Application development for the OLPC laptop

Program an activity for the XO-1 using Python

Skill Level: Intermediate

M. Tim Jones (mtj@mtjones.com)
Consultant Engineer
Emulex Corp.

18 Dec 2007

The XO laptop (of the One-Laptop-Per-Child initiative) is an inexpensive laptop project intended to help educate children around the world. The laptop includes many innovations, such as a novel, inexpensive, and durable hardware design and the use of GNU/Linux® as the underlying operating system. The XO also includes an application environment written in Python with a human interface called *Sugar,* accessible to everyone (including kids). Explore the Sugar APIs and learn how to develop and debug a graphical activity in Sugar using Python.

Section 1. Before you start

This tutorial describes an emulation of the One Child Per Laptop (OLPC) XO laptop using QEMU. This means that you'll emulate a complete XO laptop on a PC (running a Linux® or Microsoft® Windows® operating system) for the purposes of developing a Python *activity* —that is, a program in the OLPC Sugar interface, which is based on the Python programming language. At the heart of the XO laptop, like most of the cool devices out there, is the Linux operating system.

About this tutorial

This tutorial shows how to develop Python activities for the XO laptop. From this perspective, you learn about Sugar (the XO user interface, or UI) and the details

behind activity development. You also learn about Python programming, Sugar application program interfaces (APIs) for Python, and platform emulation with QEMU.

Objectives

In this tutorial, you learn about the XO laptop and how to write a Python activity using the Sugar UI. Along the way, you learn more about the XO laptop, its architecture, internals, and use.

Prerequisites

This tutorial is written for Linux developers who want to learn more about the XO laptop and the Sugar UI. It assumes a familiarity with the Linux shell and a basic understanding of the Python language.

System requirements

This tutorial provides a hands-on approach to learning about the XO laptop and Sugar activities. It assumes a Linux computer with at least 1.1GB of free space.

Section 2. OLPC and the XO laptop

For those new to the XO laptop and the OLPC, this section gives a brief overview.

What's all this, then?

The XO-1 is a laptop computer designed to be inexpensively manufactured and distributed to children in developing countries around the world. The goal of this laptop is to bring technology and access to information to children who would otherwise not have this ability. The laptop includes wireless networking, allowing children to collaborate locally with each other on projects.

The laptop was first proposed by faculty at the Massachusetts Institute of Technology (MIT) Media Lab under the non-profit organization, One Laptop Per Child. The organization includes such luminaries as Nicholas Negroponte and, in the early days, Seymour Papert (a pioneer in the field of artificial intelligence), whose

ideas and research are instrumental to the project.

The project has not been without criticism. As a technology, the laptop is novel, but some wonder whether the money spent on the laptop in some developing countries would be better spent elsewhere. For example, some of the target countries for the XO-1 lack the basic elements of education (such as schools or libraries). See Resources for more details.

The XO-1 laptop

The design of the XO-1 laptop is interesting, because it includes two competing requirements. First, the XO-1 laptop must be inexpensive (selling for USD140) but also feature rich. The design includes a 433MHz AMD Geode processor, 128MB of RAM, 1GB of NAND flash (instead of a hard disk), a 7 1/2-inch dual-mode LCD, wireless networking, and even a video camera. The laptop is also designed to be operated in non-traditional environments and is therefore very durable and rugged (see Figure 1).

Figure 1. The XO-1 laptop

Wireless networking is based on an 802.11b/g and 802.11s chip, which supports both standard networking and mesh networking. This allows laptops to autonomously route packets between themselves in a peer-to-peer network (without the need for a separate router). The mesh networking works even if the CPU is powered off.

Software

While the ideas behind the XO-1 laptop are intriguing, and its physical and hardware design make it novel, what's most interesting is the software. The XO-1 laptop is powered by GNU/Linux (a minimal version based on the Fedora project). The basic input/output system (BIOS) is the Open Firmware, which is written in a variant of Forth. The graphical user interface (GUI), called *Sugar,* is based on the X windowing

system and written in Python. This is by far the most interesting element, because it makes the XO-1 an immediate development environment. (Python is an interpreted language.) No compilers are necessary, and anyone can develop Sugar *activities* (that is, applications in the Sugar GUI). In other words, all software on the XO-1 laptop is open source and therefore free and modifiable.

This use of Python is interesting, because as the core of the XO-1 laptop, it exposes Python to millions of children around the world. Python was created as a language to learn programming, so it was an ideal decision and has the potential to create a new population of Python enthusiasts.

Section 3. XO laptop emulation

This section gives instructions for downloading and installing the QEMU platform emulator and the XO-1 laptop image.

Introduction to QEMU

QEMU is a platform virtualization application that allows you to emulate an entire computer (processors and associated peripherals) for purposes of virtualizing another operating system on it. With this, you can emulate Linux on Windows, Windows on Linux, or any other operating system on another operating system that runs QEMU. You can even emulate an operating system for a different architecture than the host. For example, you can emulate an Arm Linux system on an x86 host.

A system emulation is made up of the platform emulator (QEMU), an optional accelerator (KQEMU), and a root image (a file in the host operating system) that contains both the kernel and the root file system. For more details on QEMU and its internal operation, see Resources.

Download and install QEMU

Installing QEMU and the accelerator is as simple as you would expect with GNU/Linux. Listing 1 provides the QEMU download, build, and installation instructions.

Listing 1. Downloading, building, and installing QEMU

```
$ wget
```

```
http://fabrice.bellard.free.fr/qemu/qemu-0.9.0.tar.gz
$ tar xfvz qemu-0.9.0.tar.gz
$ cd qemu-0.9.0
$ ./configure
$ make
$ make install
$
```

The QEMU accelerator is an optional step, but I recommend it because it results in greater performance. Listing 2 gives the QEMU accelerator download, build, and installation instructions.

Listing 2. Downloading, building, and installing the QEMU accelerator

```
$ wget
http://fabrice.bellard.free.fr/qemu/kqemu-1.3.0pre11.tar.gz
$ tar xvfz kqemu-1.3.0pre11.tar.gz
$ cd kqemu-1.3.0pre11
$ ./configure
$ make
$ make install
$ insmod kqemu.ko
```

Note: At the end of Listing 2, the QEMU accelerator kernel module is installed.

Get the XO kernel and file system images

The final step in preparation is to download and prepare the QEMU image for the XO-1 laptop. Listing 3 shows how to retrieve this image and extract it prior to use. The image is quite large (183MB), so the download time can be long. Also, the downloaded image expands to almost 1GB in size, so make sure that you have sufficient space available.

Listing 3. Download and extract the OLPC image

```
$ wget
http://olpc.download.redhat.com/olpc/streams/development/LATEST-STABLE-BUILD/
devel_ext3/olpc-redhat-stream-development-devel_ext3.img.bz2
$ bunzip2
olpc-redhat-stream-development-devel_ext3.img.bz2
```

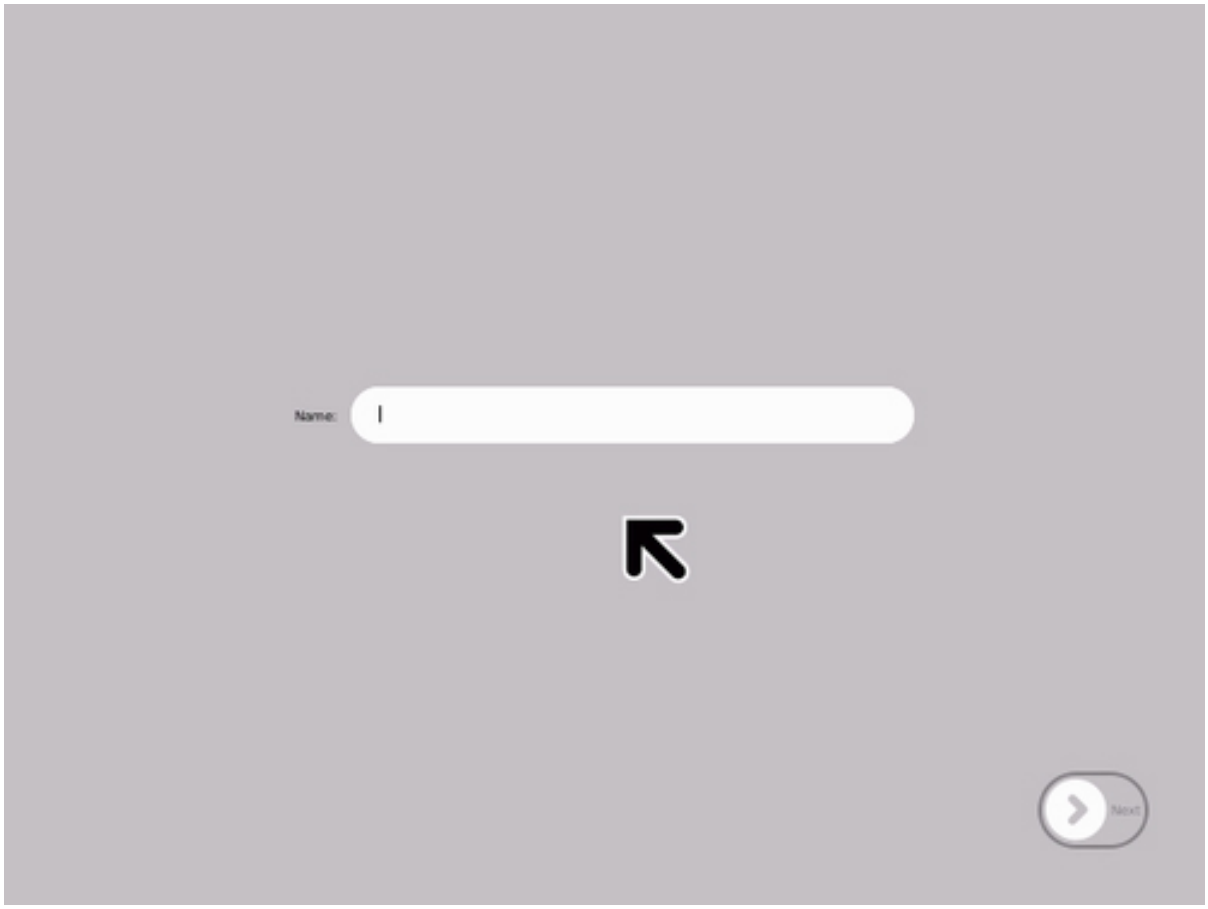
Start the XO-1 laptop

With the OLPC image downloaded and extracted, it's time for a test run. Start QEMU with two options that specify the amount of memory to make available (256MB) and the hard disk to use for the emulation (the image you downloaded above):

```
$ qemu -m 256 -hda
olpc-redhat-stream-development-devel_ext3.img
```

You should then see the standard Linux boot from GNU GRUB (GRand Unified Boot loader). At the end is the startup window, which requests that you provide a user name (see Figure 2). After you provide a name, you can optionally change the color scheme. In the end, you'll end up with the home screen for the XO-1 laptop, which you explore in the next section.

Figure 2. First-time login for the XO-1 laptop



If you'd like to leave QEMU and return the mouse context to your host operating system, press Ctrl-Alt. To return to the XO-1, click within the Sugar frame, and the mouse context returns to QEMU.

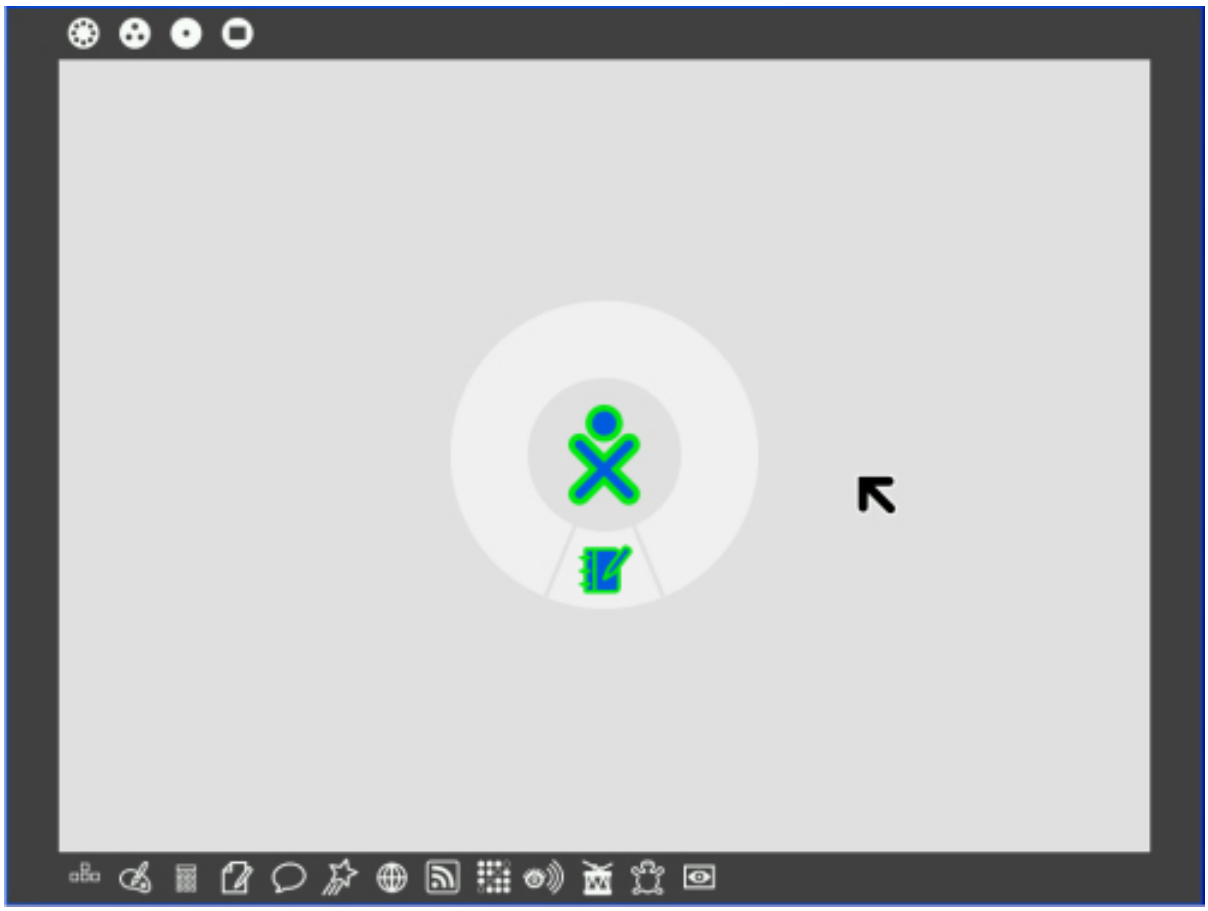
Section 4. Navigating Sugar

Now that you're logged in and ready to start, explore the OLPC Sugar UI.

Home mode and the frame

The main window in Sugar is called *Home mode* (see Figure 3). This window is basically the desktop and tells you about your environment, activities, and so on. In the center of the window is the XO icon, which represents you and your laptop. If other users were visible to you wirelessly, you would see them, as well. To shut down the laptop, place the mouse cursor over the **X** icon, then click the shutdown option.

Figure 3. Home mode for the XO-1 laptop



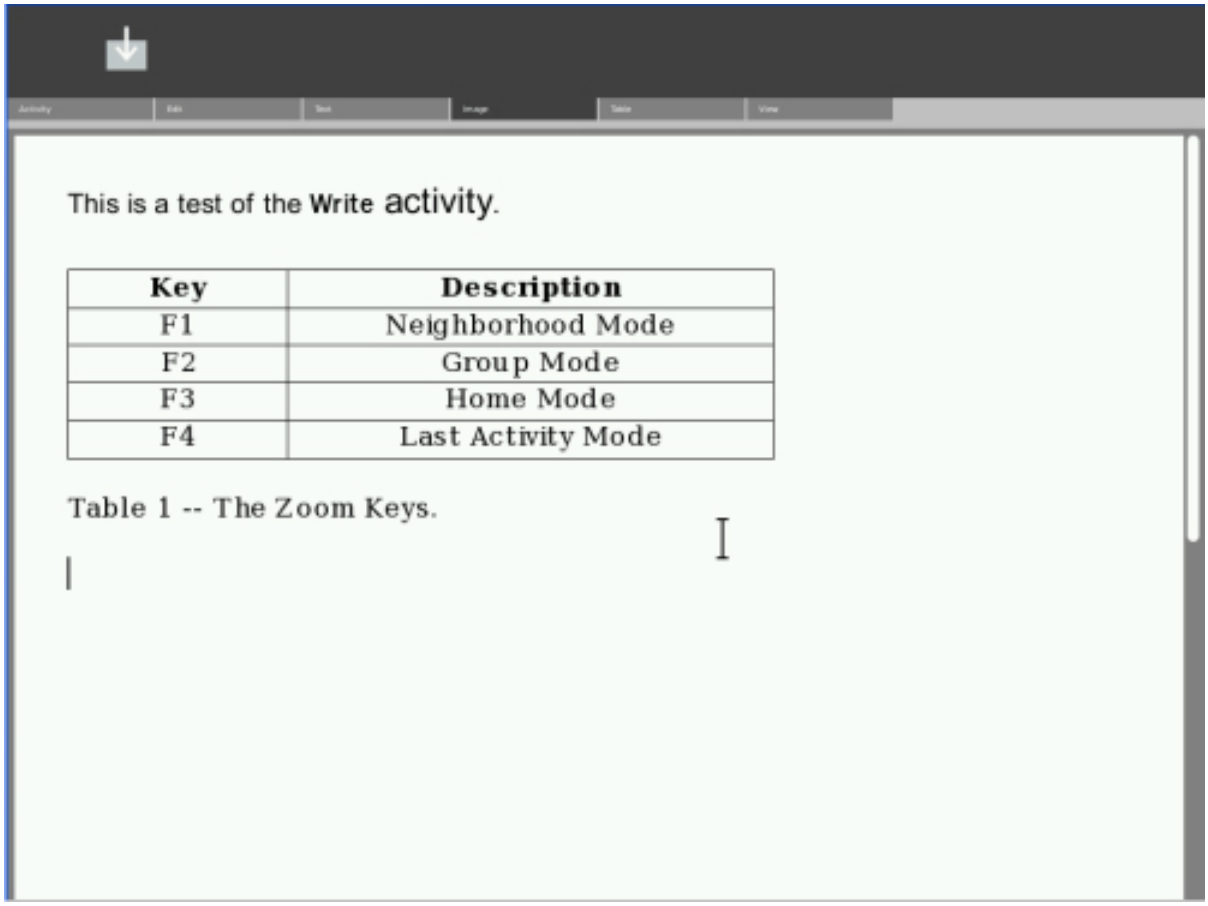
The ring around your laptop shows you the activities that are currently running. You can click any icon here to return to a running activity. To return to Home mode, move the mouse to a corner of the window. Doing so returns the frame to view. Notice the four circle "zoom" icons at the top left of the frame: The last represents the most recently accessed activities. Clicking this icon takes you back to that activity (which occupies the entire window). The second-to-last icon (the icon with a single dot in the center) represents the Home window. Clicking this icon takes you back to Home

mode. The next icon (the icon with three dots) shows you the *groups,* which are local friends and the activities that they're running. Finally, the first icon (the icon with many dots) represents the entire neighborhood. This icon shows all users and the activities they're sharing.

Start an activity

At the bottom of the frame is the collection of activities that you can run. An *activity* is an application; as you saw in Figure 3, the XO-1 laptop supports several activities. Sugar comes equipped with a Web browser, a calculator, a paint program, a news reader, and other programs that you can share with other users to collaborate on projects. To start an activity, click that activity's icon in the lower part of the frame. In Figure 4, I've started the Write activity and added some text.

Figure 4. A running activity in Sugar



Control keys

Using the mouse to navigate is simple enough, but Sugar allows control keys for

faster transitions between applications and modes. Table 1 lists several important control sequences that you can use to easily switch between modes.

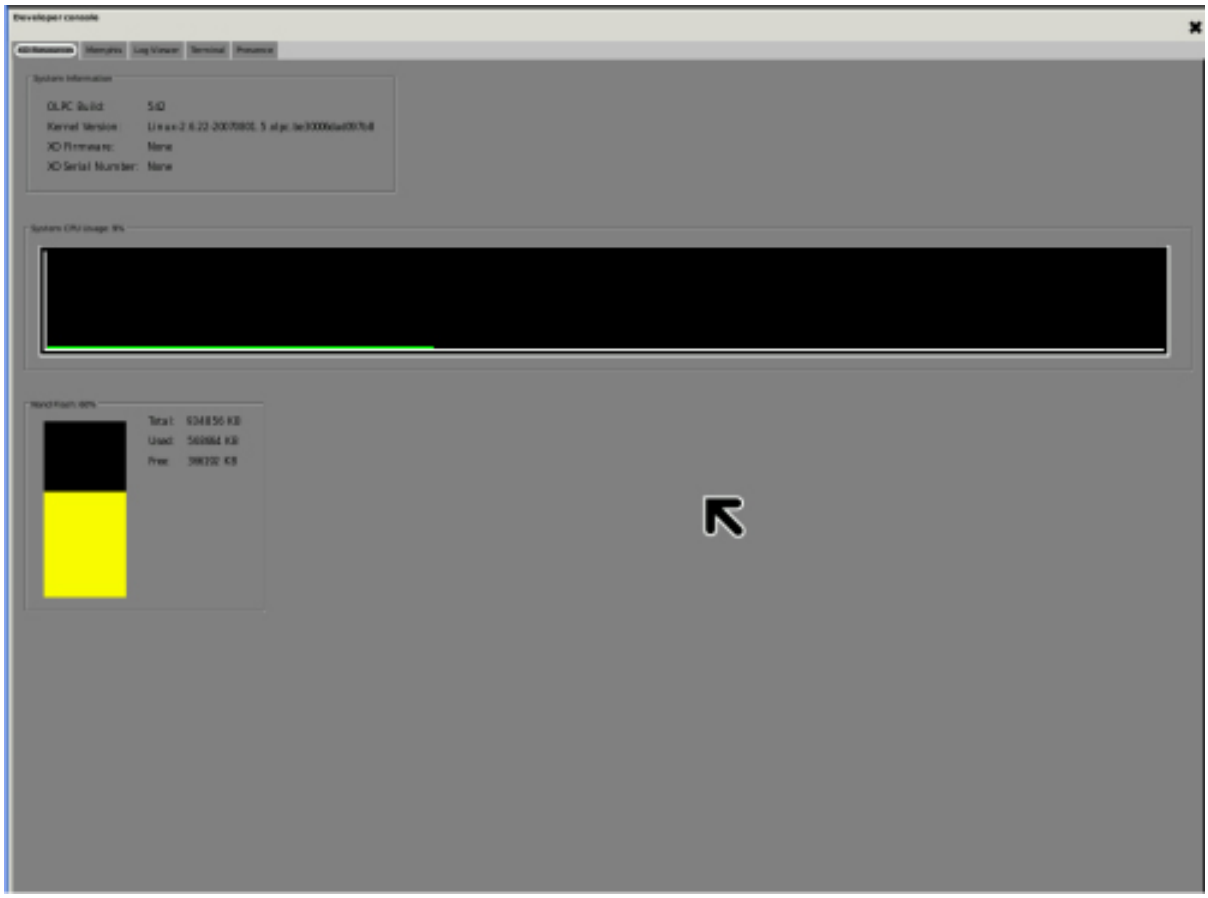
Table 1. Control sequences for switching between modes

Control key	Action
F1	Switch to neighborhood view
F2	Switch to group view
F3	Switch to home mode
F4	Switch to the last activity

Developer console

Because Sugar is built on the Linux operating system, development work requires access to a shell. You can access the shell by pressing Alt-=, which brings up a login prompt. From here, type `root` to call up a friendly Bash prompt. You can also access a shell through the developer's console, shown in Figure 5. This console includes five tabs for accessing a resource panel, a log viewer, a presence panel (for wireless attached XO-1 laptops), and a terminal (Bash shell), respectively.

Figure 5. The developer console



Whenever you want to return to Sugar, press **Ctrl-Alt-F3**. The shell remains active, so you can conveniently switch back and forth between Sugar and Bash.

Other shortcuts include **Alt-F** to toggle the frame (that is, the border around the window) on and off. You can exit the current activity by pressing **Alt-C** and terminate the Sugar GUI by pressing **Ctrl-Alt-Backspace**.

Extend the XO laptop

Downloading a file system image for the XO-1 laptop means that you get a kernel and root file system in one. The file system contains all the necessary applications and OLPC components to power the XO-1 laptop. But this doesn't mean that you can't extend the software. One of the resident packages in the file system, *YUM* (Yellow-dog Updated, Modified), is a package-management system that simplifies the downloading, installation, and configuration of software packages.

With YUM, you can download and install new packages as well as update all existing packages on a system. For example, Listing 4 shows how to update packages that have changed since the last update.

Listing 4. Using YUM to install new updates

```
-bash-3.2# yum -y update
Loading "installonlyn" plugin
Setting up Update Process
olpc_development          100%
|=========================| 1.1 kB    00:00
primary.xml.gz            100%
|=========================| 2.9 MB    00:17
olpc_devel:
##################################################
11448/11448
olpc_devel_kernel_repo    100%
|=========================|  951 B    00:00
olpc_devel:
################################################## 23/23
No Packages marked for Update/Obsoletion
-bash-3.2#
```

Section 5. Working with Python

This section briefly introduces Python programming. If you're already familiar with Python, you can skip this section and go to a review of the Sugar APIs.

What is Python?

It's not surprising that Python was chosen as the core of the XO-1 laptop. Most importantly, Python is an open source language, which means that you can see the source, modify it, and improve it. Python is a very clean language that is simple and includes a large library of APIs to simplify the task of programming complex applications.

Python is also an interpreted language, which means that it's easy to prototype activities without the step of compilation. You can compile Python for speed, or you can use libraries programmed in lower-level languages for greater performance.

While Python was originally developed as a new language to help users learn programming (the BASIC of today), its simplicity and power are appreciated by beginning and advanced developers alike.

Python by (short) example

Listing 5 provides a simple Python program that introduces the language and its use. The Python program begins by opening a file (the name of the program itself) for

reading. The handle to this open file is stored in the `infile` variable (whose type is determined as it's assigned). Next, a simple `for` loop is started using an iterator. (Note the use of the `in` keyword, which creates an iterator.) This `for` loop simply says, "for each line in the file, do."

The internals of the loop are shown indented (which is Python's way of establishing a block). The block for this `for` loop is to emit the line that was just read. The `print` statement emits the information shown, and the trailing comma (,) indicates that you don't want an **\n** character emitted at the end (because it's already contained in the line string).

Listing 5. File input and iteration

```
infile = file('pythprog.py', 'r')
for line in infile:
        print 'Read: ', line,
```

Iterators are an important part of Python and one of the main reasons the language is so simple and readable.

Python programming paradigms

In addition to being simple, Python as a multi-paradigm language supports varying styles of application development. Here are a few examples to help you explore what is meant by *multi-paradigm.*

Imperative programming

Python supports *imperative programming,* which focuses on sequences of statements to be performed. An imperative program is shown in Listing 6. In this example, you create a function (with the `def`, or *define,* statement). Afterwards, you call this new function with the `filename` argument.

Listing 6. A simple imperative program

```
def print_file(filename):
        infile = file(filename, 'r')
        for line in infile:
                print 'Read: ', line,

print_file("pythprog.py")
```

Object-oriented programming

You can also program in the object-oriented paradigm, which focuses on classes and methods that operate on data within the instance of the class. As shown in

Listing 7, the class contains two methods. The first method is `__init__()`, which is the constructor for the class. The second method is the one that performs the print of the file. Use of this class and calling the method is shown following definition of the class.

Listing 7. A simple object-oriented program

```
class PrintFile():

        def __init__(self, filename):
                self.filename = filename

        def printit(self):
                infile = file(self.filename, 'r')
                for line in infile:
                        print 'Read: ', line,

myPF = PrintFile("pythprog.py")
myPF.printit()
```

Functional programming and mixed-paradigm programming

Python also supports functional programming and mixed-paradigm programming, which allow you to, for example, use functional language concepts within an object-oriented application.

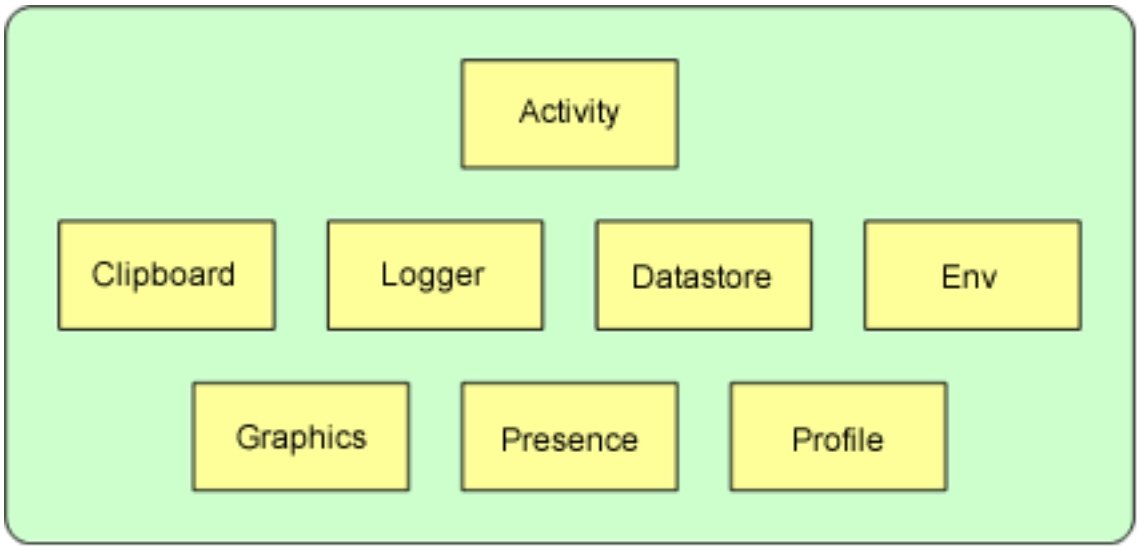
Section 6. The Sugar interface

This section introduces you to the architecture of Sugar and the elements necessary for building a graphical activity in Sugar.

Sugar architecture

Sugar is a collection of core libraries and widgets that implement the GUI and its interfaces for activity development. Figure 6 shows a partial view of the Sugar architecture.

Figure 6. Partial view of the Sugar architecture



As you can see from Figure 6, the architecture consists of the following elements:

- **`Activity` class:** At the base (for an activity) is the `Activity` class. (This class is discussed in detail in the next section, Structure of a new activity). This class is the basic interface of an activity in Sugar.
- **Clipboard:** The clipboard provides an interface to the clipboard service (a simplified API over the dbus) for adding, retrieving, or removing objects from the clipboard.
- **Logger:** The logger module provides your activity with an interface for emitting log (text) information as well as setting filtering options.
- **Datastore:** The datastore package allows your activity to store and retrieve information in a simple and convenient way for current execution of an activity.
- **Env:** Module env provides a collection of file path functions for the Sugar environment.
- **Graphics:** The graphics package provides an assortment of classes for graphics and their controls.
- **Presence:** The presence package provides the means of collaborating on activities with other laptops. It extends several services for native network presence and information sharing.
- **`Profile` class:** Finally, the `profile` class provides a way to store user-specific options and settings.

Activity class

The `Activity` class is the base class for all user-developed activities (a subclass of `GtkWindows`) and is the way that your activity hooks into Sugar. The `Activity` class provides the means for Sugar to launch your activity when requested from a mouse event.

In addition to providing a way to invoke the activity from the Sugar desktop, the activity classes provide several other useful capabilities. For example, the activity can request to be shared, making it available for collaboration with other users on the network. The method share identifies whether the activity should be publicly visible or shared by invitation only. The `Activity` inherits literally hundreds of methods from other classes such as `Window Container`, `Widget`, `Object`, and `GInterface`.

PyGTK

The Sugar UI is based on the Python GIMP Toolkit (PyGTK). GTK is a multi-platform API that includes a comprehensive set of libraries for developing graphical applications.

PyGTK is the Python interface to the GTK+ toolkit and is based on three libraries:

- **GLib:** GLib is the low-level library and forms the core of GTK+. It provides interfaces for run time functionality (such as threads, dynamic loading, and the object system).
- **Pango:** The Pango library consists of APIs for text layout and rendering.
- **ATK:** The ATK library defines a set of interfaces for accessibility (including tools such as magnifiers, screen readers, and support for alternative input devices).

In the next section, you use PyGTK, events, and callbacks to develop a simple graphical activity.

Section 7. Structure of a new activity

In this section, you develop a new activity and explore the process of integrating with Sugar.

Basic architecture

Developing a Sugar activity (based on PyGTK) requires a slightly different paradigm than you may be used to. PyGTK is an event-driven toolkit, which means that you install callbacks for events of interest, and then control is released to Sugar.

Events, such as mouse movement, a button press, loss or gain of focus in a window, or the click of a mouse button, occur within the window. When these events occur, a signal is emitted by the "widget" that they affect. For example, the mouse click may occur within a window, or the click could have been performed on a button. The widgets associated with these two events are different, and they may result in different actions. For this reason, each widget can support its own set of callbacks for the variety of events that it supports.

For each widget for which you want to support some kind of feedback, you create a callback function, and then tie the callback to the event for the widget. This process forms the basis for control in a graphical activity.

In this activity, you draw a colored rectangle, and then draw a smaller colored rectangle at each point in the window at which the user presses the left mouse button.

The __init__ method

Let's start with a discussion of the constructor for your activity (in addition to the basic class definition). Listing 8 provides the class and `__init__ method()`. (I cover the callbacks shortly.) First, you must import the modules that you use in this activity. Next, declare your class, which uses as its base class the `Activity` class. Recall that the `Activity` class serves as your hook into the Sugar infrastructure.

Listing 8. The class and __init__ constructor method (ExampleActivity.py).

```
import pygtk
pygtk.require('2.0')
import gtk, sys, random, time

from sugar.activity import activity

class ExampleActivity(activity.Activity):

        def __init__(self, handle):
                activity.Activity.__init__(self, handle)

                self.set_title("Simple Graphical
Activity")

                # Show the toolbox elements
                toolbox = activity.ActivityToolbox(self)
                self.set_toolbox(toolbox)
                toolbox.show()

                # Register the event callbacks
                self.connect("expose_event",
self.area_expose_cb)
```

```
                self.connect("key_press_event",
self.keypress_cb)
                self.connect("button_press_event",
self.button_press_cb)
                self.show()

                # Seed the random number generator
                random.seed(time.time())

                self.area = self.window
                self.gc = self.area.new_gc()

                # Allocate a color dictionary
                self.colors = {}
                self.colormap = self.gc.get_colormap()
                self.colors[0] =
self.colormap.alloc_color('green')
                self.colors[1] =
self.colormap.alloc_color('blue')
                self.colors[2] =
self.colormap.alloc_color('black')
                self.colors[3] =
self.colormap.alloc_color('red')

                # Register for events
                self.area.set_events(
                        gtk.gdk.EXPOSURE_MASK      |
                        gtk.gdk.KEY_PRESS_MASK     |
                        gtk.gdk.LEAVE_NOTIFY_MASK  |
                        gtk.gdk.BUTTON_PRESS_MASK )
```

The `__init__()` method is the constructor and is called as an instance is created for this class (that is, when the Sugar user clicks its icon). It begins by calling the activity constructor (to allow it to do its initialization). Next, you provide a simple title. A toolbox is created, which appears at the top of the window when the activity has begun. This toolbox provides an exit button and other tools.

Next, register your callbacks for the events that you're interested in. Another way to think about this is that you're creating signal handlers for the signals of interest (much like X). As shown, the `connect()` method ties a particular event to a callback. The `expose_event` can occur for numerous reasons (such as the window being exposed), but the result should be a redraw of the window. The `key_press_event` indicates that a key was pressed on the keyboard while the window was in focus. Finally, the `button_press_event` indicates that a mouse button was pressed in the window. Each event is associated with its own callback.

Notice the random number generator with a call to the `seed()` method of the `random` module. You save the window in an instance variable called `area`, and then a new graphics context is created with a call to `new_gc()`.

The next step is to create a color dictionary (`colors`). A *dictionary* is a hash in Python that allows you to create an array to associate values with other values. You use it here as an array, but the index could have just as well been another type.

Finally, you enable receipt of events with a call to `set_events()`. This method

represents a bitmask and tells PyGTK to send those signals when they occur in your context.

Event callbacks

The callbacks (and the method you use to draw in the window) are shown in Listing 9.

Listing 9. The event callbacks and draw_box method (ExampleActivity.py part 2)

```
        def draw_box(self, x, y):
                self.gc.set_foreground(
self.colors[random.randint(1,3)] )
                self.area.draw_rectangle(self.gc, True, x,
y, 50, 50)

        def area_expose_cb(self, area, event):
                self.gc_set_foreground(self.colors[0])
                self.area.draw_rectangle(self.gc, True,
200, 200, 400, 400)

                return True

        def keypress_cb(self, widget, event):
                sys.exit()

        def button_press_cb(self, widget, event):
                if event.button == 1:
                        self.draw_box(event.x, event.y)

                return True
```

The first callback, `draw_box`, is used to draw a new box (upon user request). The method takes `self` (representing the instance of this class), and then the `x` and `y` positions to draw. The foreground color is set with a random color from the dictionary (with the key of the dictionary being provided as a random index). The `draw_rectangle()` method is used to draw a filled rectangle, providing the graphics context, location, and size.

The `area_expose_cb` callback is used to redraw the window. This callback occurs if the window is exposed (focus returns to it, for example). The foreground color is set (using the first index of the color dictionary, so the color green is used), and then a large rectangle is drawn.

The `keypress_cb` callback is called when a key is pressed in the current window. For this event, simply exit the activity.

Finally, the `button_press_cb` callback is called when a mouse button is pressed in the window. The callback checks to see whether the left button was pressed; if it was, it calls the `draw_box()` method to draw a new box at the current location of

the cursor in the window.

Section 8. Installing a new activity

This section explores how to integrate a Sugar activity into the desktop. It integrates the simple graphical activity you developed in the last section into Sugar for execution from home mode.

Getting to the shell

The first step is to get to the Linux shell, which is easily accomplished from home mode by pressing **Alt-Ctrl-F1**. Log in as `root`, which requires no password. From here, change to the *olpc* user, as shown in Listing 10.

Listing 10. Getting to the shell for user olpc

```
-bash-3.2# su - olpc
[olpc@xo-12-34-56 ~]$ pwd
/home/olpc
[olpc@xo-12-34-56 ~]$
```

At this point, you're in the /home/olpc directory, which is right where you want to be. Create a new directory here (call it *ExampleActivity*), and change the directory to it:

```
[olpc@xo-12-34-56 ~]$ mkdir ExampleActivity
[olpc@xo-12-34-56 ~]$ cd ExampleActivity
[olpc@xo-12-34-56 ExampleActivity]$
```

The activity directory

In this directory (/home/olpc/ExampleActivity), place (that is, type in) the source file called *ExampleActivity.py* (the concatenation of Listing 8 and Listing 9). You'll also want the file shown in Listing 11 in the same directory. This is a special file for Sugar that bundles the activity for use.

Listing 11. setup.py

```
#!/usr/bin/env python
from sugar.activity import bundlebuilder
if __name__ == "__main__":
```

```
        bundlebuilder.start("ExampleActivity")
```

When this file is created, ensure that it's executable. To do so, type:

```
[olpc@xo-12-34-56 ExampleActivity]$ chmod +x setup.py
[olpc@xo-12-34-56 ExampleActivity]$
```

Create the support files

Just a few more support files, and you're done. In the ExampleActivity subdirectory, create a new directory called *activity,* and then change directory into it:

```
[olpc@xo-12-34-56 ExampleActivity]$ mkdir activity
[olpc@xo-12-34-56 ExampleActivity]$ cd activity
[olpc@xo-12-34-56 activity]$
```

Two additional support files are needed here, representing the icon to use for the activity in Sugar, and an information file to represent the activity, respectively. The icon file (a Scalable Vector Graphics [SVG] file) is provided in Listing 12, and the information file is shown in Listing 13.

Listing 12. The SVG icon file for the activity (ExampleActivity.svg)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" [
  <!ENTITY fill_color "#FFFFFF">
  <!ENTITY stroke_color "#000000">
]>
<svg xmlns="http://www.w3.org/2000/svg" width="50"
height="50">
  <rect x="1" y="1" width="48" height="48"
style="fill:&fill_color;;stroke:&stroke_color;;stroke_width:2"//>
</svg>
```

Listing 13. The activity information file (activity.info)

```
[Activity]
name = ExampleActivity
service_name = org.laptop.ExampleActivity
class = ExampleActivity.ExampleActivity
icon = ExampleActivity
activity_version = 1
show_launcher = yes
```

Finally, go back up to the /home/olpc/ExampleActivity directory, and then create a file called *MANIFEST*, which will contain a single line:

```
ExampleActivity.py
```

This file represents the source files that make up the package. The directory structure should now appear with the files defined in Listing 14.

Listing 14. Structure and files of the ExampleActivity package

```
/home/olpc/ExampleActivity/
/home/olpc/ExampleActivity/ExampleActivity.py
/home/olpc/ExampleActivity/MANIFEST
/home/olpc/ExampleActivity/setup.py
/home/olpc/ExampleActivity/activity/
/home/olpc/ExampleActivity/activity/ExampleActivity.svg
/home/olpc/ExampleActivity/activity/activity.info
```

Install the new bundle

Before you start the installation, be aware that you may have to provide the symbol SUGAR_PATH. If you type `echo $SUGAR_PATH` and a blank line results, type the following:

```
[olpc@xo-12-34-56 ExampleActivity]$ export
SUGAR_PATH=/share/sugar
[olpc@xo-12-34-56 ExampleActivity]$
```

Now, use the `cd` command in the root directory of your package (that is, `cd /home/olpc/ExampleActivity`), and then type:

```
[olpc@xo-12-34-56 ExampleActivity]$ python setup.py dev
```

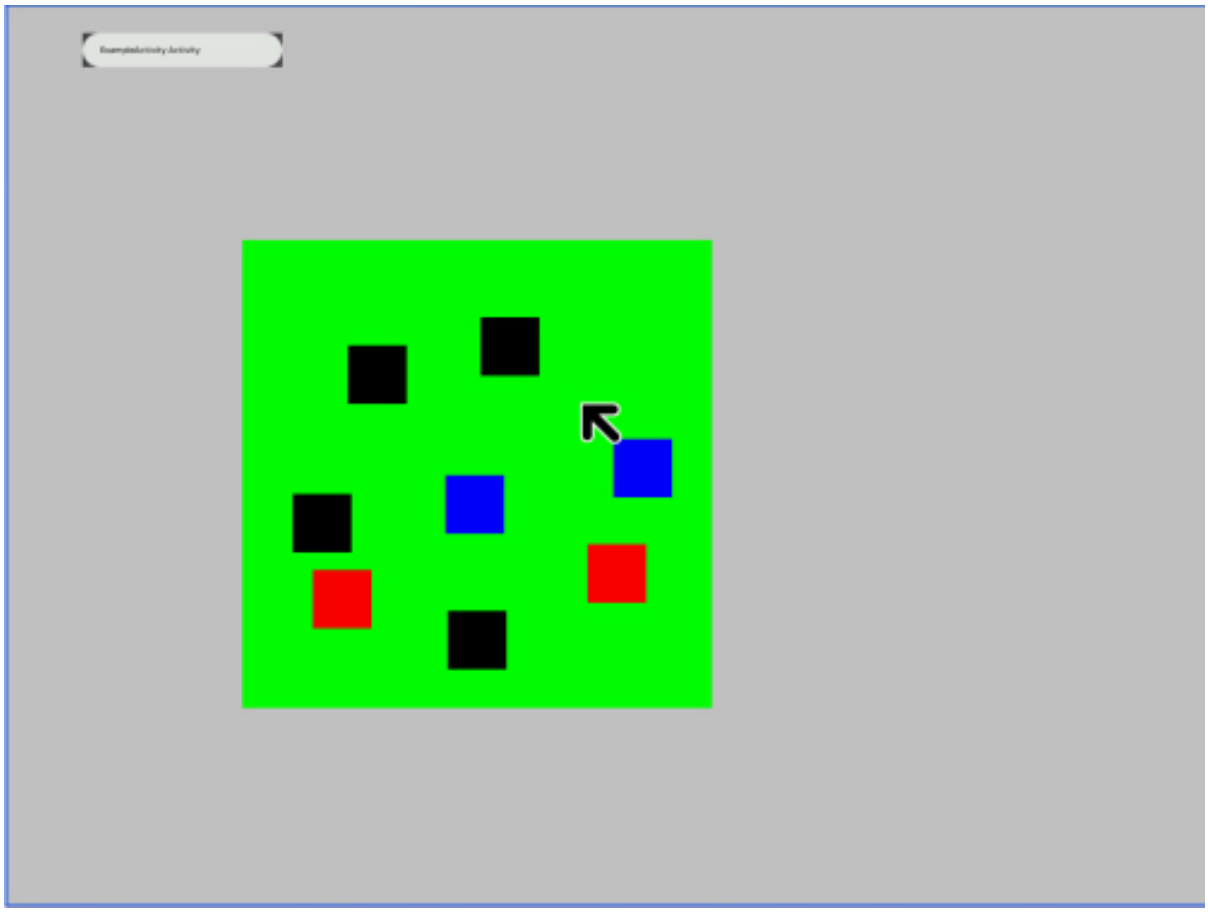
Doing so creates a symbolic link from the Activities subdirectory (in /home/olpc) to your package directory. You can now test your new activity from Sugar.

Test the activity

To test the activity, first return to Sugar's home mode. To do this, press **Ctrl-Alt-F3**. Next, restart Sugar to make the activity visible. To restart Sugar, press **Ctrl-Alt-Backspace**. The terminal window reappears for a few seconds, and Sugar restarts. In the frame at the bottom of home mode, a new icon appears (as an empty box). Place your mouse cursor over this icon to display the activity name (*ExampleActivity*). Click the icon, and the new activity appears (consuming the entire screen). After left-clicking a few times in the green square, the window shown in

Figure 7 appears.

Figure 7. The finished simple activity



Distribute your bundle

If you wanted to distribute your activity for others to use, an additional step is required. In the ExampleActivity subdirectory, type the following command to create a bundle (which contains all the necessary files for the activity):

```
[olpc@xo-12-34-56 ExampleActivity]$ python setup.py dist
```

The result is a file called *ExampleActivity-1.xo*, which can be distributed to others. Other users can migrate this file to their laptops and install it using the `sugar-install-bundle` command. For example:

```
[olpc@xo-12-34-56 ExampleActivity]$ sugar-install-bundle
ExampleActivity-1.xo
```

Section 9. Summary

The XO-1 laptop has entered mass production, so the product is now a reality. At the time of this writing, the OLPC foundation offered XO-1 laptops for sale at USD399 (which purchased two laptops: one for yourself and one for a child in a developing nation). The Linux kernel and user-space software continue development and have evolved nicely over the past six months. The software framework is very stable, and the APIs solid enough for developers not closely related to the project.

You'll also find a large number of applications developed for Sugar, including development tools (see Resources for more detail). Bindings for languages other than Python have appeared as well, so soon you'll be able to develop applications in JavaScript and other languages. It will be interesting to see what evolves from the XO-1 laptop after it gets into the hands of the children of the world. It could be a programming revolution, with Linux and Python leading the way.

Resources

Learn

- In Tim's article "Sugar, the XO laptop, and One Laptop per Child" (developerWorks, Apr 2007), take a tour of Sugar, the XO laptop's novel user interface.
- In this developerWorks interview with OLPC visionary Walter Bender (developerWorks, Apr 2007), hear about the OLPC vision, challenges faced, and progress to date.
- The official OLPC Web site introduces the philosophy and vision of the OLPC project and explores how the laptop will benefit children around the world.
- Wikipedia has a great introduction to the XO-1 laptop, it's history, design, software, and features. The laptop is novel, but some critics wonder whether the money spent on the laptop in some developing countries would be better spent elsewhere.
- This Sugar activity tutorial has introduced many developers to developing with Sugar.
- The Open Firmware standard (described by the IEEE) is the basis for the XO-1 BIOS. In the spirit of open source firmware, even the XO's BIOS is open source.
- Python is a great language for beginners and experienced programmers alike. It's a multi-paradigm language that can be used in interpretive or compiled forms. Its official Web site provides a wealth of information, including documentation and many tutorials.
- In "System emulation with QEMU" (developerWorks, September 2007), learn about QEMU and its use as a platform emulator. This article introduces you to virtualization, then discusses the internals of QEMU and its use. For more detail on virtualization and the range of Linux solutions, check out "Virtual Linux" (developerWorks, December 2006).
- A list of activities and projects for the XO-1 is on the laptop.org site. In the early days, the list was sparse, but today you'll find a large number of activities ranging from educational activities, audio and video players, media-creation tools, programming tools, and necessary network applications (such as browsers, feed-readers, and mail programs). You'll even find a variety of games. (Python is a great language for game development.)
- Sugar is based on the Python GTK (PyGTK), which is a GUI toolkit with Python bindings. PyGTK is multi-platform and was a great choice for the XO-1. At the PyGTK Web site, you'll find a wealth of information, including several tutorials. The Sugar APIs are large and extensive, as shown in the OLPC Sugar User Interface list.

- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tips and Linux tutorials on developerWorks.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- The OLPC organization is running a limited-time "Give One Get One" program. For a donation of a set amount, the organization will send one XO laptop to a child in a developing nation and one to you.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the developerWorks community through blogs, forums, podcasts, and community topics in our new developerWorks spaces.

About the author

M. Tim Jones
M. Tim Jones is an embedded software engineer and the author of *GNU/Linux Application Programming, AI Application Programming* (now in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective.* His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a Consultant Engineer for Emulex Corp. in Longmont, Colorado.

Trademarks