FEATURE

Program the world: 12 technologies you need to know

Stuffing bits in databases is boring. Why not program everything around you?

By Peter Wayner | Follow
InfoWorld | Feb 16, 2016 12:10 PM PT

Every startup talks about changing the world, but most aren't talking about the world itself or physical things in it. Most simply want to swap data packets and place entries in databases -- potentially important bags of bits, but bags of bits nonetheless. The world, though, is made up of atoms.

The barrier between bits and atoms is disappearing, with programmers no longer confined to the virtual realm, in part thanks to the Internet of things becoming more real. Now we can do more than write ones and zeros to a disk: We can actually write code that tells a machine how to extrude, cut, bend, or morph atoms. Now our software can turn on lights, change the look of a room, steer a car, move a wall, or more.

Today, many of the new markets and opportunities for developers live in the real world. Rapidly developing domains such as autonomous cars, smart homes, intelligent office spaces, and mass customization require programmers to be savvy about how changes in data structures can lead to changes in objects. If the term "object-oriented programming" weren't already taken, it would be perfect.

These jobs require new languages or, if they're not officially new languages, new protocols that work with older languages. Changing the world means learning how these languages and protocols work and how to deploy them. If you're looking to *really* change the world, here is a partial list of languages and protocols to master. Once you start flipping bits that change the world, it's hard to go back to mere databases.

Basic

One of the classic languages that drove the early microcomputer revolution lives on in the minds of some simple hardware controllers. The folks who make the ESP8266 controller board use the language because, as they say, it is “a simple but powerful language that lets you do amazing things without needing a degree in computer science.”

All the classic structure is there, including that old bugaboo: goto. But there’s also newer commands for fetching Web pages or sending email. Most of your time will be spent polling pins on the interface, though, to gather data to be passed on to the Internet.

For more information, see the ESP8266 community or the website devoted to the language.

X10

X10 is the original tool that allowed programmers to break out of their virtual world and touch the real one. It’s never been a complicated or elaborate protocol, perhaps because it dates back to 1975, a year before Steve Wozniak and Steve Jobs released the Apple I. Despite its age, it remains popular because so many low-cost devices support it.

The protocol has only a few major messages. You can send a few bits to the switches and ask them to turn themselves on, off, a bit brighter, or a bit dimmer. That’s about it. There are a few more options to poll remote switches and embed a bit more data, but most of the work seems to be switching things on and off. These packets travel over the 120V power lines of a house and must be created by a separate device such as the FireCracker made by X10.

A number of software projects such as Flipit and Bottle Rocket simplify the process of working with the FireCracker. Or you can work with the original master, X10, the company that proclaims it has been ordering around home gadgets since 1978.

unched a protocol meant to replace the X10 protocol by offering bigger messages and better transmission. Many reported that X10 signals were often lost in larger houses or those with more complex wiring. Others complained the signals were affected by noise. Thus, Insteon added a mechanism where each node or switch also acts as a repeater, ensuring that the signals will travel farther and reach deeper into the corners of the electrical wiring. It’s a clever trick that spreads the signals everywhere.

The protocol is also much richer, complex, and redundant. Instead of only two bits of commands, the Insteon packets stretch out to be 10 bytes with two entire bytes devoted to commands. If you're feeling like a big spender of bandwidth, you can mix an additional 14 bytes of data into the packets going to your devices. The major commands are still focused on turning things on and off, but there are some options for polling sensors and creating a smart thermostat.

Insteon's protocol is used by a number of smart hubs or home automation tools such as Amazon's Echo or Logitech's Harmony. Several open source tools such as the Linux Home Automation project and OpenRemote will integrate with the power line controllers used to send the packets.

Zigbee, Z-Wave, and more

X10 and Insteon merely scratch the surface. There are many more approaches to sending signals to devices, sometimes over power lines, sometimes wirelessly. Both Zigbee and Z-Wave are low-powered, wireless standards for communication between devices, especially the kind of low-powered, embedded sensors and processors that might be spread around a business or a home.

Zigbee, for example, recently announced that supermarkets are experimenting with temperature sensors for monitoring produce sections in hopes of ensuring that fruits and vegetables don't spoil. One section of Z-Wave's website describes devices delivering "dignity at home" via a wireless button that makes it possible for older or weaker people to call for help if they should happen to fall. There are hundreds of applications like these that need to send only a few bits of data at a time.

Zigbee and Z-Wave are not the only standards. The growing list includes Panstamp, AMX, KNX, Lutron, and more. Some target niches; AMX, for instance, focuses on the audio-visual devices that enhance conference rooms. Panstamp is aimed at small, wireless controllers that are deeply embedded in the world. All use their own format but might interact with a bit of extra programming. The cornucopia of competing standards may be confusing for programmers, but it's better than no standards at all.

XBMC, Freebox, and more

It's a bit of a stretch to say that someone sitting on a sofa watching the idiot box is changing the world, but there's no doubt that digital images, videos, and audio are breaking out of their traditional roles as entertainment. A number of the protocols and frameworks such as XBMC, Freebox, and VLC were traditionally designed to spoon-feed video to a lounging couch potato, but they have uses throughout the house or building.

These audio-visual devices may not seem like they're doing much to change the world because they're largely juggling digital content files, but they're becoming more concrete as we cover the world with flat-panel displays. The buildings in Times Square, for instance, change their appearance based on which images are displayed on the screens. More screens means that digital content isn't only something to watch on your phone. It's a way to repaint a building or redesign a room.

PostScript

Many people don't understand the complexity of the data structure used to store data in PDF files or send a page of text to a printer. The data inside a PostScript file isn't merely a list of letters but a program for moving a pen around the page and drawing letters, lines, numbers, and shapes. The language has primitives for moving the pen along straight lines or Bézier curves and then filling in the shapes. The fonts aren't only bitmaps but complex collections of curves that are easy to scale or position with subpixel accuracy.

The language itself is a '70s-era artifact, with a stack-based syntax that saves on parentheses. Anyone who learned to use an HP calculator will feel right at home. The language is Turing complete and people have written PostScript code to calculate complex fractals and other weird things -- oh, and viruses.

Today, the language often takes a backseat to SVG because this XML variant is widely supported on the Web. But the structure underneath is similar and conversion is straightforward. Both of these languages can be turned into the codes used to drive laser cutters and milling machines through a variety of packages like PSToEdit.

OBD-II

It's been years since a car was simply a gas tank, some pistons, and a crankshaft with gears for redirecting the force of the explosions to the rear wheels. Today's cars are complex networks of multiple computers that happen to have four wheels. The OBD-II standard is the way that mere mortals can interact with the car and find out what is going on.

Much of the codes that travel between the computer and the car are purely informational. You can, for example, send a few bytes to the port underneath the steering wheel and it will return the current speed. There are similar codes for RPMs, engine efficiency, and dozens of other numbers. Many basic apps such as Torque will poll the OBD-II port to track your car.

The apps are very popular with amateur racers and gearheads, but they can be useful for others. The ArduinoOBD library is one of several good options for connecting your computer to your car.

G

Computer Numerical Control, aka CNC, of milling machines began in the 1950s and engineers quickly developed G, a language for specifying how the cutting tools should move. The process is a bit counterintuitive at first because the code controls what is removed from a solid instead what is added, but once you invert how you think about the end result, you can start imagining how moving a sharp, spinning bit will cut what you want.

Much of the coding involves choosing your coordinate system and arranging for the cutting bit to move to particular places. The machines can often "interpolate" and compute the midpoints along a straight line or a circle, making simpler shapes relatively easy to create. Complex shapes, though, take a bit of planning.

The nature of G has changed markedly over the years as manufacturers add their local enhancements. Many now use more modern macros and object-oriented layers that are compiled down to the raw G code sent to the machines.

Today, a variant of G will drive many 3D printers. The codes aren't exactly the same, but the core of language is.

STL

The standard format for describing 3D objects is the lingua franca for the 3D printing world. Online stores sell virtual objects by delivering STL files that you can either edit or send straight to the 3D printer.

The language itself is quite basic. Most of the file is composed of three-dimensional coordinates for the corners of the triangles that form the facets that cover the surface of an object. Although the format seems to support more complex polygons, the files traditionally hold triangles. The file format itself does not require the triangles to fully cover a solid or define every part of the surface, although that is a requirement for building a 3D part.

The STL files can either contain an ASCII representation of the points or a binary version, but most seem to exchange the binary versions because they are more compact.

Slicer

STL files do not contain enough information to drive a 3D printer, so the triangles must be converted into a list of instructions for moving the 3D printer head and turning the extruder on and off. The printers generally use GCode, aka Slicer, which is similar to the GCode fed to CNC milling machines. The biggest difference is that the code is additive, building up the part from scratch, whereas

CNC machines are subtractive, carving the part out of a solid starting block. Many of the instructions for positioning the printer head are the same, but there are new ones for turning on the extruder.

A number of Slicer programs exist, both open source and proprietary. Some include sophisticated IDEs for editing the object before printing it. The KISSlicer, for instance, comes in a free version that works with single-head printers and a pro version that supports more sophisticated printers with multiple heads.

Python

The language is a favorite in biology labs, in the social sciences, and throughout the Raspberry Pi community where it is the "official language." This is overselling it a bit because Raspberry Pi boards usually boot directly to Linux, and much of the Linux code base can run on the boards, but only if it compiles directly to the ARM (v6 or 7) architecture.

Still, Python remains a favorite with good reason. It offers a high-level language with a clean syntax and fairly forgiving rules for passing data around. The programmers don't need to get lost in deep abstractions such as closures or memorize complex rules about pointers. They simply write a set of instructions for processing numbers and using these values to tell the Raspberry Pi what to do.

The Raspberry Pi community offers extensive documentation on learning Python from scratch.

Processing

The world of robotics and real things is filled with Arduino boards that speak Arduino's language, a subset of C and C++. Many programmers, though, want something a bit simpler, which may be why a language called Processing has caught on. It's a simplified subset of Java that leaves out many of the gnarly details of classes while adding a number of standard methods that help draw results and flip switches. Java programmers may find it fascinating to look at how Processing is built on the old AWT Applet and Frame, but the rest of the world will simply want to write some loops and change the code.