down inside the tank with a weight attached to hold it on the bottom (see Figure 12-5). This can work in the short term but can cause problems over time due to diffusion of air in the tube into the water in the tank. Eventually, the water level will rise up the hose with the result that your Arduino will be reading the pressure from the height of the top of the water in the hose, not the pressure at the bottom of the hose. If left in long enough, a full tank will appear to be slowly draining until you get an “empty” reading. Of course, you can periodically pull the hose out of the tank and drain it to restore accuracy, but it’s not really a good long-term solution.











You can see in Figure 12-9 that the transducer is mounted upside down with the part number

marking hidden on the other side of the case. This is to allow port 1 (the port opposite pin 4, which is on the top right in the picture) to be aligned vertically near the center of the box and provide enough clearance for the cable gland. Your physical mounting arrangement may vary, of course, so you’ll need to plan how you will mount the transducer while retaining sufficient clearance for connections to it.

Measure sideways from one edge of the box and once again transfer that measurement to the outside, then remove the transducer and drill a small pilot hole through the box at the ntersection of the markings. Drill a hole for a second cable gland on one of the other sides for the cable to the Arduino, keeping in mind the location in which the sensor box will be mounted. The cable can enter from the top, side, bottom, or even back of the box, whichever is most convenient.

Also drill a small hole of perhaps 2 or 3mm in the bottom of the box, well clear of the area that will be covered by the cable gland nut so that the inside air pressure will always be the same as atmospheric pressure. Alternatively, you could drill a hole just large enough for the tubing and fit a very short piece to port 2 on the transducer so that it just protrudes outside the box, keeping the inside of the box isolated from the atmosphere.

Switch to a drill bit large enough to allow the cable gland to be mounted (we used a 13mm spade bit) and enlarge the holes for the tubing and the cable. Use a hobby knife to clean up the edges of the large holes so there are no burrs to obstruct the cable glands or keep them from sitting flat against the box (see Figure 12-10).



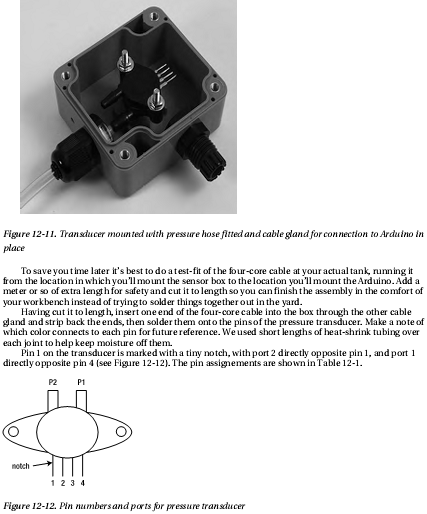
The purpose of the cable glands is not to provide a watertight seal, because the box needs to be open to the atmosphere anyway to sample ambient atmospheric pressure. Rather, the cable glands provide physical support for the tubing and cable and protect the transducer port from mechanical strain if the tubing or box are moved.

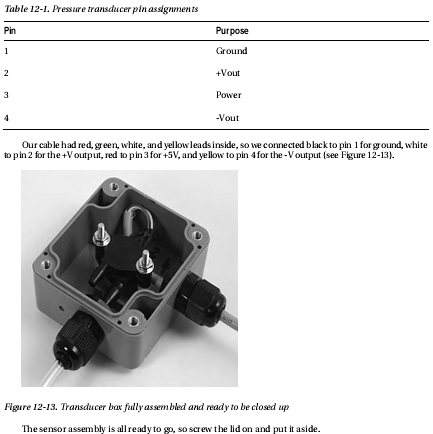
If you don’t have any cable glands available you can make do without them, but you will need to make sure the tubing has good mechanical support as it enters the box. You could drill the hole out to the same size as the outside diameter of the tubing so that it fits through but is a snug fit. After everything is assembled and tested, you could then apply silicone sealant to the tube and cable to give them some mechanical support.

Before mounting the transducer back in the box it’s a good idea to put a drop of mineral oil (commonly sold as baby oil) into each port. The mineral oil will help protect the transducer from water.

Fit one end of the tubing onto transducer port 1 while it’s still out of the box. The tubing should be a very tight fit onto the transducer and you might have quite a bit of trouble getting it on. A handy trick is to boil some water and pour it into a mug, then hold the end of the tubing in the hot water for a minute to soften it up. Slide it onto the transducer while it’s still hot and it should go on much more easily, then when it cools down it will shrink and form a tight seal. Just remember that if the tube isn’t sealed properly onto the transducer port the water pressure from the tank may cause it to spray out inside the box, quickly flooding your expensive pressure transducer.

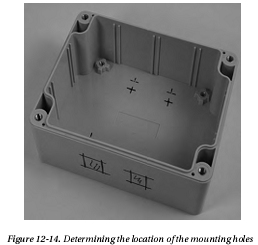
Slide the tubing through the cable gland mounted in the box until the transducer is in the correct place and is aligned with the spacers. Bolt it in again, and tighten up the cable gland around the tubing enough to give it good support without squashing it closed (see Figure 12-11).





**Assemble the Arduino Housing**

Mounting the Arduino along with the WiShield and the tank depth shield in a weatherproof box follows pretty much the same process as building the sensor assembly. Sit the Arduino in the bottom of the box and mark the location of the mounting holes. In our prototype we wanted to provide external access to the USB and power connections with everything mounted in place, so allowance had to be made for the distance the USB socket protrudes from the board. We marked the position of the mounting holes with the Arduino pushed up hard against the bottom of the box, then measured the distance that the USB socket protruded and offset all the markings by that same distance toward the bottom. This way, when the USB and power holes are cut, the Arduino will slide into them and the mounting holes will be in the correct location (see Figure 12-14).



Cutting square holes is always awkward. We started by drilling a small hole in each corner, then drilled holes down each edge. The major section of the center of each hole was then clipped out with wire cutters before the edges were trimmed straight with a hobby knife.

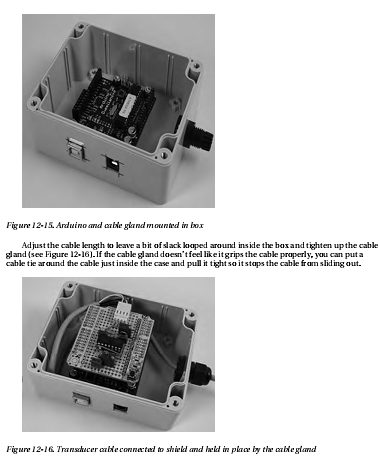
Also drill a hole for a cable gland, once again keeping in mind where the cable will be mounted. We fitted the cable so it exits on the right-hand side of the box.

Fit M3 bolts up through the holes in the back of the box and sit 10mm spacers on top of them, then slide the Arduino down into the holes in the bottom and onto the bolts. This might take a bit of experimentation, but it shouldn’t be too difficult if you let the bolts slide back down a little bit while you get the Arduino in position.

Put insulating washers over the bolts on top of the Arduino, followed by nuts, then tighten the bolts to hold everything firmly in place (see Figure 12-15).

With the Arduino mounted in the box plug the WiShield on top. One very neat thing about the WiShield is that it ships with long-lead female headers and all the parts on the board are kept low to make it stackable, so for this project it’s perfect: after plugging the WiShield into your Arduino, just plug the prototyping shield on top to create a triple-decker assembly with the connector for the sensor cable easily accessible.

Slide the four-core sensor cable through the cable gland, strip back the insulation from the end of the cable and “tin” the ends with solder. Either solder the cable directly to the prototyping shield or, if you fitted a 4-pin female header to the shield, solder it to a 4-pin male header and plug it in.



That’s the hardware assembly all done. Time to start playing with the software.

Install the WiShield Library and Driver The WiShield supports 802.11b at 1 and 2Mbps (the Arduino couldn’t keep up with anything faster anyway!) and even supports WEP and WPA/WPA2, in case you want to keep your top-secret water tank depth data secure from prying wardrivers. It communicates with the Arduino using SPI, which ties up digital I/O lines 10, 11, 12, and 13, along with line 2 for an interrupt connection and pin 9 for the shield’s status LED, so it’s important to keep those free in your project. Because we’re only using one of the analog inputs, that’s not a problem for us in this project.

The WiShield needs two software components to function correctly: the WiShield library from AsyncLabs, and the driver code from ZeroG Wireless. Unfortunately, the driver from ZeroG Wireless hasn’t been released under a FOSS (Free / Open Source Software) license and therefore can’t be distributed along with the library, so you need to install the library and then download and install the driver separately. More information is available from the AsyncLabs web site at www.asynclabs.com, or you can follow along the steps here.

Start by downloading the WiShield library using the WiShield Library link from the project page on the Practical Arduino web site. Extract it on your local computer and rename the directory to WiShield if necessary, and move it into your sketchbook/libraries directory so the Arduino environment can see it. Next, go to the driver page using the WiShield Driver link from the project page. There you’ll see the terms under which the driver has been made available. Click the WiShield Driver Download link at the bottom of the page to get an archive called wishield-g2100-driver.zip. Extract the ZIP file, and move the g2100.c and g2100.h source files into your sketchbook/libraries/WiShield/ directory.

The WiShield library supports several different modes. The default mode is the APP\_WEBSERVER mode, which should run on most Arduinos but has limitations such as not being able to run as both a client and server simultaneously. An alternative mode called APP\_WISERVER will run on any Arduino with an ATMega328P or better CPU, which includes the Duemilanove and most current third-party equivalents such as the Arduino Pro. Any reasonably modern Arduino should be able to run in APP\_WISERVER mode so that’s what we’ll be using.

To switch the library into APP\_WISERVER mode, open up the file sketchbook/libraries/WiShield/apps-conf.h and go to about line 39. Comment out the existing APP\_WEBSERVER entry and uncomment the APP\_WISERVER entry a few lines below it so that it looks like the following:



With everything in place and those changes made to the library, you can now open up the Arduino IDE and have access to the WiShield library and example code. Note, though, that there are examples provided for several different modes: some examples only work in APP\_WEBSERVER mode and some only work in APP\_WISERVER mode. The ones that work in APP\_WISERVER mode are all prefixed with “Simple.”

To test that the WiShield is working correctly, open up the example at File Examples WiShield SimpleServer, adjust the network settings and WiFi encryption settings to suit your network, compile it, and upload it to your Arduino. If you’re running on an open network it should associate pretty much immediately and the red status LED will illuminate to show it’s joined the wireless network. If you’re running WPA the process may take 30 seconds or so while the WiShield negotiates with your access point, but eventually the red LED will come to life to show that your Arduino is now on your network.

Open up a web browser, go to the IP address you set in the program, and you should see a “Hello World!” message from your Arduino.

The SimpleServer example implements logging via the serial connection, so if you’re curious to see what the Arduino is doing you can open the serial monitor in the IDE and set the baud rate to 57600bps.

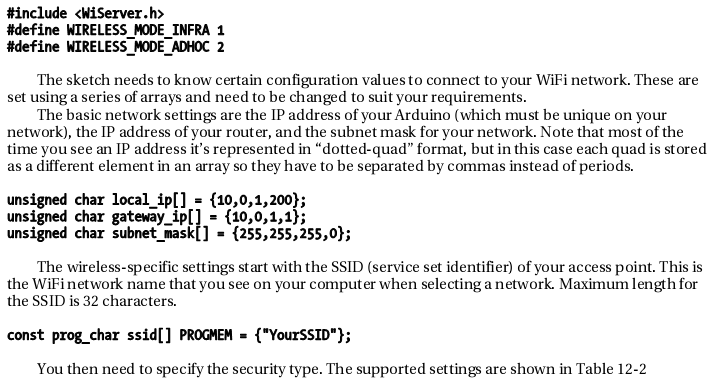
Be warned, though, that with most modern Arduinos the act of opening the serial connection forces it to reset, and every time the Arduino boots up it may take another 30 seconds or so to reassociate with your wireless network. Lots of patience is required if you keep opening and closing the serial monitor.

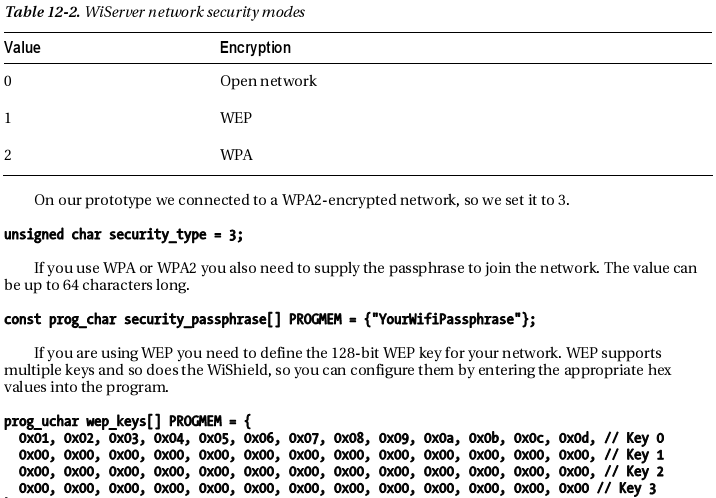
Congratulations! Your Arduino is now connected to your WiFi network as a web server.

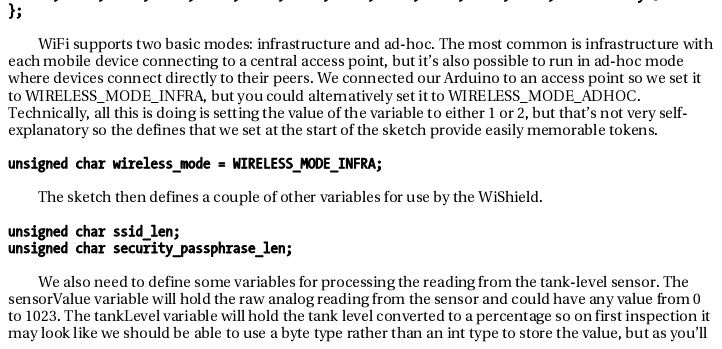
**Load the Tank-Level Sketch**

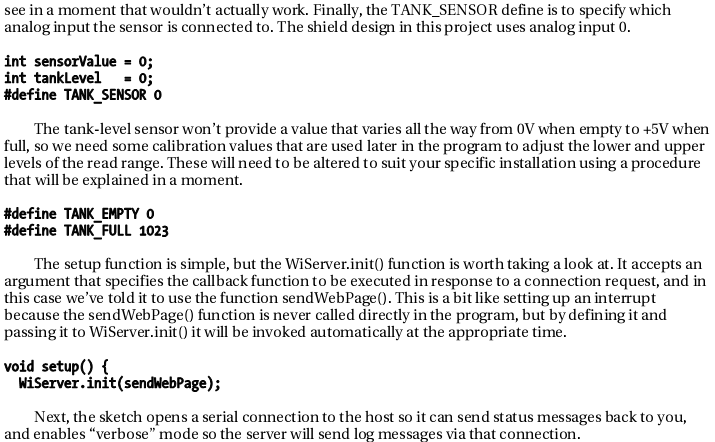
The tank-level reporting sketch is based on the SimpleServer example code with a simple addition to read an analog input and include the value in the web page sent back to the browser.

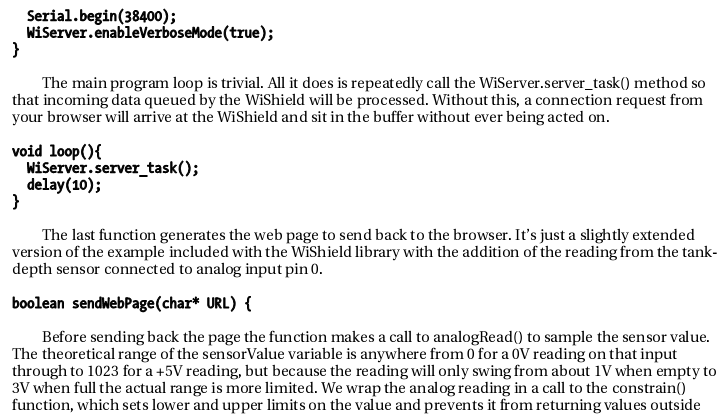
The sketch starts by including the WiShield library. Because we’re using it in APP\_WISERVER mode we include the WiServer.h header file instead of the WiShield.h header referenced in some of the other included examples. The sketch then defines a couple of tokens to make the code further down a bit more readable.

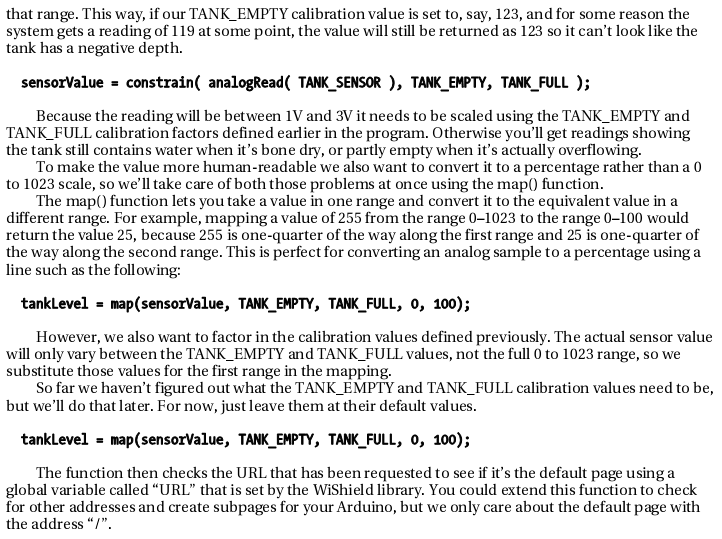


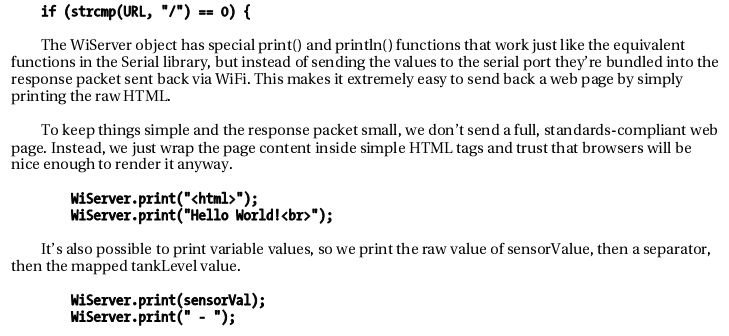


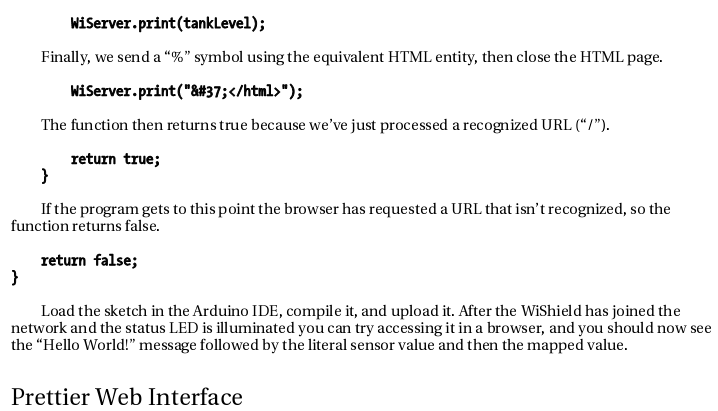


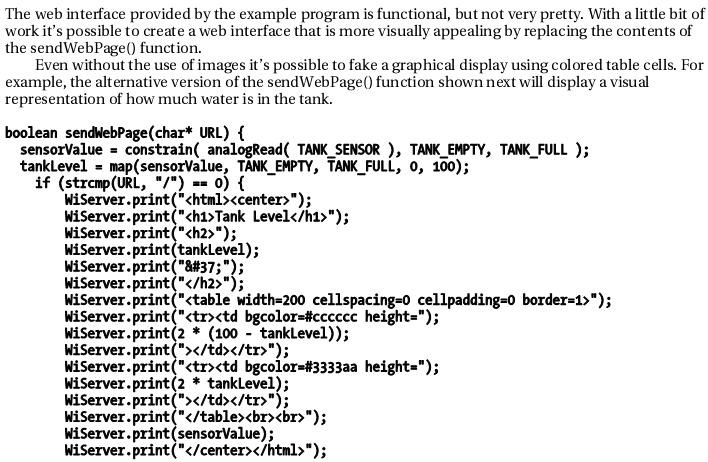


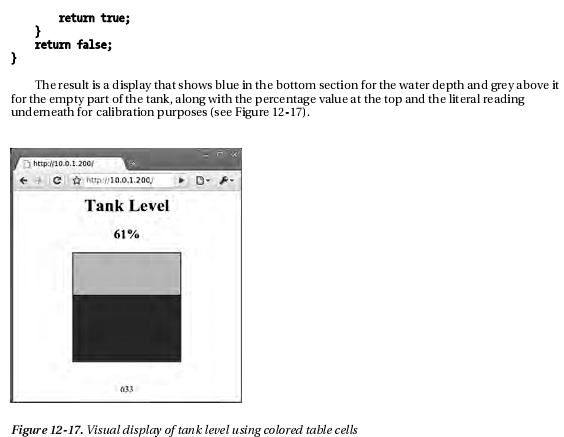


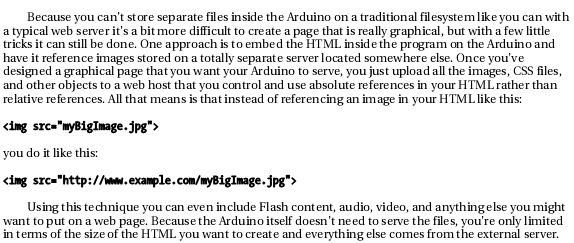


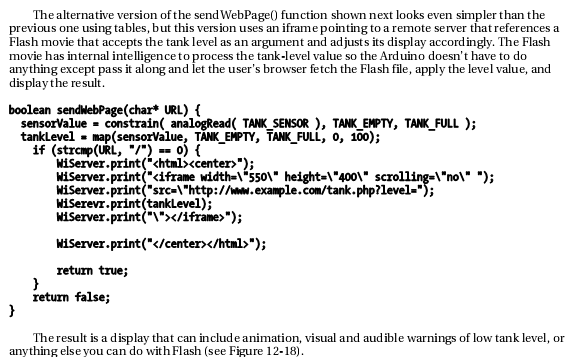


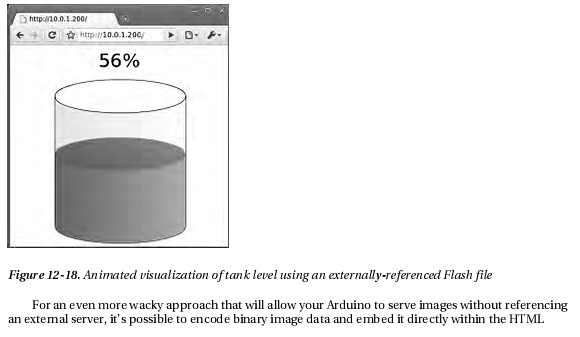


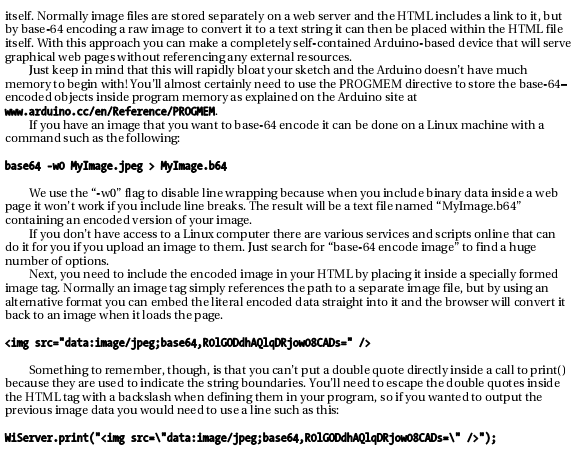


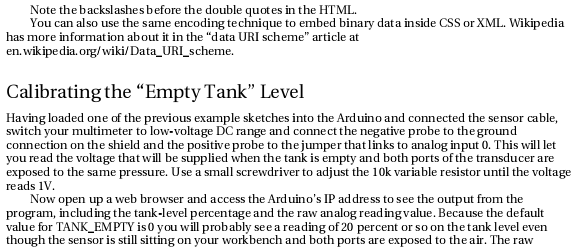


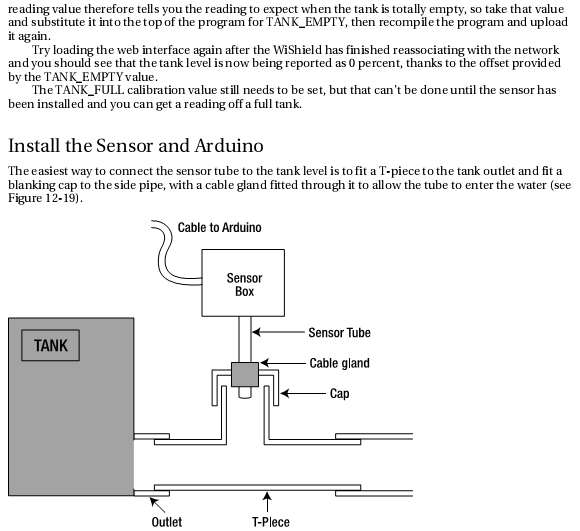












Turn off the stop-valve on the tank outlet and disconnect the pipe that attaches to it, and install a T-piece between the two.

Then drill a hole through a blanking cap for a cable gland and screw the gland in place very firmly. Screw the blanking cap onto the T-piece, using plumbers teflon tape if necessary to get a perfect seal.

Due to the pressure that will be applied the cable gland will need to be sealed onto the sensor tubing very tightly. Because the tube will tend to be squashed by the cable gland when trying to get a really tight seal it’s a good idea to insert a short length of metal pipe into the plastic tube first. A short section cut from an old telescopic antenna is perfect: cut out about 25mm from a section that fits snugly inside the tube and slide it in, then slide the tube into the cable gland. You can now tighten up the cable gland very tightly without the tubing being squashed closed, but air can still pass though the hollow metal tube to apply pressure to the transducer port.Rather than leaving the sensor box dangling loose on the top of the tube it’s best to give it some form of mechanical mounting. A good solution is to attach the box to a piece of timber, hammered into the ground beside the tank outlet.

When everything is nice and tight, open the stop-valve again and watch carefully for leaks. If you’ve done a good job everything should stay nice and dry and the water should stay in the pipe where it belongs.

Mount the Arduino box in the location you previously selected and attach the sensor cable securely using cable ties or similar to keep it neatly out of the way.

**Calibrating the “Full Tank” Level**

To determine the TANK\_FULL value you need the tank to actually be full and the sensor connected as previously described. If your tank isn’t full at the moment you may need to fudge this value a bit based on an estimate of how full it currently is, and then adjust it later when the tank really is full.

With the tank stop-valve open so that the sensor is exposed to the full tank pressure, attach the negative probe of your multimeter to the ground connection on the shield and the positive probe to the jumper going to Arduino analog input 0. You’ll get a reading somewhat higher than 1V, so using a small screwdriver adjust the 1K variable resistor until it reads 3V. This adjusts the gain on the amplifier for the TANK\_FULL value.

Now use your computer to load the page again with the sensor exposed to the pressure from a full tank, and you’ll see a tank-level reading probably somewhere around 60 percent and the literal sensor value below it. Take that literal sensor value and set it as the TANK\_FULL value at the top of the program. Then recompile the program with those new values, upload it to your Arduino, and you’re ready to go. The system should now report 0 percent when the tank is empty, 100 percent when it’s full, and appropriate values in between.

