MAKE YOUR OWN SUGAR ACTIVITIES!

Published : 2014-03-05

SUGAR ACTIVITIES

1. INTRODUCTION

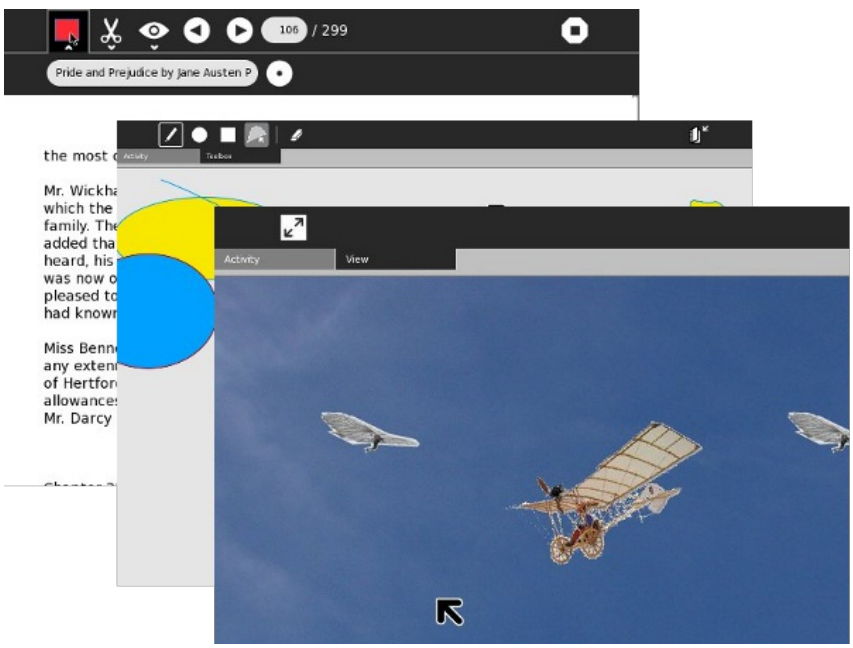
"This book is a record of a pleasure trip. If it were a record of a solemn scientific expedition, it would have about it that gravity, that profundity, and that impressive incomprehensibility which are so proper to works of that kind, and withal so attractive."

From the Preface to *The Innocents Abroad*, by Mark Twain

The purpose of this book is to teach you what you need to know to write Activities for Sugar, the operating environment developed for the One Laptop Per Child project. This book does not assume that you know how to program a computer, although those who do will find useful information in it. My primary goal in writing it is to encourage non programmers, including children and their teachers, to create their own Sugar Activities. Because of this goal I will include some details that other books would leave out and leave out things that others would include. Impressive incomprehensibility will be kept to a minimum.

If you just want to learn how to write computer programs Sugar provides many Activities to help you do that: Etoys, Turtle Art, Scratch, and Pippy. None of these are really suitable for creating Activities so I won't cover them in this book, but they're a great way to learn about programming. If you decide after playing with these that you'd like to try writing an Activity after all you'll have a good foundation of knowledge to build on.

When you have done some programming then you'll know how satisfying it can be to use a program that you made yourself, one that does *exactly* what you want it to do. Creating a Sugar Activity takes that enjoyment to the next level. A useful Sugar Activity can be translated by volunteers into every language, be downloaded hundreds of times a week and used every day by students all over the world.



A book that teaches *everything* you need to know to write Activities would be really, really long and would duplicate material that is already available elsewhere. Because of this, I am going to write this as sort of a guided tour of Activity development. That means, for example, that I'll teach you what Python is and why it's important to learn it but I won't teach you the Python language itself. There are excellent tutorials on the Internet that will do that, and I'll refer you to those tutorials.

There is much sample code in this book, but there is no need for you to type it in to try it out. All of the code is in a Git repository that you can download to your own computer. If you've never used Git there is a chapter that explains what it is and how to use it.

I started writing Activities shortly after I received my XO laptop. When I started I didn't know *any* of the material that will be in this book. I had a hard time knowing where to begin. What I did have going for me though was a little less than 30 years as a professional programmer. As a result of that I think like a programmer. A good programmer can take a complex task and divide it up into manageable pieces. He can figure out how things *must* work, and from that figure out how they *do* work. He knows how to ask for help and where. If there is no obvious place to begin he can begin *somewhere* and eventually get where he needs to go.

Because I went through this process I think I can be a pretty good guide to writing Sugar Activities. Along the way I hope to also teach you how to think like a programmer does.

From time to time I may add chapters to this book. Sugar is a great application platform and this book can only begin to tell you what is possible.

In the first edition of this book I expressed the wish that future versions of the book would have guest chapters on more advanced topics written by other experienced Activity developers. That wish has been realized. Not only does the book have guest chapters, but some of the new content has been created by young developers from the *Google Code-in* of 2012, for which I was one of the mentors. Their bios are in the **About The Authors** chapter. Read them and prepare to be impressed!

This book eliminates some content that was part of the first edition. The current version of Sugar uses GTK 3, and most of the examples in this addition use GTK 3 as well. In the first edition all the examples used GTK 2. The first edition also explained how to support multiple versions of Sugar with one Activity. That is no longer relevant to most Activity developers, so it has been removed.

If you need the content of the first edition you will find it at archive.org. Code examples from the first edition are also still available at git.sugarlabs.org.

In addition to examples using GTK3 this edition introduces the use of HMTL 5 and WebKit to create Activities. The advantage of HTML 5 is that it can run on Android and other tablet devices. There has been a fair amount of interest in using these devices for education, and if you develop using HTML 5 you can make versions of your Activity for both Sugar and Android.

FORMATS FOR THIS BOOK

This book is part of the *FLOSS Manuals* project and is available for online viewing at their website:

http://en.flossmanuals.net/

At this time this is the only place the second edition is available, but it will in time be made available everywhere the first edition is.

You will can purchase a printed and bound version of the first edition of book at *Amazon.com*:

http://www.amazon.com/James-Simmons/e/B005197ZTY/ref=ntt_dp_epwbk_0

The *Internet Archive* has the first edition of this book available as a full color PDF, as well as EPUB, MOBI, and DjVu versions, all of which you can download for free:

http://www.archive.org/details/MakeYourOwnSugarActivities

The *Amazon Kindle Store* has exactly the same MOBI version as the Internet Archive does.

If you choose to read this book on a Kindle or a Nook be aware that these devices have narrow screens not well suited for displaying program listings. I suggest you refer to the FLOSS Manuals website to see what the code looks like properly formatted.

There is a Spanish version of the first edition of this book, **Cómo Hacer Una Actividad Sugar**, which is available in all the same places as this version.

2. WHAT IS SUGAR?

Sugar is the user interface designed for the XO laptop. It can now be installed on most PCs, including older models that can't run the latest Windows software. You can also install it on a thumb drive (Sugar on a Stick) and boot your PC from that.

When the XO laptop first came out some people questioned the need for a new user interface. Wouldn't it be better for children to learn something more like what they would use as adults? Why not give them Microsoft Windows instead?

This would be a reasonable question if the goal was to train children to use computers and nothing else. It would be even more reasonable if we could be sure that the software they would use as adults looked and worked like the Microsoft Windows of today. These are of course not reasonable assumptions.

The OLPC project is not just about teaching computer literacy. It is about teaching *everything*: reading, writing, arithmetic, history, science, arts and crafts, computer programming, music composition, and everything else. Not only do we expect the child to use the computer for her school work, we expect her to take it home and use it for her own explorations into subjects that interest her.

This is a great deal more than anyone has done with computers for education, so it is reasonable to rethink how children should work with computers. Sugar is the result of that rethinking.

Sugar has the following unique features:

THE JOURNAL

The Journal is where all the student's work goes. Instead of files and folders there is a list of Journal entries. The list is sorted in descending order by the date and time it was last worked on. In a way it's like the "Most Recently Used" document menu in Windows, except instead of containing just the last few items it contains everything and is the normal way to save and resume work on something.

The Journal makes it easy to organize your work. Any work you do is saved to the Journal. Anything you download from the web goes in the Journal. If you've ever downloaded a file using a web browser, then had to look for it afterwards because it went in some directory other than the one you expected, or if you ever had to help your parents when they were in a similar situation, you can understand the value of the Journal.

The Journal has metadata for each item in it. Metadata is information about information. Every Journal entry has a title, a description, a list of keywords, and a screen shot of what it looked like the last time it was used. It has an activity id that links it to the Activity that created it, and it may have a MIME type as well (which is a way of identifying Journal entries so that items not created by an Activity may still be used by an Activity that supports that MIME type).

In addition to these common metadata items a Journal entry may be given custom metadata by an Activity. For instance, the **Read** Activity uses custom metadata to save the page number you were reading when you quit the Activity. When you resume reading later the Activity will put you on that page again.

In addition to work created by Activities, the Journal can contain Activities themselves. To install an Activity you can use the **Browse** Activity to visit the website http://activities.sugarlabs.org and download it. It will automatically be saved to the Journal and be ready for use. If you don't want the Activity any more, simply delete it from the Journal and it's *completely gone*. No uninstall programs, no dialog boxes telling you that such and such a .DLL doesn't seem to be needed anymore and do you want to delete it? No odd bits and pieces left behind.

COLLABORATION

The second unique feature Sugar is Collaboration. Collaboration means that Activities can be used by more than one person at the same time. While not every Activity needs collaboration and not every Activity that could use it supports it, a really first rate Activity will provide some way to interact with other Sugar users on the network. For instance, all the e-book reading Activities provide a way of giving a copy of the book you're reading (with any notes you added to it) to a friend or to the whole class. The **Write** Activity lets several students work on the same document together. The **Distance** Activity lets two students see how far apart from each other they are.

There are five views of the system you can switch to at the push of a button (Function Keys F1-4). They are:

- The Neighborhood View
- The Friends View
- The Activity Ring
- The Journal

Of these Views, the first two are used for Collaboration.

The Neighborhood View shows icons for everyone on the network. Every icon looks like a stick figure made by putting an "O" above an "X". Each icon has a name, chosen by the student when she sets up her computer. Every icon is displayed in two colors, also chosen by the student. In addition to these "XO" icons there will be icons representing mesh networks and others representing WiFi hot spots. Finally there will be icons representing active Activities that their owners wish to share.

To understand how this works consider the **Chat** Activity. The usual way applications do chat is to have all the participants start up a chat client and visit a particular chat room at the same time. With Sugar it's different. One student starts the Chat Activity on her own computer and goes to the Neighborhood View to invite others on the network to participate. They will see a Chat icon in their own Neighborhood View and they can accept. The act of accepting starts up their own Chat Activity and connects them to the other participants.

The Friends View is similar to the Neighborhood View, but only contains icons for people you have designated as Friends. Collaboration can be offered at three levels: with individual persons, with the whole Neighborhood, and with Friends. Note that the student alone decides who her Friends are. There is no need to ask to be someone's Friend. It's more like creating a mailing list in email.

SECURITY

Protecting computers from malicious users is very important, and if the computers belong to students it is doubly important. It is also more difficult, because we can't expect young students to remember passwords and keep them secret. Since Sugar runs on top of Linux viruses aren't much of a problem, but malicious Activities definitely are. If an Activity was allowed unrestricted access to the Journal, for instance, it could wipe it out completely. Somebody could write an Activity that seems to be harmless and amusing, but perhaps after some random number of uses it could wipe out a student's work.

The most common way to prevent a program from doing malicious things is to make it run in a sandbox. A sandbox is a way to limit what a program is allowed to do. With the usual kind of sandbox you either have an untrusted program that can't do much of anything or a trusted program that is not restricted at all. An application becomes trusted when a third party vouches for it by giving it a *signature*. The signature is a mathematical operation done on the program that only remains valid if the program is not modified.

Sugar has a more sophisticated sandbox for Activities than that. No Activity needs to be trusted or is trusted. Every Activity can only work with the Journal in a limited, indirect way. Each Activity has directories specific to it that it can write to, and all other directories and files are limited to read-only access. In this way no Activity can interfere with the workings of any other Activity. In spite of this, an Activity can be made to do what it needs to do.

SUMMARY

Sugar is an operating environment designed to support the education of children. It organizes a child's work without needing files and folders. It supports collaboration between students. Finally, it provides a robust security model that prevents malicious programs from harming a student's work.

It would not be surprising to see these features someday adopted by other desktop environments.

3. WHAT IS A SUGAR ACTIVITY?

A Sugar Activity is a self-contained Sugar application packaged in a .xo bundle.

An .xo bundle is an archive file in the Zip format. It contains:

- A MANIFEST file listing everything in the bundle
- An **activity.info** file that has attributes describing the Activity as name=value pairs. These attributes include the Activity name, its version number, an identifier, and other things we will discuss when we write your first Activity.
- An icon file (in SVG format)
- Files containing translations of the text strings the Activity uses into many languages
- The program code to run the Activity

A Sugar Activity will generally have some Python code that extends a Python class called Activity. It may also make use of code written in other languages if that code is written in a way that allows it to be used from Python (this is called having **Python bindings**). For instance, you can use WebKit, an HTML component, as part of your Activity. This would enable you to write most of your Activity using HTML 5 instead of Python.

It is even possible to write a Sugar Activity without using Python at all, but this is beyond the scope of this book.

There are only a few things that an Activity can depend on being included with every version of Sugar. These include modules like Evince (PDF and other document viewing), WebKit (rendering web pages), and Python libraries like PyGTK and PyGame. Everything needed to run the Activity that is *not* supplied by Sugar must go in the bundle file. A question sometimes heard on the mailing lists is "How do I make Sugar install *X* the first time my Activity is run?" The answer: you don't. If you need *X* it needs to go in the bundle.

You can install an Activity by copying or downloading it to the Journal. You uninstall it by removing it from the Journal. There is no *Install Shield* to deal with, no deciding where you want the files installed, no possibility that installing a new Activity will make an already installed Activity stop working. For the child installing an Activity the process is no more difficult than installing an app on a smart phone.

An Activity generally creates and reads objects in the Journal. A first rate Activity will provide some way for the Activity to be shared by multiple users.

4. WHAT DO I NEED TO KNOW TO WRITE A SUGAR ACTIVITY?

If you are going to write Sugar Activities you should learn something about the topics described in this chapter. There is no need to become an expert in any of them, but you should bookmark their websites and skim through their tutorials. This will help you to understand the code samples we'll be looking at.

PYTHON

Python is the most used language for writing Activities. While you can use other languages, most Activities have at least some Python in them. Sugar provides a Python API that simplifies creating Activities. While it is possible to write Activities using no Python at all (like **Etoys**), it is unusual.

Most of the examples in this book are written entirely in Python. In a later Guest Chapter you'll see how you can mix Python and HTML 5 to make an impressive Activity.

There are compiled languages and interpreted languages. In a compiled language the code you write is translated into the language of the chip it will run on and it is this translation that is actually run by the OS. In an interpreted language there is a program called an interpreter that reads the code you write and does what the code tells it to do. (This is over simplified, but close enough to the truth for this chapter).

Python is an interpreted language. There are advantages to having a language that is compiled and there are advantages to having an interpreted language. The advantages Python has for developing Activities are:

- It is portable. In other words, you can make your program run on any chip and any OS without making a version specific to each one. Compiled programs only run on the OS and chip they are compiled for.
- Since the source code is the thing being run, you can't give someone a Python program without giving them the source code. You can learn a lot about Activity programming by studying other people's code, and there is plenty of it to study.
- It is an easy language for new programmers to learn, but has language features that experienced programmers need.
- It is widely used. One of the best known Python users is Google. They use it enough that they have a project named “Unladen Swallow” to make Python programs run faster.

The big advantage of a compiled language is that it can run much faster than an interpreted language. However, in actual practice a Python program can perform as well as a compiled program. To understand why this is you need to understand how a Python program is made.

Python is known as a “glue” language. The idea is that you have components written in various languages (usually C and C++) and they have Python bindings. Python is used to “glue” these components together to create applications. In most applications the bulk of the application's function is done by these compiled components, and the application spends relatively little time running the Python code that glues the components together.

In addition to Activities using Python most of the Sugar environment itself is written in Python.

If you have programmed in other languages before there is a good tutorial for learning Python at the Python website: http://docs.python.org/tutorial/. If you're just starting out in programming you might check out *Invent Your Own Computer Games With Python*, which you can read for free at http://inventwithpython.com/.

PYGTK

GTK+ is a set of components for creating user interfaces. These components include things like buttons, scroll bars, list boxes, and so on. It is used by GNOME desktop environment and the applications that run under it. Sugar Activities use a special GNOME theme that give GTK+ controls a unique look.

PyGTK is a set of Python bindings that let you use GTK+ components in Python programs. Sugar 3 uses GTK+ 3 and Activities written for it should also, although the previous version of PyGTK still works. GTK+3 is very different from what went before. There is a tutorial showing how to use it at the PyGTK website: http://python-gtk-3-tutorial.readthedocs.org/en/latest/. You'll also find an article on converting Activities using GTK 2 to use GTK 3 in the guest chapters section of this book.

PYGAME

The alternative to using PyGTK for your Activity is PyGame. PyGame can create images called sprites and move them around on the screen. As you might expect, PyGame is mostly used for writing games. It is less commonly used in Activities than PyGTK.

The tutorial to learn about PyGame is at the PyGame website: http://www.pygame.org/wiki/tutorials. The website also has a bunch of pygame projects you can download and try out.

PROGRAMMING

5. SETTING UP A SUGAR DEVELOPMENT ENVIRONMENT

It is not currently practical to develop Activities for the XO on the XO. It's not so much that you can't do it, but that it's easier and more productive to do your development and testing on another machine running a more conventional environment. This gives you access to better tools and it also enables you to simulate collaboration between two computers running Sugar using only one computer.

INSTALLING LINUX

If you're going to develop Sugar Activities you're going to need a machine that runs Linux. While you can write programs using Python, PyGTK, and PyGame on any operating system you're going to need Linux to make them into Sugar Activities and test them under Sugar. It is possible to write a standalone Python program in Windows and then turn over the code to another developer who runs Linux who would then make an Activity out of it. If you're just starting out in computer programming that may be an attractive option.

Installing Linux is not the test of manhood it once was. Anyone can do it. The GNOME desktop provided with Linux is very much like Windows so you'll feel right at home using it.

When you install Linux you have the option to do a dual boot, running Linux and Windows on the same computer (but not at the same time). This means you set aside a disk partition for use by Linux and when you start the computer a menu appears asking which OS you want to start up. The Linux install will even create the partition for you, and a couple of gigabytes is more than enough disk space. Sharing a computer with a Linux installation will not affect your Windows installation at all.

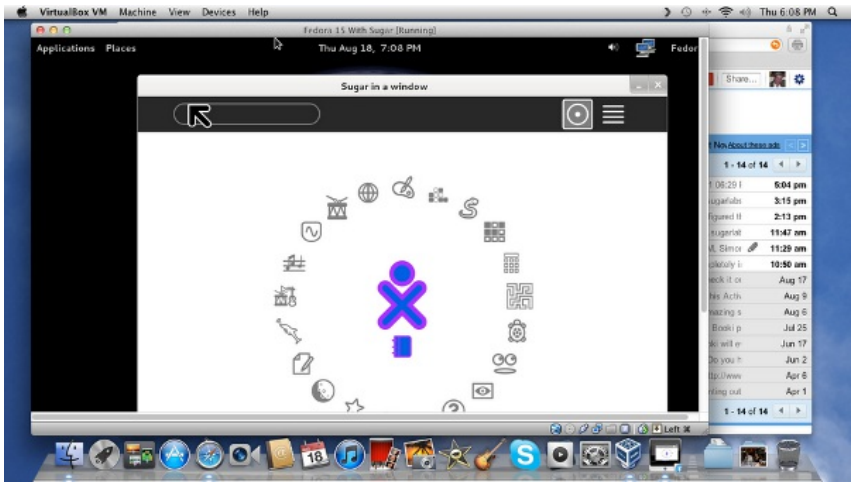
Sugar Labs has been working to get Sugar included with all Linux distributions. If you already have a favorite distribution, chances are the latest version of it includes Sugar. Fedora, openSuse, Debian, and Ubuntu all include Sugar. If you already use Linux, see if Sugar is included in your distribution. If not, Fedora is what is used by the XO computer so a recent version of Fedora might be your best bet. You can download the Fedora install CD or DVD here: https://fedoraproject.org/get-fedora.

It is worth pointing out that all of the other tools I'm recommending are included in every Linux distribution, and they can be installed with no more effort than checking a check box. The same tools often will run on Windows, but installing them there is more work than you would expect for Windows programs.

There is another way to install Linux which is to use a virtual machine. A virtual machine is a way to run one operating system on top of another one. The operating system being run is fooled into thinking it has the whole computer to itself. (Computer industry pundits will tell you that using virtual machines is the newest new thing out there. Old timers like me know that IBM was doing it on their mainframe computers back in the 1970's).

If you're used to Windows you might think that running Linux in a VM from Windows instead of installing Linux might be the easier option. In practice it is not. Linux running in a VM is still Linux, so you're still going to have to learn some things about Linux to do Activity development. Also, running a second OS in a VM requires a really powerful machine with gigabytes of memory. On the other hand, I did my first Sugar development using Linux on an IBM NetVista Pentium IV I bought used for a little over a hundred dollars, shipping included. It was more than adequate, and would still be adequate today.

Having said that, there are reasons to like the virtual machine approach. I have used this method to run Linux on a Macintosh. It looks like this:



It is really fast and easy to set this up if you have a recent Mac. The screenshot is from a Mac Mini my wife and I bought recently which has 2 gig of RAM and more disk space than we'll ever use. All I needed to do was download the Mac OS version from http://www.virtualbox.org, then download a Fedora image from http://virtualboxes.org/images/fedora/. This image needs to be unpacked before you can use it, so get *Stuffit Expander* for free from the Apple Market. Run Virtual Box, configure your unpacked image and in no time flat you have Fedora running in a window on your desktop. You can go to **Add/Remove Programs** and add Sugar and your favorite development tools and you're set.

One of the neat things about Virtual Box images is you can have more than one. The Virtual Boxes website has images from Fedora 10 up to the latest, so I can have a second image running Fedora 10 to test compatibility with very old versions of Sugar that may still be in use in the field. I once set up a machine running Fedora 10 and deliberately avoided upgrading just so I could do this kind of testing. Now that I have the Mac Mini running Virtual Box I should be able to give that box an upgrade. Being able to run multiple versions of Linux on the same box easily is really the biggest advantage of the Virtual Box method.

Virtual Box will also run on Windows if you have a powerful enough machine.

As for the Macintosh, it should also be possible to install Fedora Linux on an Intel Macintosh as a dual boot, just like you do with Windows. Check the Fedora website for details.

What About Using sugar-build?

Sugar-build is a script that downloads the source code for the latest version of all the Sugar modules and compiles it into a subdirectory of your home directory. It doesn't actually install Sugar on your system. Instead, you run it out of the directory you installed it in. Because of the way it is built and run it doesn't interfere with the modules that make up your normal desktop. If you are developing Sugar itself, or if you are developing Activities that depend on the very latest Sugar features you'll need to run sugar-build.

Running this script is a bit more difficult than just installing the Sugar packages that come with the distribution. You'll need to install Git, run a Git command from the terminal to download the sugar-build scripts, then do a **make run** which will download more code, possibly ask you to install more packages, and ultimately compile everything and run it. When you're done you'll have an up to date test environment that you can run as an alternative to **sugar-emulator**. There is no need to uninstall sugar-emulator; both can coexist.

The documentation for sugar-build will be found here:

http://developer.sugarlabs.org/

Should you consider using it? The short answer is no. A longer answer is *probably not yet*.

If you want your Activities to reach the widest possible audience you *don't* want the latest Sugar. In fact, if you want a test environment that mimics what is on most XO computers right now you might need to use an older version of Fedora. Because updating operating systems in the field can be a major undertaking for a school many XO's will be not be running the very latest Sugar or even the one shipped with the latest Fedora for quite some time.

Strictly speaking sugar-build is just the script that downloads and compiles Sugar. If you wanted to be correct you would say "Run the copy of **sugar-runner** you made with sugar-build". However, recent Linux distributions like Fedora 20 come with the Sugar environment as something you can install like any other Linux package. There needs to be a way to distinguish between running what came with your distribution and what you download and compile yourself and run out of your home directory. To refer to the second option I'll say "use sugar-build" or "run sugar-build".

PYTHON

We'll be doing all the code samples in Python so you'll need to have Python installed. Python comes with every Linux distribution. You can download installers for Windows and the Macintosh at http://www.python.org/.

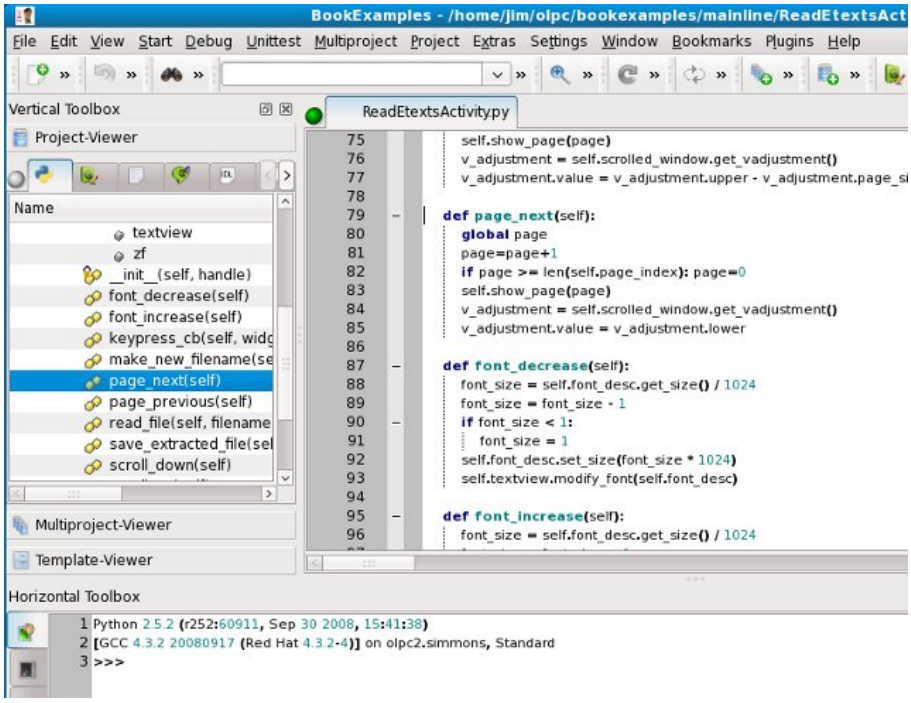
There are a couple of Python libraries you'll want to have as well: PyGTK and PyGame. These come with every Linux distribution. If you want to try them out in Windows or on the Macintosh you can get them here:

http://www.pygtk.org/

http://www.pygame.org/news.html

ERIC

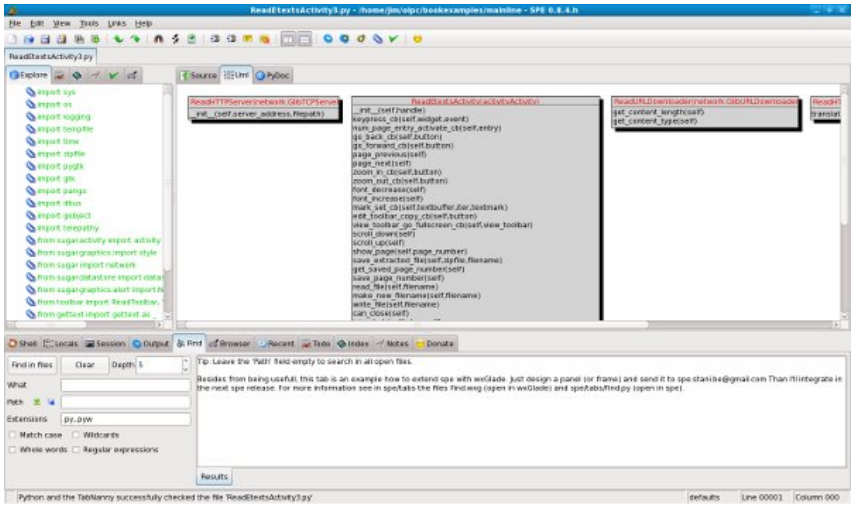
Developers today expect their languages to be supported by an **Integrated Development Environment** and Python is no exception. An **IDE** helps to organize your work and provides text editing and a built in set of programming and debugging tools.



There are two Python IDE's I have tried: Eric and Idle. Eric is the fancier of the two and I recommend it. Every Linux distribution should include it. It looks like it might work on Windows too. You can learn more about it at the Eric website: http://eric-ide.python-projects.org/.

SPE (STANI'S PYTHON EDITOR)

This is an IDE I discovered while writing this book. It comes with Fedora and in addition to being a Python editor it will make **UML** diagrams of your code and show **PyDoc** for it. Here is SPE showing a UML diagram for one of the Activities in this book:



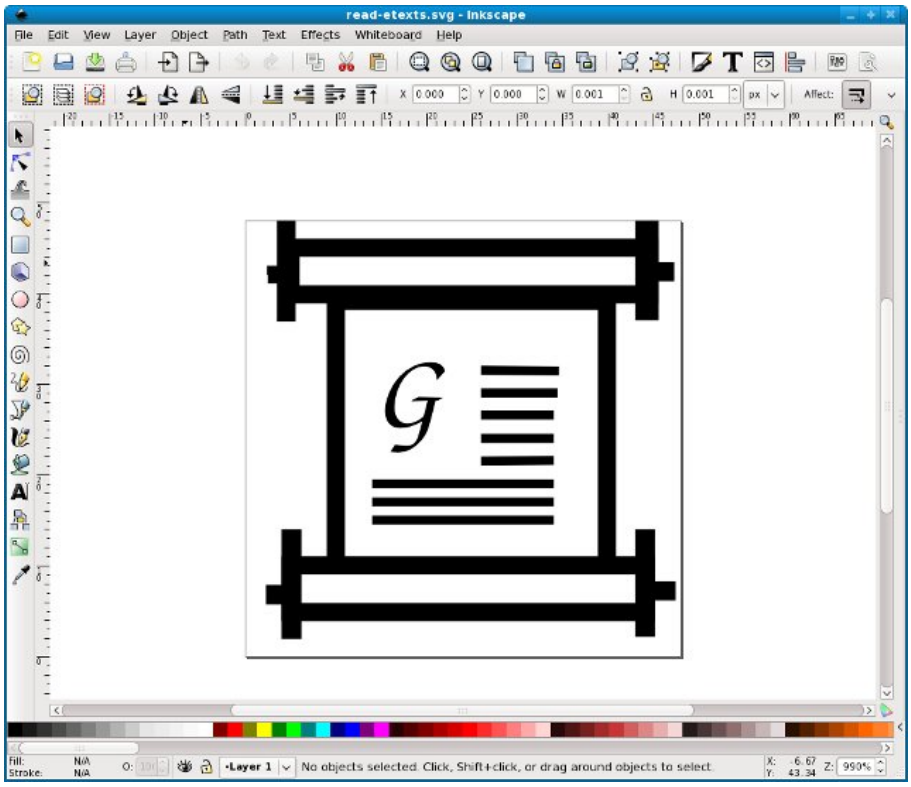
If you're an experienced developer you might find this a useful alternative to Eric. If you're just starting out Eric should meet your needs pretty well.

OTHER IDE'S

There is also a commercial Python IDE called Wingware, which has a version you can use for free. You can learn more about it at http://www.wingware.com/.

INKSCAPE

Inkscape is a tool for creating images in SVG format. Sugar uses SVG for Activity icons and other kinds of artwork. The "XO" icon that represents each child in the Neighborhood view is an SVG file that can be modified.



Inkscape comes with every Linux distribution, and can be installed on Windows as well. You can learn more about it here: http://www.inkscape.org/.

GIT

Git is a version control system. It stores versions of your program code in a way that makes them easy to get back. Whenever you make changes to your code you ask Git to store your code in its repository. If you need to look at an old version of that code later you can. Even better, if some problem shows up in your code you can compare your latest code to an old, working version and see exactly what lines you changed.

readtoolbar.py

```
15  15 # along with this program; if not, write to the Free Software
16  16 # Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
17  17
    18 import os
18  19 import logging
19  20 from gettext import gettext as _
20  21 import re
...
416 416         combotool.show()
417 417
418 418         self.pitchadj = gtk.Adjustment(0, -100, 100, 1, 10, 0)
419         -   self.pitchadj.connect("value_changed", self.pitch_adjusted_cb)
420 419         pitchbar = gtk.HScale(self.pitchadj)
421 420         pitchbar.set_draw_value(False)
422 421         pitchbar.set_update_policy(gtk.UPDATE_DISCONTINUOUS)
...
427 427         pitchbar.show()
428 428
429 429         self.rateadj = gtk.Adjustment(0, -100, 100, 1, 10, 0)
430         -   self.rateadj.connect("value_changed", self.rate_adjusted_cb)
431 430         ratebar = gtk.HScale(self.rateadj)
432 431         ratebar.set_draw_value(False)
433 432         ratebar.set_update_policy(gtk.UPDATE_DISCONTINUOUS)
...
453 453     def pitch_adjusted_cb(self, get):
454 454         speech.pitch = int(get.value)
455 455         speech.say(_("pitch adjusted"))
    456         f = open(os.path.join(self.activity.get_activity_root(), 'insta
```

If there are two people working on the same program independently a version control system will merge their changes together automatically.

Suppose you're working on a major new version of your Activity when someone finds a really embarrassing bug in the version you just released. If you use Git you don't need to tell people to live with it until the next release, which could be months away. Instead you can create a branch of the previous version and work on it alongside the version you're enhancing. In effect Git treats the old version you're fixing and the version you're improving as two separate projects.

You can learn more about Git at the Git website: http://git-scm.com/.

When you're ready for a Git repository for your project you can set one up here: http://git.sugarlabs.org/. I will have more to say about setting up and using a Git repository later in this book.

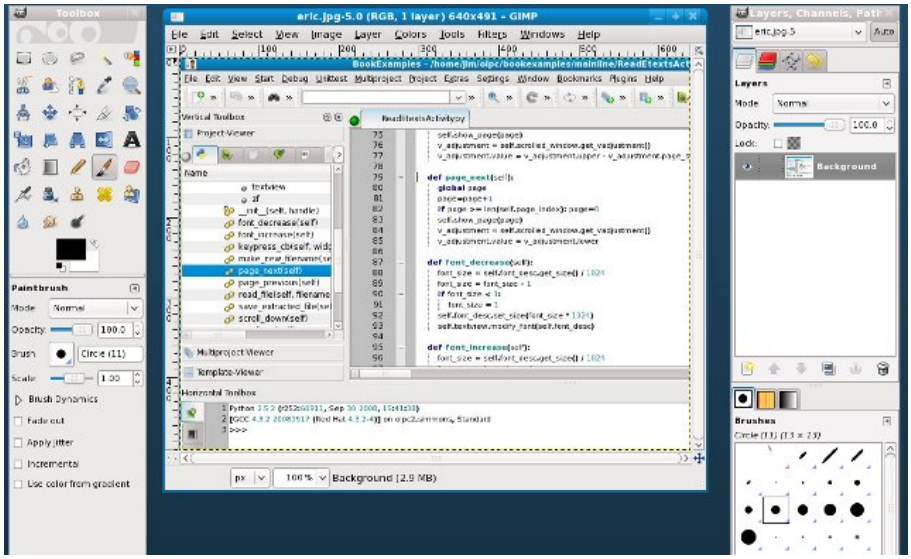
There is a Git repository containing all the code examples from this book. Once you have Git installed you can copy the repository to your computer with this command:

```
git clone git://git.sugarlabs.org/\
myo-sugar-activities-examples/mainline.git
```

This command should be typed all on one line. The backslash (\) character at the end of the first line is used in Linux to continue a long command to a second line. It is used here to make the command fit on the page of the printed version of this book. When you type in the command you can leave it out and type **myo-sugar-activities-examples/mainline.git** immediately following **git.sugarlabs.org/**.

THE GIMP

The GIMP is one of the most useful and badly named programs ever developed. You can think of it as a free version of Adobe Photoshop. If you need to work with image files (other than SVG's) you need this program.

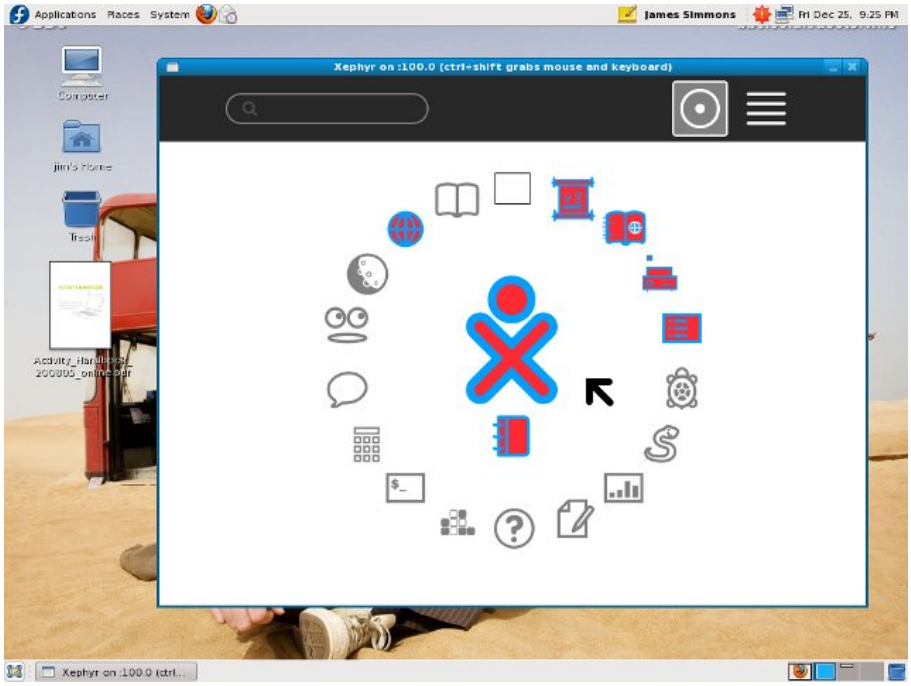


You may never need this program to develop the Activity itself, but when it's time to distribute the Activity you'll use it to create screen shots of your Activity in action. Nothing sells an Activity to a potential user like good screen shots.

SUGAR EMULATION

Most Linux distributions should have Sugar included. In Fedora you can run Sugar as an alternative desktop environment. When you log in to GDM Sugar appears as a desktop selection alongside GNOME, KDE, Window Maker, and any other window managers you have installed.

This is not the normal way to use Sugar for testing. The normal way uses a tool called Xephyr to run a Sugar environment in a window on your desktop. In effect, Xephyr runs an X session inside a window and Sugar runs in that. You can easily take screen shots of Sugar in action, stop and restart Sugar sessions without restarting the computer, and run multiple copies of Sugar to test collaboration.



I'll have more to say about this when it's time to test your first Activity.

6. CREATING YOUR FIRST SUGAR ACTIVITY

MAKE A STANDALONE PYTHON PROGRAM FIRST

The best advice I could give a beginning Activity developer is to make a version of your Activity that can run on its own, outside of the Sugar environment. Testing and debugging a Python program that stands alone is faster, easier and less tedious than doing the same thing with a similar Activity. You'll understand why when you start testing your first Activity.

The more bugs you find before you turn your code into an Activity the better. In fact, it's a good idea to keep a standalone version of your program around even after you have the Activity version well underway. I used my standalone version of **Read Etexts** to develop the text to speech with highlighting feature. This saved me a *lot* of time, which was especially important because I was figuring things out as I went.

Our first project will be a version of the Read Etexts Activity I wrote.

INHERIT FROM THE SUGAR.ACTIVITY.ACTIVITY CLASS

Next we're going to take our standalone Python program and make an Activity out of it. To do this we need to understand the concept of *inheritance*. In everyday speech inheritance means getting something from your parents that you didn't work for. A king will take his son to a castle window and say, "Someday, lad, this will all be yours!" That's inheritance.

In the world of computers programs can have parents and inherit things from them. Instead of inheriting property, they inherit code. There is a piece of Python code called sugar.activity.Activity that's the best parent an Activity could hope to have, and we're going to convince it to adopt our program. This doesn't mean that our program will never have to work again, but it won't have to work as much.

PACKAGE THE ACTIVITY

Now we have to package up our code to make it something that can be run under Sugar and distributed as an .xo file. This involves setting up a MANIFEST, activity.info, setup.py, and creating a suitable icon with Inkscape.

ADD REFINEMENTS

Every Activity will have the basic Activity toolbar. For most Activities this will not be enough, so we'll need to create some custom toolbars as well. Then we need to hook them up to the rest of the Activity code so that what happens to the toolbar triggers actions in the Activity and what happens outside the toolbar is reflected in the state of the toolbar.

In addition to toolbars we'll look at some other ways to spiff up your Activity.

PUT THE PROJECT CODE IN VERSION CONTROL

By this time we'll have enough code written that it's worth protecting and sharing with the world. To do that we need to create a Git repository and add our code to it. We'll also go over the basics of using Git.

GOING INTERNATIONAL WITH POOTLE

Now that our code is in Git we can request help from our first collaborator: the Pootle translation system. With a little setup work we can get volunteers to make translated versions of our Activity available.

DISTRIBUTING THE ACTIVITY

In this task we'll take our Activity and set it up on http://activities.sugarlabs.org plus we'll package up the source code so it can be included in Linux distributions.

ADD COLLABORATION

Next we'll add code to share e-books with Friends and the Neighborhood.

ADD TEXT TO SPEECH

Text to Speech with word highlighting is next. Our simple project will become a Kindle-killer!

7. A STANDALONE PYTHON PROGRAM FOR READING ETEXTS

THE PROGRAM

Our example program is based on the first Activity I wrote, **Read Etexts**. This is a program for reading free e-books.

The oldest and best source of free e-books is a website called *Project Gutenberg* (http://www.gutenberg.org/wiki/Main_Page). They create books in plain text format, in other words the kind of file you could make if you typed a book into Notepad and hit the Enter key at the end of each line. They have thousands of books that are out of copyright, including some of the best ever written. Before you read further go to that website and pick out a book that interests you. Check out the "Top 100" list to see the most popular books and authors.

The program we're going to create will read books in plain text format only.

There is a Git repository containing all the code examples in this book. Once you have Git installed you can copy the repository to your computer with this command:

```
git clone git://git.sugarlabs.org/\
myo-sugar-activities-examples/mainline.git
```

The code for our standalone Python program will be found in the directory **Make_Standalone_Python_gtk3** in a file named **ReadEtexts.py**. It looks like this:

```
#! /usr/bin/env python
#
# ReadEtexts.py Standalone version of ReadEtextsActivity.py
# Copyright (C) 2010  James D. Simmons
# Copyright (C) 2012  Aneesh Dogra <lionaneesh@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
along
# with this program; if not, write to the Free Software Foundation,
Inc.,
# 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
#

import sys
import os
import zipfile
from gi.repository import Gtk
from gi.repository import Gdk
import getopt
from gi.repository import Pango

page = 0
PAGE_SIZE = 45

class ReadEtexts():

    def keypress_cb(self, widget, event):
        "Respond when the user presses one of the arrow keys"
        keyname = Gdk.keyval_name(event.keyval)
```

```
        if keyname == 'plus':
            self.font_increase()
            return True
        if keyname == 'minus':
            self.font_decrease()
            return True
        if keyname == 'Page_Up' :
            self.page_previous()
            return True
        if keyname == 'Page_Down':
            self.page_next()
            return True
        if keyname == 'Up' or keyname == 'KP_Up' \
                or keyname == 'KP_Left':
            self.scroll_up()
            return True
        if keyname == 'Down' or keyname == 'KP_Down' \
                or keyname == 'KP_Right':
            self.scroll_down()
            return True
        return False

    def page_previous(self):
        global page
        page=page-1
        if page < 0: page=0
        self.show_page(page)
        v_adjustment = self.scrolled_window.get_vadjustment()
        v_adjustment.value = v_adjustment.get_upper() - \
            v_adjustment.get_page_size()

    def page_next(self):
        global page
        page=page+1
        if page >= len(self.page_index): page=0
        self.show_page(page)
        v_adjustment = self.scrolled_window.get_vadjustment()
        v_adjustment.value = v_adjustment.get_lower()

    def font_decrease(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size - 1
        if font_size < 1:
            font_size = 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def font_increase(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size + 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def scroll_down(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.value == v_adjustment.get_upper() - \
                v_adjustment.get_page_size():
            self.page_next()
            return
        if v_adjustment.value < v_adjustment.upper -
v_adjustment.page_size:
            new_value = v_adjustment.get_value() + \
                v_adjustment.get_step_increment()
            if new_value > v_adjustment.get_upper() -
v_adjustment.page_size:
                new_value = v_adjustment.upper -
v_adjustment.get_page_size()
            v_adjustment.set_value(new_value)

    def scroll_up(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.get_value() == v_adjustment.get_lower():
            self.page_previous()
            return
        if v_adjustment.get_value() > v_adjustment.get_lower():
            new_value = v_adjustment.get_value() - \
                v_adjustment.get_step_increment()
            if new_value < v_adjustment.get_lower():
                new_value = v_adjustment.get_lower()
            v_adjustment.set_value(new_value)

    def show_page(self, page_number):
        global PAGE_SIZE, current_word
        position = self.page_index[page_number]
        self.etext_file.seek(position)
        linecount = 0
        label_text = '\n\n\n'
        textbuffer = self.textview.get_buffer()
```

```
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line, 'iso-8859-1')
            linecount = linecount + 1
        label_text = label_text + '\n\n\n'
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        filebytes = zipfile.read(filename)
        f = open("/tmp/" + filename, 'w')
        try:
            f.write(filebytes)
        finally:
            f.close()

    def read_file(self, filename):
        "Read the Etext file"
        global PAGE_SIZE

        if zipfile.is_zipfile(filename):
            self.zf = zipfile.ZipFile(filename, 'r')
            self.book_files = self.zf.namelist()
            self.save_extracted_file(self.zf, self.book_files[0])
            currentFileName = "/tmp/" + self.book_files[0]
        else:
            currentFileName = filename

        self.etext_file = open(currentFileName,"r")
        self.page_index = [ 0 ]
        linecount = 0
        while self.etext_file:
            line = self.etext_file.readline()
            if not line:
                break
            linecount = linecount + 1
            if linecount >= PAGE_SIZE:
                position = self.etext_file.tell()
                self.page_index.append(position)
                linecount = 0
        if filename.endswith(".zip"):
            os.remove(currentFileName)

    def destroy_cb(self, widget, data=None):
        Gtk.main_quit()

    def main(self, file_path):
        self.window = Gtk.Window(Gtk.WindowType.TOPLEVEL)
        self.window.connect("destroy", self.destroy_cb)
        self.window.set_title("Read Etexts")
        self.window.set_size_request(640, 480)
        self.window.set_border_width(0)
        self.read_file(file_path)
        self.scrolled_window = Gtk.ScrolledWindow(hadjustment=None, \
                                                  vadjustment=None)
        self.textview = Gtk.TextView()
        self.textview.set_editable(False)
        self.textview.set_left_margin(50)
        self.textview.set_cursor_visible(False)
        self.textview.connect("key_press_event", self.keypress_cb)
        self.font_desc = Pango.FontDescription("sans 12")
        self.textview.modify_font(self.font_desc)
        self.show_page(0)
        self.scrolled_window.add(self.textview)
        self.window.add(self.scrolled_window)
        self.textview.show()
        self.scrolled_window.show()
        self.window.show()
        Gtk.main()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print "Usage: %s <file>" % (sys.argv[0])
        sys.exit(1)
    try:
        opts, args = getopt.getopt(sys.argv[1:], "")
        ReadEtexts().main(args[0])
    except getopt.error, msg:
        print msg
        print "This program has no options"
        sys.exit(2)
```

RUNNING THE PROGRAM

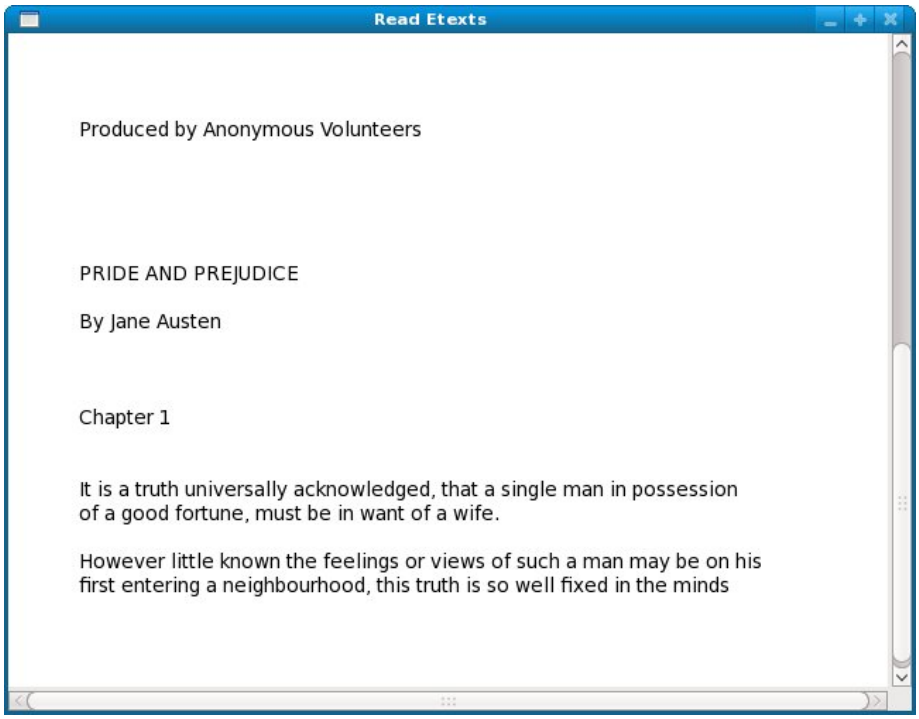
To run the program you should first make it executable. You only need to do this once:

```
chmod 755 ReadEtexts.py
```

For this example I downloaded the file for *Pride and Prejudice*. The program will work with either of the Plain text formats, which are either uncompressed text or a Zip file. The zip file is named **1342.zip**, and we can read the book by running this from a terminal:

```
./ReadEtexts.py 1342.zip
```

This is what the program looks like in action:



You can use the *Page Up, Page Down, Up, Down, Left*, and *Right* keys to navigate through the book and the '+' and '-' keys to adjust the font size.

HOW THE PROGRAM WORKS

This program reads through the text file containing the book and divides it into pages of 45 lines each. We need to do this because the **gtk.TextView** component we use for viewing the text would need a lot of memory to scroll through the whole book and that would hurt performance. A second reason is that we want to make reading the e-book as much as possible like reading a regular book, and regular books have pages. If a teacher assigns reading from a book she might say "read pages 35-50 for tommorow". Finally, we want this program to remember what page you stopped reading on and bring you back to that page again when you read the book next time. (The program we have so far doesn't do that yet).

To page through the book we use **random access** to read the file. To understand what random access means to a file, consider a VHS tape and a DVD. To get to a certain scene in a VHS tape you need to go through all the scenes that came before it, in order. Even though you do it at high speed you still have to look at all of them to find the place you want to start watching. This is **sequential access**. On the other hand a DVD has chapter stops and possibly a chapter menu. Using a chapter menu you can look at any scene in the movie right away, and you can skip around as you like. This is random access, and the chapter menu is like an **index**. Of course you can access the material in a DVD sequentially too.

We need random access to skip to whatever page we like, and we need an index so that we know where each page begins. We make the index by reading the entire file one line at a time. Every 45 lines we make a note of how many characters into the file we've gotten and store this information in a Python list. Then we go back to the beginning of the file and display the first page. When the program user goes to the next or previous page we figure out what the new page number will be and look in the list entry for that page. This tells us that page starts 4,200 characters into the file. We use seek() on the file to go to that character and then we read 45 lines starting at that point and load them into the TextView.

When you run this program notice how fast it is. Python programs take longer to run a line of code than a compiled language would, but in this program it doesn't matter because the heavy lifting in the program is done by the TextView, which was created in a compiled language. The Python parts don't do that much so the program doesn't spend much time running them.

Sugar uses Python a lot, not just for Activities but for the Sugar environment itself. You may read somewhere that using so much Python is "a disaster" for performance. Don't believe it.

There are no slow programming languages, only slow programmers.

8. INHERIT FROM SUGAR3.ACTIVITY.ACTIVITY

OBJECT ORIENTED PYTHON

Python supports two styles of programming: **procedural** and **object oriented**. Procedural programming is when you have some input data, do some processing on it, and produce an output. If you want to calculate all the prime numbers under a hundred or convert a Word document into a plain text file you'll probably use the procedural style to do that.

Object oriented programs are built up from units called **objects**. An object is described as a collection of fields or attributes containing data along with methods for doing things with that data. In addition to doing work and storing data objects can send messages to one another.

Consider a word processing program. It doesn't have just one input, some process, and one output. It can receive input from the keyboard, from the mouse buttons, from the mouse traveling over something, from the clipboard, etc. It can send output to the screen, to a file, to a printer, to the clipboard, etc. A word processor can edit several documents at the same time too. Any program with a GUI is a natural fit for the object oriented style of programming.

Objects are described by *classes*. When you create an object you are creating an *instance* of a class.

There's one other thing that a class can do, which is to **inherit** methods and attributes from another class. When you define a class you can say it **extends** some class, and by doing that in effect your class has the functionality of the other class plus its own functionality. The extended class becomes its parent.

All Sugar Activities extend a Python class called **sugar3.activity.Activity**. This class provides methods that all Activities need. In addition to that, there are methods that you can override in your own class that the parent class will call when it needs to. For the beginning Activity writer three methods are important:

__init__()

This is called when your Activity is started up. This is where you will set up the user interface for your Activity, including toolbars.

read_file(self, file_path)

This is called when you resume an Activity from a Journal entry. It is called after the *__init__()* method is called. The file_path parameter contains the name of a temporary file that is a copy of the file in the Journal entry. The file is deleted as soon as this method finishes, but because Sugar runs on Linux if you open the file for reading your program can continue to read it even after it is deleted and it the file will not actually go away until you close it.

write_file(self, file_path)

This is called when the Activity updates the Journal entry. Just like with *read_file()* your Activity does not work with the Journal directly. Instead it opens the file named in file_path for output and writes to it. That file in turn is copied to the Journal entry.

There are three things that can cause *write_file()* to be executed:

- Your Activity closes.
- Someone presses the **Keep** button in the Activity toolbar.
- Your Activity ceases to be the active Activity, or someone moves from the Activity View to some other View.

In addition to updating the file in the Journal entry the *read_file()* and *write_file()* methods are used to read and update the metadata in the Journal entry.

When we convert our standalone Python program to an Activity we'll take out much of the code we wrote and replace it with code inherited from the sugar.activity.Activity class.

EXTENDING THE ACTIVITY CLASS

Here's a version of our program that extends Activity. You'll find it in the Git repository in the directory **Inherit_From_sugar.activity.Activity_gtk3** under the name **ReadEtextsActivity.py**:

```
import os
import zipfile
from gi.repository import Gtk
from gi.repository import Gdk
from gi.repository import Pango
from sugar3.activity import widgets
from sugar3.activity.widgets import StopButton
from sugar3.activity import activity
from sugar3.graphics import style

page=0
PAGE_SIZE = 45

class ReadEtextsActivity(activity.Activity):
    def __init__(self, handle):
        "The entry point to the Activity"
        global page
        activity.Activity.__init__(self, handle)

        toolbox = widgets.ActivityToolbar(self)
        toolbox.share.props.visible = False

        stop_button = StopButton(self)
        stop_button.show()
        toolbox.insert(stop_button, -1)

        self.set_toolbar_box(toolbox)

        toolbox.show()
        self.scrolled_window = Gtk.ScrolledWindow()
        self.scrolled_window.set_policy(Gtk.PolicyType.NEVER, \
            Gtk.PolicyType.AUTOMATIC)

        self.textview = Gtk.TextView()
        self.textview.set_editable(False)
        self.textview.set_cursor_visible(False)
        self.textview.set_left_margin(50)
        self.textview.connect("key_press_event", self.keypress_cb)

        self.scrolled_window.add(self.textview)
        self.set_canvas(self.scrolled_window)
        self.textview.show()
        self.scrolled_window.show()
        page = 0
        self.textview.grab_focus()
        self.font_desc = Pango.FontDescription("sans %d" %
style.zoom(10))
        self.textview.modify_font(self.font_desc)

    def keypress_cb(self, widget, event):
```

```
    "Respond when the user presses one of the arrow keys"
    keyname = Gdk.keyval_name(event.keyval)
    print keyname
    if keyname == 'plus':
        self.font_increase()
        return True
    if keyname == 'minus':
        self.font_decrease()
        return True
    if keyname == 'Page_Up' :
        self.page_previous()
        return True
    if keyname == 'Page_Down':
        self.page_next()
        return True
    if keyname == 'Up' or keyname == 'KP_Up' \
            or keyname == 'KP_Left':
        self.scroll_up()
        return True
    if keyname == 'Down' or keyname == 'KP_Down' \
            or keyname == 'KP_Right':
        self.scroll_down()
        return True
    return False

def page_previous(self):
    global page
    page=page-1
    if page < 0: page=0
    self.show_page(page)
    v_adjustment = self.scrolled_window.get_vadjustment()
    v_adjustment.set_value(v_adjustment.get_upper() - \
        v_adjustment.get_page_size())

def page_next(self):
    global page
    page=page+1
    if page >= len(self.page_index): page=0
    self.show_page(page)
    v_adjustment = self.scrolled_window.get_vadjustment()
    v_adjustment.set_value(v_adjustment.get_lower())

def font_decrease(self):
    font_size = self.font_desc.get_size() / 1024
    font_size = font_size - 1
    if font_size < 1:
        font_size = 1
    self.font_desc.set_size(font_size * 1024)
    self.textview.modify_font(self.font_desc)

def font_increase(self):
    font_size = self.font_desc.get_size() / 1024
    font_size = font_size + 1
    self.font_desc.set_size(font_size * 1024)
    self.textview.modify_font(self.font_desc)

def scroll_down(self):
    v_adjustment = self.scrolled_window.get_vadjustment()
    if v_adjustment.get_value() == v_adjustment.get_upper() - \
            v_adjustment.get_page_size():
        self.page_next()
        return
    if v_adjustment.get_value() < v_adjustment.get_upper() - \
        v_adjustment.get_page_size():
        new_value = v_adjustment.get_value() + \
            v_adjustment.step_increment
        if new_value > v_adjustment.get_upper() - \
            v_adjustment.get_page_size():
            new_value = v_adjustment.get_upper() - \
                v_adjustment.get_page_size()
        v_adjustment.set_value(new_value)

def scroll_up(self):
    v_adjustment = self.scrolled_window.get_vadjustment()
    if v_adjustment.get_value() == v_adjustment.get_lower():
        self.page_previous()
        return
    if v_adjustment.get_value() > v_adjustment.get_lower():
        new_value = v_adjustment.get_value() - \
            v_adjustment.step_increment
        if new_value < v_adjustment.get_lower():
            new_value = v_adjustment.get_lower()
        v_adjustment.set_value(new_value)

def show_page(self, page_number):
    global PAGE_SIZE, current_word
    position = self.page_index[page_number]
    self.etext_file.seek(position)
```

```
        linecount = 0
        label_text = '\n\n\n'
        textbuffer = self.textview.get_buffer()
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line, 'iso-8859-1')
            linecount = linecount + 1
        label_text = label_text + '\n\n\n'
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        filebytes = zipfile.read(filename)
        outfn = self.make_new_filename(filename)
        if (outfn == ''):
            return False
        f = open(os.path.join(self.get_activity_root(), 'tmp',  outfn),
'w')
        try:
            f.write(filebytes)
        finally:
            f.close()

    def read_file(self, filename):
        "Read the Etext file"
        global PAGE_SIZE

        if zipfile.is_zipfile(filename):
            self.zf = zipfile.ZipFile(filename, 'r')
            self.book_files = self.zf.namelist()
            self.save_extracted_file(self.zf, self.book_files[0])
            currentFileName = os.path.join(self.get_activity_root(),
'tmp', \
                self.book_files[0])
        else:
            currentFileName = filename

        self.etext_file = open(currentFileName,"r")
        self.page_index = [ 0 ]
        linecount = 0
        while self.etext_file:
            line = self.etext_file.readline()
            if not line:
                break
            linecount = linecount + 1
            if linecount >= PAGE_SIZE:
                position = self.etext_file.tell()
                self.page_index.append(position)
                linecount = 0
        if filename.endswith(".zip"):
            os.remove(currentFileName)
        self.show_page(0)

    def make_new_filename(self, filename):
        partition_tuple = filename.rpartition('/')
        return partition_tuple[2]
```

This program has some significant differences from the standalone version. First, note that this line:

```
#! /usr/bin/env python
```

has been removed. We are no longer running the program directly from the Python interpreter. Now Sugar is running it as an Activity. Notice that much (but not all) of what was in the main() method has been moved to the *__init__()* method and the *main()* method has been removed.

Notice too that the *class* statement has changed:

```
class ReadEtextsActivity(activity.Activity)
```

This statement now tells us that class ReadEtextsActivity extends the class **sugar3.activity.Activity**. As a result it inherits the code that is in that class. Therefore we no longer need a GTK main loop, or to define a window. The code in this class we extend will do that for us.

While we gain much from this inheritance, we lose something too: a title bar for the main window. In a graphical operating environment a piece of software called a *window manager* is responsible for putting borders on windows, making them resizeable, reducing them to icons, maximizing them, etc. Sugar uses a window manager named Matchbox which makes each window fill the whole screen and puts no border, title bar, or any other window decorations on the windows. As a result of that we can't close our application by clicking on the "X" in the title bar as before. To make up for this we need to have a toolbar that contains a Close button. Thus every Activity has an Activity toolbar that contains some standard controls and buttons. If you look at the code you'll see I'm hiding a couple of controls which we have no use for yet.

The *read_file()* method is no longer called from the main() method and doesn't seem to be called from anywhere in the program. Of course it does get called, by some of the Activity code we inherited from our new parent class. Similarly the *__init__()* and *write_file()* methods (if we had a *write_file()* method) get called by the parent Activity class.

If you're especially observant you might have noticed another change. Our original standalone program created a temporary file when it needed to extract something from a Zip file. It put that file in a directory called /tmp. Our new Activity still creates the file but puts it in a different directory, one specific to the Activity.

All writing to the file system is restricted to subdirectories of the path given by *self.get_activity_root()*. This method will give you a directory that belongs to your Activity alone. It will contain three subdirectories with different policies:

data
: This directory is used for data such as configuration files. Files stored here will survive reboots and OS upgrades.

tmp
: This directory is used similar to the /tmp directory, being backed by RAM. It may be as small as 1 MB. This directory is deleted when the activity exits.

instance
: This directory is similar to the **tmp** directory, being backed by the computer's drive rather than by RAM. It is unique per instance. It is used for transfer to and from the Journal. This directory is deleted when the activity exits.

Making these changes to the code is not enough to make our program an Activity. We have to do some packaging work and get it set up to run from the Sugar emulator. We also need to learn how to run the Sugar emulator. That comes next!

9. PACKAGE THE ACTIVITY

ADD SETUP.PY

You'll need to add a Python program called **setup.py** to the same directory that you Activity program is in. Every setup.py is exactly the same as every other setup.py. The copies in our Git repository look like this:

```
#!/usr/bin/env python

# Copyright (C) 2006, Red Hat, Inc.
#
# This program is free software; you can redistribute it
# and/or modify it under the terms of the GNU General
# Public License as published by the Free Software
# Foundation; either version 2 of the License, or (at
# your option) any later version.
#
# This program is distributed in the hope that it will
# be useful, but WITHOUT ANY WARRANTY; without even
# the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General
# Public License for more details.
#
# You should have received a copy of the GNU General
# Public License along with this program; if not,
# write to the Free Software Foundation, Inc.,
# 51 Franklin St, Fifth Floor, Boston, MA
# 02110-1301  USA

from sugar3.activity import bundlebuilder

bundlebuilder.start()
```

Be sure and copy the entire text above, including the comments.

The setup.py program is used by sugar for a number of purposes. If you run setup.py from the command line you'll see the options that are used with it and what they do.

```
[james@olpc mainline]$ ./setup.py
Available commands:

build                Build generated files
dev                  Setup for development
dist_xo              Create a xo bundle package
dist_source          Create a tar source package
fix_manifest         Add missing files to the manifest (OBSOLETE)
genpot               Generate the gettext pot file
install              Install the activity in the system

(Type "./setup.py <command> --help" for help about a particular
command's
options.
```

We'll be running some of these commands later on. Don't be concerned about the fix_manifest command being obsolete. Earlier versions of the bundle builder used a file named MANIFEST and the current one doesn't need it. It isn't anything you need to worry about.

CREATE ACTIVITY.INFO

Next create a directory within the one your progam is in and name it **activity**. Create a file named **activity.info** within that directory and enter the lines below into it. Here is the one for our first Activity:

```
[Activity]
name = Read ETexts II
bundle_id = net.flossmanuals.ReadEtextsActivity
icon = read-etexts
exec = sugar-activity ReadEtextsActivity.ReadEtextsActivity
```

```
show_launcher = no
activity_version = 1
mime_types = text/plain;application/zip
license = GPLv2+
summary = Read plain text ebooks from Project Gutenberg.
```

This file tells Sugar how to run your Activity. The properties needed in this file are:

name	The name of your Activity as it will appear to the user.
bundle_id	A unique name that Sugar will use to refer to your Activity. Any Journal entry created by your Activity will have this name stored in its metadata, so that when someone resumes the Journal entry Sugar knows to use the program that created it to read it. This property used to be called service_name and you may run into old code that still uses that name. If you do, you'll need to change it to bundle_id.
icon	The name of the icon file you have created for the Activity. Since icons are always .svg files the icon file in the example is named read-etexts.svg.
exec	This tells Sugar how to launch your Activity. What it says is to create an instance of the class **ReadEtextsActivity** which it will find in file **ReadEtextsActivity.py**.
show_launcher	There are two ways to launch an Activity. The first is to click on the icon in the Activity view. The second is to resume an entry in the Journal. Activities that don't create Journal entries can only be resumed from the Journal, so there is no point in putting an icon in the Activity ring for them. Read Etexts is an Activity like that.
activity_version	An integer that represents the version number of your program. The first version is 1, the next is 2, and so on.
mime_types	Generally when you resume a Journal entry it launches the Activity that created it. In the case of an e-book it wasn't created by any Activity, so we need another way to tell the Journal which Activity it can use. A MIME type is the name of a common file format. Some examples are text/plain, text/html, application/zip and application/pdf. In this entry we're telling the Journal that our program can handle either plain text files or Zip archive files.
license	Owning a computer program is not like buying a car. With a car, you're the owner and you can do what you like with it. You can sell it, rent it out, make it into a hot rod, whatever. With a computer program there is always a license that tells the person receiving the program what he is allowed to do with it. GPLv2+ is a popular standard license that can be used for Activities, and since this is *my* program that is what goes here. When you're ready to distribute one of *your* Activities I'll have more to say about licenses.
summary	A description of what the Activity does.

CREATE AN ICON

Next we need to create an icon named **read-etexts.svg** and put it in the **activity** subdirectory. ⁞ We're going to use Inkscape to create the icon. From the **New** menu in Inkscape select **icon_48x48**. This will create a drawing area that is a good size.

You don't need to be an expert in Inkscape to create an icon. In fact the less fancy your icon is the better. When drawing your icon remember the following points:

- Your icon needs to look good in sizes ranging from really, really small to large.
- It needs to be recognizable when its really, really small.
- You only get to use two colors: a stroke color and a fill color. It doesn't matter which ones you choose because Sugar will need to override your choices anyway, so just use black strokes on a white background.
- A fill color is only applied to an area that is contained within an unbroken stroke. If you draw a box and one of the corners doesn't quite connect the area inside that box will not be filled. Free hand drawing is only for the talented. Circles, boxes, and arcs are easy to draw with Inkscape so use them when you can.
- Inkscape will also draw 3D boxes using two point perspective. Don't use them. Icons should be flat images. 3D just doesn't look good in an icon.
- Coming up with good ideas for icons is tough. I once came up with a rather nice picture of a library card catalog drawer for **Get Internet Archive Books**. The problem is, no child under the age of forty has ever seen a card catalog and fewer still understand its purpose.

When you're done making your icon you need to modify it so it can work with Sugar. Specifically, you need to make it show Sugar can use its own choice of stroke color and fill color. The SVG file format is based on XML, which means it is a text file with some special tags in it. This means that once we have finished editing it in Inkscape we can load the file into Eric and edit it as a text file.

I'm not going to put the entire file in this chapter because most of it you'll just leave alone. The first part you need to modify is at the very beginning.

Before:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->
<svg
```

After:

```
<?xml version="1.0" ?>
<!DOCTYPE svg  PUBLIC '-//W3C//DTD SVG 1.1//EN'
  'http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd' [
 <!ENTITY stroke_color "#000000">
 <!ENTITY fill_color "#FFFFFF">
]><svg
```

Now in the body of the document you'll find references to *fill* and *stroke* as part of an attribute called *style*. Every line or shape you draw will have these, like this:

```
<rect
   style="fill:#ffffff;stroke:#000000;stroke-opacity:1"
   id="rect904"
   width="36.142857"
   height="32.142857"
   x="4.1428571"
```

```
   y="7.1428571" />
```

You need to change each one to look like this:

```
<rect
   style="fill:&fill_color;;stroke:&stroke_color;
;stroke-opacity:1"
   id="rect904"
   width="36.142857"
   height="32.142857"
   x="4.1428571"
   y="7.1428571" />
```

Note that *&stroke_color;* and *&fill_color;* both end with semicolons (;), and semicolons are also used to separate the properties for style. Because of this it is an extremely common beginner's mistake to leave off the trailing semicolon because two semicolons in a row don't look right. Be assured that the two semicolons in a row are intentional and absolutely necessary! Second, the value for style should all go *on one line*. We split it here to make it fit on the printed page; do not split it in your own icon!

INSTALL THE ACTIVITY

There's just one more thing to do before we can test our Activity under the Sugar emulator. We need to install it, which in this case means making a symbolic link between the directory we're using for our code in the ~/Activities/ directory. The symbol ~ refers to the "home" directory of the user we're running Sugar under, and a symbolic link is a way to make a file or directory appear to be located in more than one place without copying it. We make this symbolic link by running setup.py again:

```
./setup.py dev
```

RUNNING OUR ACTIVITY

Now at last we can run our Activity under Sugar. To do that we need to learn how to run **sugar-emulator** or **sugar-runner**. sugar-runner replaces sugar-emulator and is available in Fedora 20 or later. Everything before that uses sugar-emulator.

Fedora puts a Sugar menu item under the Education menu in Xfce or GNOME 2. You can also run it from the command line:

```
sugar-emulator
```

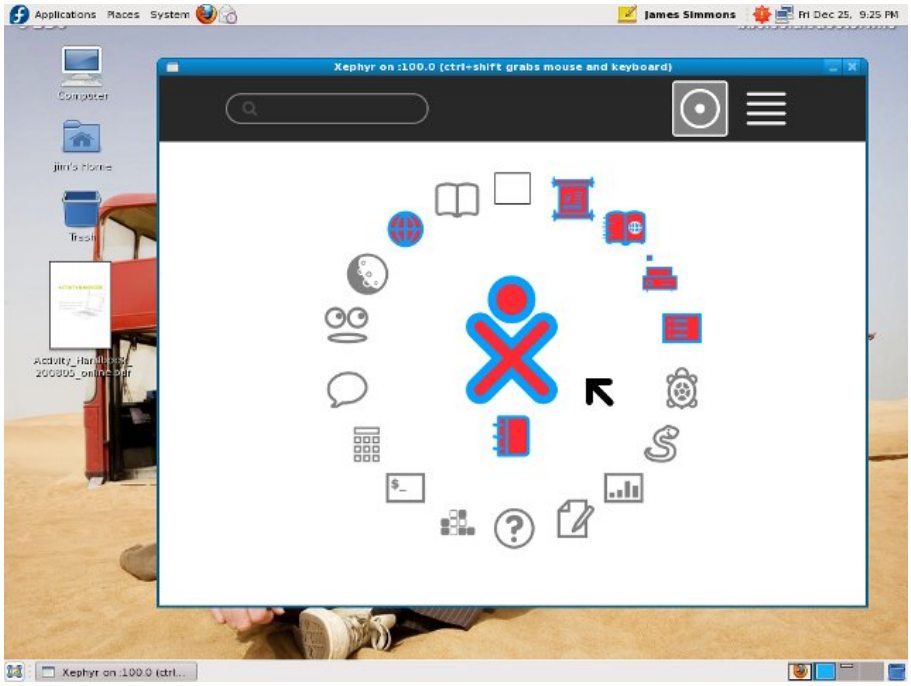
If your screen resolution is smaller than the default size sugar-emulator runs at it will run full screen. This is not convenient for testing, so you may want to specify your own size:

```
sugar-emulator -i 800x600
```

sugar-runner is similar, but it defaults to running in full screen mode if you run it from the menu or without parameters. To get it to run in a window you need to specify a --resolution parameter:

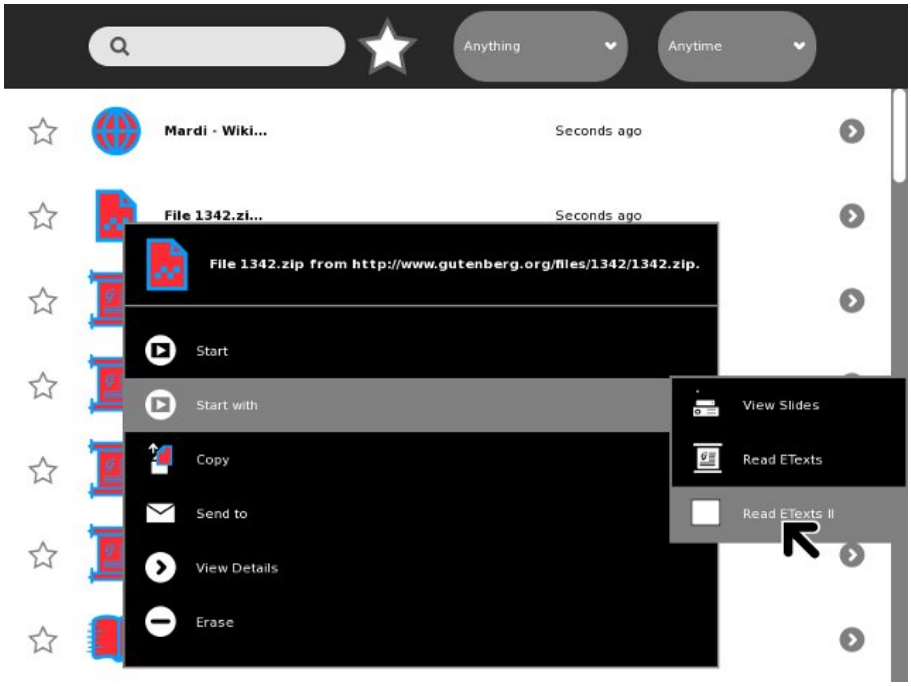
```
sugar-runner --resolution 800x600
```

When you run sugar-emulator or sugar-runner a window opens up and the Sugar environment starts up and runs inside it. It looks like this:

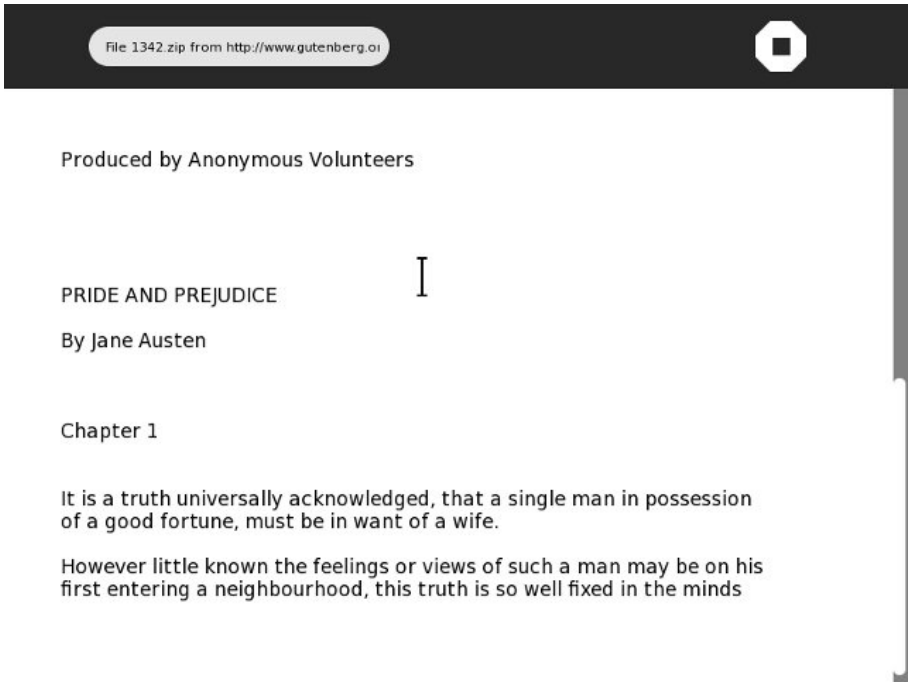


To test our Activity we're going to need to have a book in the Journal, so use the **Browse** Activity to visit Project Gutenberg again and download the book of your choice. This time it's important to download the book in Zip format, because Browse cannot download a plain text file to the Journal. Instead, it opens the file for viewing as if it was a web page. If you try the same thing with the Zip file it will create an entry in the Journal.

We can't just open the file with one click in the Journal because our program did not create the Journal entry and there are several Activities that support the MIME type of the Journal entry. We need to use the Start With menu option like this:



When we do open the Journal entry this is what we see:



Technically, this is the first **iteration** of our Activity. (Iteration is a vastly useful word meaning something you do more than once. In this book we're building our Activity a bit at a time so I can demonstrate Activity writing principles, but actually building a program in pieces, testing it, getting feedback, and building a bit more can be a highly productive way of creating software. Using the word *iteration* to describe each step in the process makes the process sound more formal than it really is).

While this Activity might be good enough to show your own mother, we really should improve it a bit before we do that. That part comes next.

10. ADD REFINEMENTS

TOOLBARS

It is a truth universally acknowledged that a first rate Activity needs good Toolbars. In this chapter we'll learn how to make them. We put the toolbar classes in a separate file from the rest. Originally this was because there used to be two styles of toolbar (old and new) and I wanted to support both in the Activity. With Sugar 3 there is only one style of toolbar so I could have put everything in one file.

The toolbar code is in a file called **toolbar.py** in the **Add_Refinements_gtk** directory of the Git repository. It looks like this:

```
from gettext import gettext as _
import re

from gi.repository import Gtk
from gi.repository import GObject

from sugar3.graphics.toolbutton import ToolButton

class ViewToolbar(Gtk.Toolbar):
    __gtype_name__ = 'ViewToolbar'

    __gsignals__ = {
        'needs-update-size': (GObject.SIGNAL_RUN_FIRST,
                              GObject.TYPE_NONE,
                              ([])),
        'go-fullscreen': (GObject.SIGNAL_RUN_FIRST,
                          GObject.TYPE_NONE,
                          ([]))
    }

    def __init__(self):
        Gtk.Toolbar.__init__(self)
        self.zoom_out = ToolButton('zoom-out')
        self.zoom_out.set_tooltip(_('Zoom out'))
        self.insert(self.zoom_out, -1)
        self.zoom_out.show()

        self.zoom_in = ToolButton('zoom-in')
        self.zoom_in.set_tooltip(_('Zoom in'))
        self.insert(self.zoom_in, -1)
        self.zoom_in.show()

        spacer = Gtk.SeparatorToolItem()
        spacer.props.draw = False
        self.insert(spacer, -1)
        spacer.show()

        self.fullscreen = ToolButton('view-fullscreen')
        self.fullscreen.set_tooltip(_('Fullscreen'))
        self.fullscreen.connect('clicked', self.fullscreen_cb)
        self.insert(self.fullscreen, -1)
        self.fullscreen.show()

    def fullscreen_cb(self, button):
        self.emit('go-fullscreen')
```

Another file in the same directory of the Git repository is named **ReadEtextsActivity2.py**. It looks like this:

```
import os
import zipfile
import re
from gi.repository import Gtk
from gi.repository import Gdk
from gi.repository import Pango
from sugar3.activity import activity
from sugar3.graphics import style
from sugar3.graphics.toolbutton import ToolButton
from sugar3.graphics.toolbarbox import ToolbarButton
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3.activity.widgets import StopButton
```

```
from sugar3.activity.widgets import EditToolbar
from sugar3.activity.widgets import ActivityToolbar
from sugar3.activity.widgets import _create_activity_icon
from toolbar import ViewToolbar
from gettext import gettext as _

page = 0
PAGE_SIZE = 45
TOOLBAR_READ = 2

class CustomActivityToolbarButton(ToolbarButton):
    """
        Custom Activity Toolbar button, adds the functionality to
disable or
        enable the share button.
    """
    def __init__(self, activity, shared=False, **kwargs):
        toolbar = ActivityToolbar(activity, orientation_left=True)

        if not shared:
            toolbar.share.props.visible = False

        ToolbarButton.__init__(self, page=toolbar, **kwargs)

        icon = _create_activity_icon(activity.metadata)
        self.set_icon_widget(icon)
        icon.show()

class ReadEtextsActivity(activity.Activity):
    def __init__(self, handle):
        "The entry point to the Activity"
        global page
        activity.Activity.__init__(self, handle)

        toolbar_box = ToolbarBox()

        activity_button = CustomActivityToolbarButton(self)
        toolbar_box.toolbar.insert(activity_button, 0)
        activity_button.show()

        self.edit_toolbar = EditToolbar()
        self.edit_toolbar.undo.props.visible = False
        self.edit_toolbar.redo.props.visible = False
        self.edit_toolbar.separator.props.visible = False
        self.edit_toolbar.copy.set_sensitive(False)
        self.edit_toolbar.copy.connect('clicked',
self.edit_toolbar_copy_cb)
        self.edit_toolbar.paste.props.visible = False
        edit_toolbar_button = ToolbarButton(
            page=self.edit_toolbar,
            icon_name='toolbar-edit')
        self.edit_toolbar.show()
        toolbar_box.toolbar.insert(edit_toolbar_button, -1)
        edit_toolbar_button.show()

        view_toolbar = ViewToolbar()
        view_toolbar.connect('go-fullscreen',
                self.view_toolbar_go_fullscreen_cb)
        view_toolbar.zoom_in.connect('clicked', self.zoom_in_cb)
        view_toolbar.zoom_out.connect('clicked', self.zoom_out_cb)
        view_toolbar.show()
        view_toolbar_button = ToolbarButton(
            page=view_toolbar,
            icon_name='toolbar-view')
        toolbar_box.toolbar.insert(view_toolbar_button, -1)
        view_toolbar_button.show()

        self.back = ToolButton('go-previous')
        self.back.set_tooltip(_('Back'))
        self.back.props.sensitive = False
        self.back.connect('clicked', self.go_back_cb)
        toolbar_box.toolbar.insert(self.back, -1)
        self.back.show()

        self.forward = ToolButton('go-next')
        self.forward.set_tooltip(_('Forward'))
        self.forward.props.sensitive = False
        self.forward.connect('clicked', self.go_forward_cb)
        toolbar_box.toolbar.insert(self.forward, -1)
        self.forward.show()

        num_page_item = Gtk.ToolItem()
        self.num_page_entry = Gtk.Entry()
        self.num_page_entry.set_text('0')
        self.num_page_entry.set_alignment(1)
        self.num_page_entry.connect('insert-text',
                                    self.num_page_entry_insert_text_cb)
        self.num_page_entry.connect('activate',
```

```
                                self.num_page_entry_activate_cb)
        self.num_page_entry.set_width_chars(4)
        num_page_item.add(self.num_page_entry)
        self.num_page_entry.show()
        toolbar_box.toolbar.insert(num_page_item, -1)
        num_page_item.show()

        total_page_item = Gtk.ToolItem()
        self.total_page_label = Gtk.Label()

        self.total_page_label.set_markup("<span foreground='#FFF'" \
                                         " size='14000'></span>")

        self.total_page_label.set_text(' / 0')
        total_page_item.add(self.total_page_label)
        self.total_page_label.show()
        toolbar_box.toolbar.insert(total_page_item, -1)
        total_page_item.show()

        separator = Gtk.SeparatorToolItem()
        separator.props.draw = False
        separator.set_expand(True)
        toolbar_box.toolbar.insert(separator, -1)
        separator.show()

        stop_button = StopButton(self)
        stop_button.props.accelerator = '<Ctrl><Shift>Q'
        toolbar_box.toolbar.insert(stop_button, -1)
        stop_button.show()

        self.set_toolbar_box(toolbar_box)
        toolbar_box.show()

        self.scrolled_window = Gtk.ScrolledWindow()
        self.scrolled_window.set_policy(Gtk.PolicyType.NEVER,
            Gtk.PolicyType.AUTOMATIC)

        self.textview = Gtk.TextView()
        self.textview.set_editable(False)
        self.textview.set_cursor_visible(False)
        self.textview.set_left_margin(50)
        self.textview.connect("key_press_event", self.keypress_cb)

        self.scrolled_window.add(self.textview)
        self.set_canvas(self.scrolled_window)
        self.textview.show()
        self.scrolled_window.show()
        page = 0
        self.clipboard = Gtk.Clipboard.get(Gdk.SELECTION_CLIPBOARD)
        self.textview.grab_focus()
        self.font_desc = Pango.FontDescription("sans %d" %
style.zoom(10))
        self.textview.modify_font(self.font_desc)

        buffer = self.textview.get_buffer()
        self.markset_id = buffer.connect("mark-set",
            self.mark_set_cb)

    def num_page_entry_insert_text_cb(self, entry, text, length,
position):
        if not re.match('[0-9]', text):
            entry.emit_stop_by_name('insert-text')
            return True
        return False

    def num_page_entry_activate_cb(self, entry):
        global page
        if entry.props.text:
            new_page = int(entry.props.text) - 1
        else:
            new_page = 0

        if new_page >= self.total_pages:
            new_page = self.total_pages - 1
        elif new_page < 0:
            new_page = 0

        self.current_page = new_page
        self.set_current_page(new_page)
        self.show_page(new_page)
        entry.props.text = str(new_page + 1)
        self.update_nav_buttons()
        page = new_page

    def update_nav_buttons(self):
        current_page = self.current_page
        self.back.props.sensitive = current_page > 0
        self.forward.props.sensitive = \
```

```
        current_page < self.total_pages - 1

    self.num_page_entry.props.text = str(current_page + 1)
    self.total_page_label.props.label = \
        ' / ' + str(self.total_pages)

def set_total_pages(self, pages):
    self.total_pages = pages

def set_current_page(self, page):
    self.current_page = page
    self.update_nav_buttons()

def keypress_cb(self, widget, event):
    "Respond when the user presses one of the arrow keys"
    keyname = Gdk.keyval_name(event.keyval)
    print keyname
    if keyname == 'plus':
        self.font_increase()
        return True
    if keyname == 'minus':
        self.font_decrease()
        return True
    if keyname == 'Page_Up' :
        self.page_previous()
        return True
    if keyname == 'Page_Down':
        self.page_next()
        return True
    if keyname == 'Up' or keyname == 'KP_Up' \
            or keyname == 'KP_Left':
        self.scroll_up()
        return True
    if keyname == 'Down' or keyname == 'KP_Down' \
            or keyname == 'KP_Right':
        self.scroll_down()
        return True
    return False

def go_back_cb(self, button):
    self.page_previous()

def go_forward_cb(self, button):
    self.page_next()

def page_previous(self):
    global page
    page=page - 1
    if page < 0: page = 0
    self.set_current_page(page)
    self.show_page(page)
    v_adjustment = self.scrolled_window.get_vadjustment()
    v_adjustment.set_value(v_adjustment.get_upper() - \
                           v_adjustment.get_page_size())

def page_next(self):
    global page
    page = page + 1
    if page >= len(self.page_index): page = 0
    self.set_current_page(page)
    self.show_page(page)
    v_adjustment = self.scrolled_window.get_vadjustment()
    v_adjustment.set_value(v_adjustment.get_lower())

def zoom_in_cb(self,  button):
    self.font_increase()

def zoom_out_cb(self,  button):
    self.font_decrease()

def font_decrease(self):
    font_size = self.font_desc.get_size() / 1024
    font_size = font_size - 1
    if font_size < 1:
        font_size = 1
    self.font_desc.set_size(font_size * 1024)
    self.textview.modify_font(self.font_desc)

def font_increase(self):
    font_size = self.font_desc.get_size() / 1024
    font_size = font_size + 1
    self.font_desc.set_size(font_size * 1024)
    self.textview.modify_font(self.font_desc)

def mark_set_cb(self, textbuffer, iter, textmark):

    if textbuffer.get_has_selection():
        self.edit_toolbar.copy.set_sensitive(True)
```

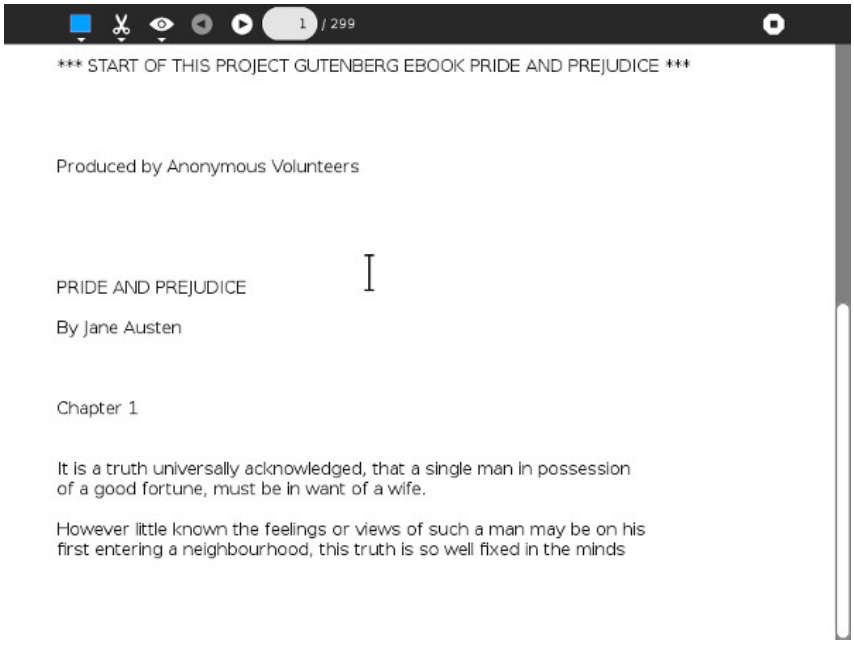
```
        else:
            self.edit_toolbar.copy.set_sensitive(False)

    def edit_toolbar_copy_cb(self, button):
        textbuffer = self.textview.get_buffer()
        textbuffer.copy_clipboard(self.clipboard)

    def view_toolbar_go_fullscreen_cb(self, view_toolbar):
        self.fullscreen()

    def scroll_down(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.get_value() == v_adjustment.get_upper() - \
                v_adjustment.get_page_size():
            self.page_next()
            return
        if v_adjustment.get_value() < v_adjustment.get_upper() - \
                v_adjustment.get_page_size():
            new_value = v_adjustment.get_value() + \
                v_adjustment.step_increment
            if new_value > v_adjustment.get_upper() \
                - v_adjustment.get_page_size():
                new_value = v_adjustment.get_upper() \
                    - v_adjustment.get_page_size()
            v_adjustment.set_value(new_value)

    def scroll_up(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.get_value() == v_adjustment.get_lower():
            self.page_previous()
            return
        if v_adjustment.get_value() > v_adjustment.get_lower():
            new_value = v_adjustment.get_value() - \
                v_adjustment.step_increment
            if new_value < v_adjustment.get_lower():
                new_value = v_adjustment.get_lower()
            v_adjustment.set_value(new_value)

    def show_page(self, page_number):
        global PAGE_SIZE, current_word
        position = self.page_index[page_number]
        self.etext_file.seek(position)
        linecount = 0
        label_text = '\n\n\n'
        textbuffer = self.textview.get_buffer()
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line, 'iso-8859-1')
            linecount = linecount + 1
        label_text = label_text + '\n\n\n'
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        filebytes = zipfile.read(filename)
        outfn = self.make_new_filename(filename)
        if (outfn == ''):
            return False
        f = open(os.path.join(self.get_activity_root(), 'tmp',  outfn),
'w')
        try:
            f.write(filebytes)
        finally:
            f.close()

    def get_saved_page_number(self):
        global page
        title = self.metadata.get('title', '')
        if title == ''  or not title[len(title)- 1].isdigit():
            page = 0
        else:
            i = len(title) - 1
            newPage = ''
            while (title[i].isdigit() and i > 0):
                newPage = title[i] + newPage
                i = i - 1
            if title[i] == 'P':
                page = int(newPage) - 1
            else:
                # not a page number; maybe a volume number.
                page = 0

    def save_page_number(self):
        global page
        title = self.metadata.get('title', '')
        if title == ''  or not title[len(title)- 1].isdigit():
            title = title + ' P' +  str(page + 1)
```

```
        else:
            i = len(title) - 1
            while (title[i].isdigit() and i > 0):
                i = i - 1
            if title[i] == 'P':
                title = title[0:i] + 'P' + str(page + 1)
            else:
                title = title + ' P' + str(page + 1)
        self.metadata['title'] = title

    def read_file(self, filename):
        "Read the Etext file"
        global PAGE_SIZE,  page

        if zipfile.is_zipfile(filename):
            self.zf = zipfile.ZipFile(filename, 'r')
            self.book_files = self.zf.namelist()
            self.save_extracted_file(self.zf, self.book_files[0])
            currentFileName = os.path.join(self.get_activity_root(), \
                    'tmp',  self.book_files[0])
        else:
            currentFileName = filename

        self.etext_file = open(currentFileName,"r")
        self.page_index = [ 0 ]
        pagecount = 0
        linecount = 0
        while self.etext_file:
            line = self.etext_file.readline()
            if not line:
                break
            linecount = linecount + 1
            if linecount >= PAGE_SIZE:
                position = self.etext_file.tell()
                self.page_index.append(position)
                linecount = 0
                pagecount = pagecount + 1
        if filename.endswith(".zip"):
            os.remove(currentFileName)
        self.get_saved_page_number()
        self.show_page(page)
        self.set_total_pages(pagecount + 1)
        self.set_current_page(page)

    def make_new_filename(self, filename):
        partition_tuple = filename.rpartition('/')
        return partition_tuple[2]

    def write_file(self, filename):
        "Save meta data for the file."
        self.metadata['activity'] = self.get_bundle_id()
        self.save_page_number()
```

This is the **activity.info** for this example:

```
[Activity]
name = ReadEtexts II
bundle_id = net.flossmanuals.ReadETextsActivity2
icon = read-etexts
exec = sugar-activity ReadEtextsActivity2.ReadEtextsActivity
show_launcher = no
mime_types = text/plain;application/zip
activity_version = 1
license = GPLv2+
summary = Example of adding a toolbar to your application.
```

When we run this new version this is what we'll see:



There are a few things worth pointing out in this code. First, have a look at this import:

```
from gettext import gettext as _
```

We'll be using the *gettext* module of Python to support translating our Activity into other languages. We'll be using it in statements like this one:

```
        self.back.set_tooltip(_('Back'))
```

The underscore acts the same way as the gettext function because of the way we imported gettext. The effect of this statement will be to look in a special translation file for a word or phrase that matches the key "Back" and replace it with its translation. If there is no translation file for the language we want then it will simply use the word "Back". We'll explore setting up these translation files later, but for now using gettext for all of the words and phrases we will show to our Activity users lays some important groundwork.

The second thing worth pointing out is that while our revised Activity has three toolbars we only had to create one of them. The other two, **Activity** and **Edit**, are part of the Sugar Python library. We can use those toolbars as is, hide the controls we don't need, or even extend them by adding new controls. In the example we're hiding the **Share** control of the Activity toolbar and the **Undo**, **Redo**, and **Paste** buttons of the Edit toolbar. We currently do not support sharing books or modifying the text in books so these controls are not needed.

Another thing to notice is that the Activity class doesn't just provide us with a window. The window has a VBox to hold our toolbar box and the body of our Activity. We install the toolbox using *set_toolbar_box()* and the body of the Activity using *set_canvas()*.

The **Read** and **View** toolbars are regular PyGtk programming, but notice that there is a special button for Sugar toolbars that can have a tooltip attached to it, plus the **View** toolbar has code to hide the toolbox and **ReadEtextsActivity2** has code to unhide it. This is an easy function to add to your own Activities and many games and other kinds of Activities can benefit from the increased screen area you get when you hide the toolbox.

METADATA AND JOURNAL ENTRIES

Every Journal entry represents a single file plus **metadata**, or information describing the file. There are standard metadata entries that all Journal entries have and you can also create your own custom metadata.

Unlike ReadEtextsActivity, this version has a *write_file()* method.

```
def write_file(self, filename):
    "Save meta data for the file."
    self.metadata['activity'] = self.get_bundle_id()
    self.save_page_number()
```

We didn't have a *write_file()* method before because we weren't going to update the file the book is in, and we still aren't. We will, however, be updating the metadata for the Journal entry. Specifically, we'll be doing two things:

- Save the page number our Activity user stopped reading on so when he launches the Activity again we can return to that page.
- Tell the Journal entry that it belongs to our Activity, so that in the future it will use our Activity's icon and can launch our Activity with one click.

The way the **Read** Activity saves page number is to use a custom metadata property.

```
self.metadata['Read_current_page'] = \
    str(self._document.get_page_cache().get_current_page())
```

Read creates a custom metadata property named *Read_current_page* to store the current page number. You can create any number of custom metadata properties just this easily, so you may wonder why we aren't doing that with **Read Etexts**. Actually, the first version of **Read Etexts** did use a custom property, but in Sugar .82 or lower there was a bug in the Journal such that custom metadata did not survive after the computer was turned off. As a result my Activity would remember pages numbers while the computer was running, but would forget them as soon as it was shut down. This has not been a problem with Sugar for quite some time, but it was a real problem when the first edition of the book came out.

You might find how I got around this problem instructive. I created the following two methods:

```
def get_saved_page_number(self):
    global page
    title = self.metadata.get('title', '')
    if title == '' or not title[len(title)-1].isdigit():
        page = 0
    else:
        i = len(title) - 1
        newPage = ''
        while (title[i].isdigit() and i > 0):
            newPage = title[i] + newPage
            i = i - 1
        if title[i] == 'P':
            page = int(newPage) - 1
        else:
            # not a page number; maybe a volume number.
            page = 0
```

```
    def save_page_number(self):
        global page
        title = self.metadata.get('title', '')
        if title == '' or not title[len(title)-1].isdigit():
            title = title + ' P' +  str(page + 1)
        else:
            i = len(title) - 1
            while (title[i].isdigit() and i > 0):
                i = i - 1
            if title[i] == 'P':
                title = title[0:i] + 'P' + str(page + 1)
            else:
                title = title + ' P' + str(page + 1)
        self.metadata['title'] = title
```

save_page_number() looks at the current title metadata and either adds a page number to the end of it or updates the page number already there. Since title is standard metadata for all Journal entries the Journal bug does not affect it.

These examples show how to read metadata too.

```
        title = self.metadata.get('title', '')
```

This line of code says "Get the metadata property named *title* and put it in the variable named *title*, If there is no title property put an empty string in *title*.

Generally you will save metadata in the *write_file()* method and read it in the *read_file()* method.

In a normal Activity that writes out a file in write_file() this next line would be unnecessary:

```
        self.metadata['activity'] = self.get_bundle_id()
```

Any Journal entry created by an Activity will automatically have this property set. In the case of *Pride and Prejudice*, our Activity did not create it. We are able to read it because our Activity supports its *MIME type*. Unfortunately, that MIME type, *application/zip*, is used by other Activities. I found it very frustrating to want to open a book in **Read Etexts** and accidentally have it opened in **EToys** instead. This line of code solves that problem. You only need to use *Start Using...* the first time you read a book. After that the book will use the **Read Etexts** icon and can be resumed with a single click.

This does not at all affect the MIME type of the Journal entry, so if you wanted to deliberately open *Pride and Prejudice* with **Etoys** it is still possible.

Before we leave the subject of Journal metadata let's look at all the standard metadata that every Activity has. Here is some code that creates a new Journal entry and updates a bunch of standard properties:

```
    def create_journal_entry(self,  tempfile):
        journal_entry = datastore.create()
        journal_title = self.selected_title
        if self.selected_volume != '':
            journal_title +=  ' ' + _('Volume') + ' ' + \
                self.selected_volume
        if self.selected_author != '':
            journal_title = journal_title  + ', by ' + \
                self.selected_author
        journal_entry.metadata['title'] = journal_title
        journal_entry.metadata['title_set_by_user'] = '1'
        journal_entry.metadata['keep'] = '0'
        format = \
            self._books_toolbar.format_combo.props.value
        if format == '.djvu':
            journal_entry.metadata['mime_type'] = \
                'image/vnd.djvu'
        if format == '.pdf' or format == '_bw.pdf':
            journal_entry.metadata['mime_type'] = \
```

```
            'application/pdf'
        journal_entry.metadata['buddies'] = ''
        journal_entry.metadata['preview'] = ''
        journal_entry.metadata['icon-color'] = \
            profile.get_color().to_string()
        textbuffer = self.textview.get_buffer()
        journal_entry.metadata['description'] = \
            textbuffer.get_text(textbuffer.get_start_iter(),
            textbuffer.get_end_iter())
        journal_entry.file_path = tempfile
        datastore.write(journal_entry)
        os.remove(tempfile)
        self._alert(_('Success'), self.selected_title + \
            _(' added to Journal.'))
```

This code is taken from an Activity I wrote that downloads books from a website and creates Journal entries for them. The Journal entries contain a friendly title and a full description of the book.

Most Activities will only deal with one Journal entry by using the *read_file()* and *write_file()* methods but you are not limited to that. In a later chapter I'll show you how to create and delete Journal entries, how to list the contents of the Journal, and more.

We've covered a lot of technical information in this chapter and there's more to come, but before we get to that we need to look at some other important topics:

- Putting your Activity in version control. This will enable you to share your code with the world and get other people to help work on it.
- Getting your Activity translated into other languages.
- Distributing your finished Activity. (Or your not quite finished but still useful Activity).

11. ADD YOUR ACTIVITY CODE TO VERSION CONTROL

WHAT IS VERSION CONTROL?

"If I have seen further it is only by standing on the shoulders of giants."

Isaac Newton, in a letter to Robert Hooke.

Writing an Activity is usually not something you do by yourself. You will usually have collaborators in one form or another. When I started writing **Read Etexts** I copied much of the code from the **Read** Activity. When I implemented text to speech I adapted a toolbar from the **Speak** Activity. When I finally got my copied file sharing code working the author of **Image Viewer** thought it was good enough to copy into that Activity. Another programmer saw the work I did for text to speech and thought he could do it better. He was right, and his improvements got merged into my own code. When I wrote **Get Internet Archive Books** someone else took the user interface I came up with and made a more powerful and versatile Activity called **Get Books**. Like Newton, everyone benefits from the work others have done before.

Even if I wanted to write Activities without help I would still need collaborators to translate them into other languages.

To make collaboration possible you need to have a place where everyone can post their code and share it. This is called a code repository. It isn't enough to just share the latest version of your code. What you really want to do is share *every* version of your code. Every time you make a significant change to your code you want to have the new version and the previous version available. Not only do you want to have every version of your code available, you want to be able to compare any two versions your code to see what changed between them. This is what version control software does.

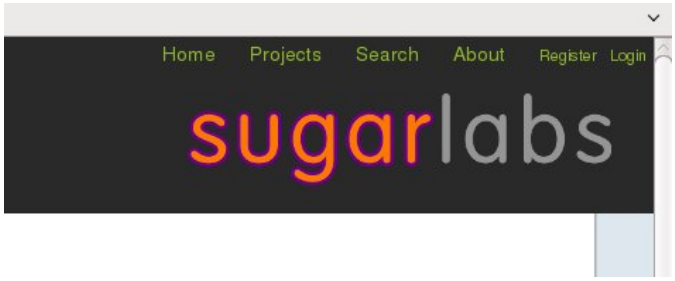
The three most popular version control tools are **CVS**, **Subversion**, and **Git**. Git is the newest and is the one used by Sugar Labs. While not every Activity has its code in the Sugar Labs Git repository (other free code repositories exist) there is no good reason not to do it and significant benefits if you do. If you want to get your Activity translated into other languages using the Sugar Labs Git repository is a must.

GIT ALONG LITTLE DOGIES

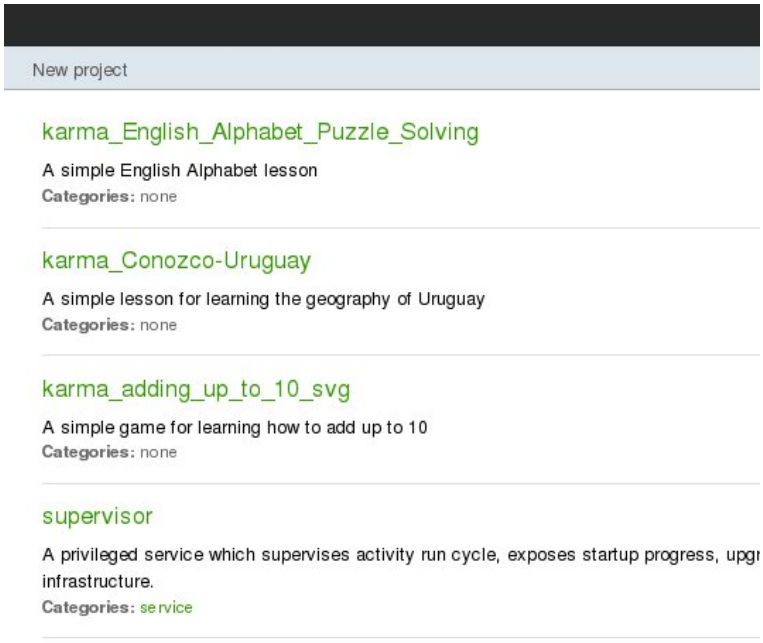
Git is a **distributed** version control system. This means that not only are there copies of every version of your code in a central repository, the same copies exist on every user's computer. This means you can update your local repository while you are not connected to the Internet, then connect and share everything at one time.

There are two ways you will interact with your Git repository: through Git commands and through the website at http://git.sugarlabs.org/. We'll look at this website first.

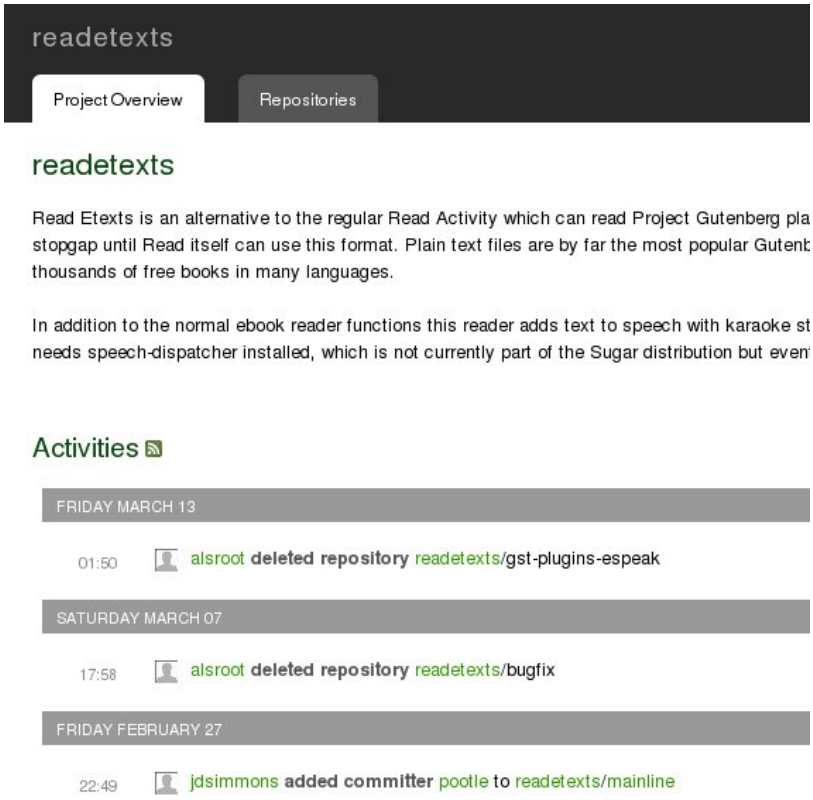
Go to http://git.sugarlabs.org/ and click on the **Projects** link in the upper right corner:



You will see a list of projects in the repository. They will be listed from newest to oldest. You'll also see a **New Project** link but you'll need to create an account to use that and we aren't ready to do that yet.

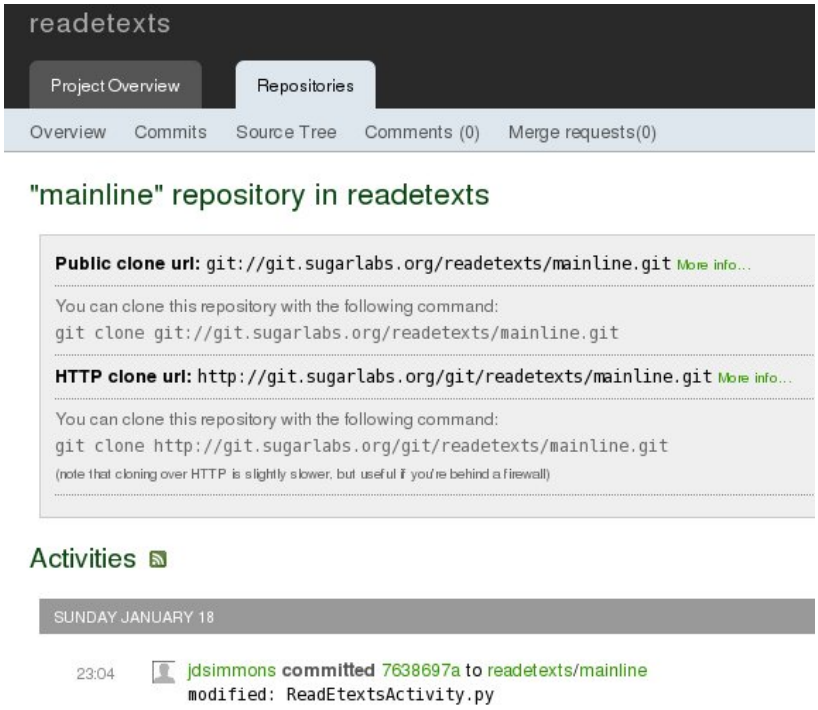


If you use the **Search** link in the upper right corner of the page you'll get a search form. Use it to search for "read etexts". Click on the link for that project when you find it. You should see something like this:



This page lists *some* of the activity for the project but I don't find it particularly useful. To get a much better look at your project start by clicking on the repository name on the right side of the page. In this case the repository is named **mainline**.

You'll see something like this at the top of the page:



This page has some useful information on it. First, have a look at the **Public clone url** and the **HTTP clone url**. You need to click on **More info...** to see either one. If you run either of these commands from the console you will get a copy of the git repository for the project copied to your computer. This copy will include every version of every piece of code in the project. You would need to modify it a bit before you could share your changes back to the main repository, but everything would be there.

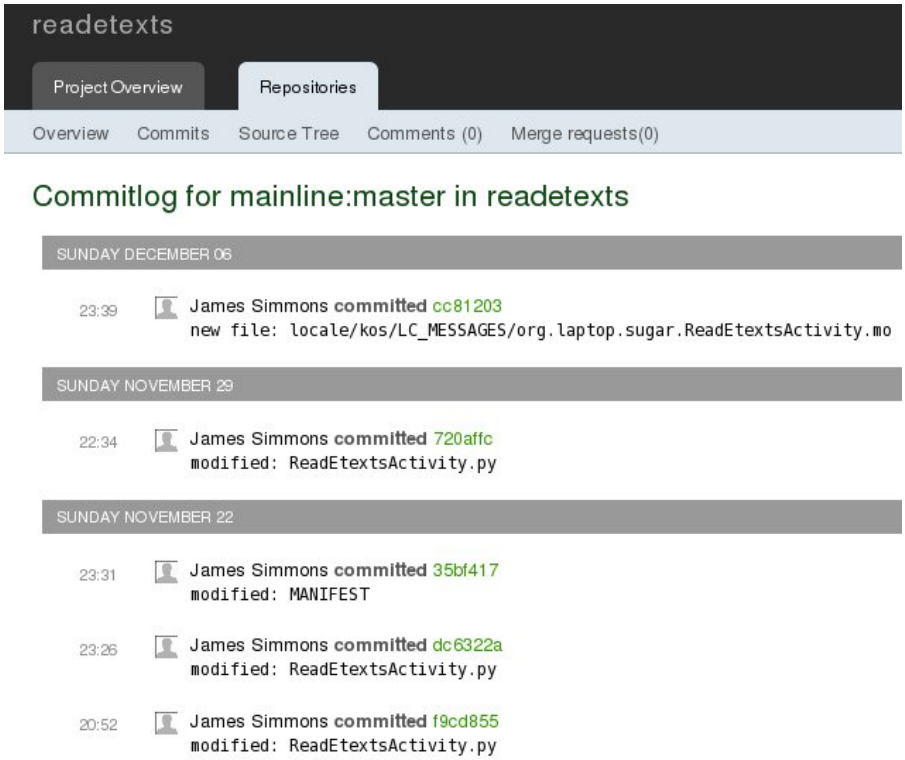
The list under **Activities** is not that useful, but if you click on the **Source Tree** link you'll see something really good:

Here is a list of every file in the project, the date it was last updated, and a comment on what was modified. Click on the link for **ReadEtextsActivity.py** and you'll see this:

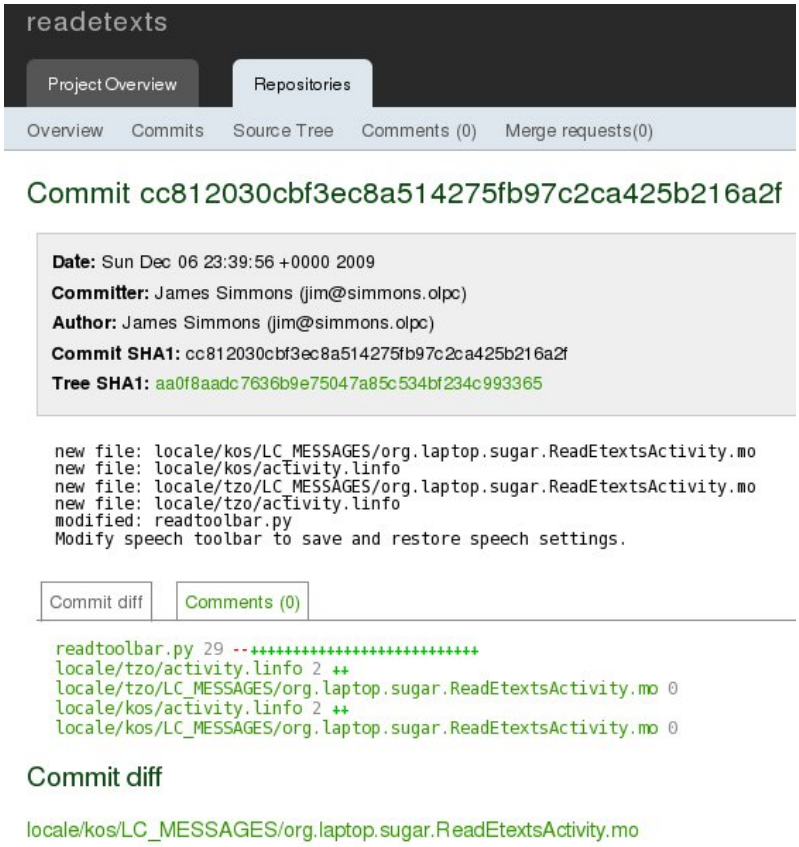
This is the latest code in that file in pretty print format. Python keywords are shown in a different color, there are line numbers, etc. This is a good page for looking at code on the screen, but it doesn't print well and it's not much good for copying snippets of code into Eric windows either. For either of those things you'll want to click on **raw blob data** at the top of the listing:

```
#! /usr/bin/env python

# Copyright (C) 2008, 2009 James D. Simmons
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
import os
import logging
import tempfile
import time
import zipfile
import pygtk
pygtk.require('2.0')
import gtk
import string
from sugar.graphics import style
from sugar import profile
from sugar.activity import activity
from sugar import network
from sugar.datastore import datastore
from sugar.graphics.alert import NotifyAlert
```

We're not done yet. Use the **Back** button to get back to the pretty print listing and click on the **Commits** link. This will give us a list of everything that changed each time we committed code into Git:



You may have noticed the odd combination of letters and numbers after the words **James Simmons committed**. This is a kind of version number. The usual practice with version control systems is to give each version of code you check in a version number, usually a simple sequence number. Git is distributed, with many separate copies of the repository being modified independently and then merged. That makes using just a sequential number to identify versions unworkable. Instead, Git gives each version a really, really large random number. The number is expressed in base 16, which uses the symbols 0-9 and a-f. What you see in green is only a small part of the complete number. The number is a link, and if you click on it you'll see this:



At the top of the page we see the complete version number used for this commit. Below the gray box we see the full comment that was used to commit the changes. Below that is a listing of what files were changed. If we look further down the page we see this:

readtoolbar.py

```
 15  15 # along with this program; if not, write to the Free Software
 16  16 # Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
 17  17
     18 import os
 18  19 import logging
 19  20 from gettext import gettext as _
 20  21 import re
...  ...
416 416         combotool.show()
417 417
418 418         self.pitchadj = gtk.Adjustment(0, -100, 100, 1, 10, 0)
419         -   self.pitchadj.connect("value_changed", self.pitch_adjusted_cb)
420 419         pitchbar = gtk.HScale(self.pitchadj)
421 420         pitchbar.set_draw_value(False)
422 421         pitchbar.set_update_policy(gtk.UPDATE_DISCONTINUOUS)
...  ...
427 427         pitchbar.show()
428 428
429 429         self.rateadj = gtk.Adjustment(0, -100, 100, 1, 10, 0)
430         -   self.rateadj.connect("value_changed", self.rate_adjusted_cb)
431 430         ratebar = gtk.HScale(self.rateadj)
432 431         ratebar.set_draw_value(False)
433 432         ratebar.set_update_policy(gtk.UPDATE_DISCONTINUOUS)
...  ...
453 453     def pitch_adjusted_cb(self, get):
454 454         speech.pitch = int(get.value)
455 455         speech.say(_("pitch adjusted"))
     456         f = open(os.path.join(self.activity.get_activity_root(), 'insta
```

This is a *diff* report which shows the lines that have changed between this version and the previous version. For each change it shows a few lines before and after the change to give you a better idea of what the change does. Every change shows line numbers too.

A report like this is a wonderful aid to programming. Sometimes when you're working on an enhancement to your program something that had been working mysteriously stops working. When that happens you will wonder just what you changed that could have caused the problem. A diff report can help you find the source of the problem.

By now you must be convinced that you want your project code in Git. Before we can do that we need to create an account on this website. That is no more difficult than creating an account on any other website, but it will need an important piece of information from us that we don't have yet. Getting that information is our next task.

SETTING UP SSH KEYS

To send your code to the **Gitorious** code repository you need an SSH public/private key pair. ⁞ SSH is a way of sending data over the network in encrypted format. (In other words, it uses a secret code so nobody but the person getting the data can read it). Public/private key encryption is a way of encrypting data that provides a way to guarantee that the person who is sending you the data is who he claims to be.

In simple terms it works like this: the SSH software generates two very large numbers that are used to encode and decode the data going over the network. The first number, called the **private key**, is kept secret and is only used by you to encode the data. The second number, called the **public key**, is given to anyone who needs to decode your data. He can decode it using the public key; there is no need for him to know the private key. He can also use the public key to encode a message to send back to you and you can decode it using your private key.

Git uses SSH like an electronic signature to verify that code changes that are supposed to be coming from you actually are coming from you. The Git repository is given your public key. It knows that anything it decodes with that key must have been sent by you because only you have the private key needed to encode it.

We will be using a tool called **OpenSSH** to generate the public and private keys. This is included with every version of Linux so you just need to verify that it has been installed. Then use the **ssh-keygen** utility that comes with OpenSSH to generate the keys:

```
[jim@olpc2 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jim/.ssh/id_rsa):
```

By default ssh-keygen generates an **RSA** key, which is the kind we want. By default it puts the keyfiles in a directory called ***/yourhome/*.ssh** and we want that too, so DO NOT enter a filename when it asks you to. Just hit the **Enter** key to continue.

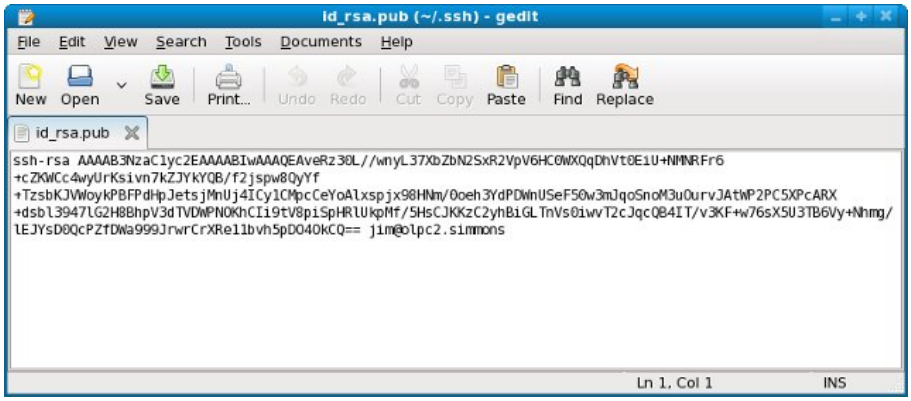
```
[jim@olpc2 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jim/.ssh/id_rsa):
Created directory '/home/jim/.ssh'.
Enter passphrase (empty for no passphrase):
```

Now we DO want a **passphrase** here. A passphrase is like a password that is used with the public and private keys to do the encrypting. When you type it in you will not be able to see what you typed. Because of that it will ask you to type the same thing again, and it will check to see that you typed them in the same way both times.

```
[jim@olpc2 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jim/.ssh/id_rsa):
Created directory '/home/jim/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jim/.ssh/id_rsa.
Your public key has been saved in /home/jim/.ssh/id_rsa.pub.
The key fingerprint is:
d0:fe:c0:0c:1e:72:56:7a:19:cd:f3:85:c7:4c:9e:18
jim@olpc2.simmons
The key's randomart image is:
+--[ RSA 2048]----+
|        oo  E=.  |
|       + o+ .+=. |
|    . B +  o.oo  |
|     = O    .    |
|      . S        |
|         o       |
|          .      |
|                 |
|                 |
+-----------------+
```

When choosing a passphrase remember that it needs to be something you can type reliably without seeing it and it would be better if it was *not* a word you can find in the dictionary, because those are easily broken. When I need to make a password I use the tool at http://www.multicians.org/thvv/gpw.html. This tool generates a bunch of nonsense words that are pronounceable. Pick one that appeals to you and use that.

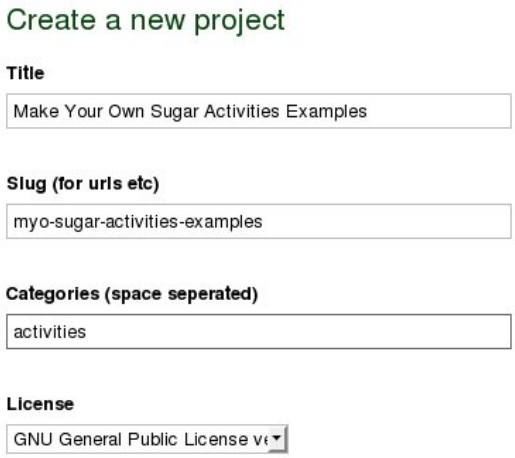
Now have a look inside the .ssh directory. By convention every file or directory name that begins with a period is considered hidden by Linux, so it won't show up in a GNOME file browser window unless you use the option on the View menu to Show Hidden Files. When you display the contents of that directory you'll see two files: **id_rsa** and **id_rsa.pub**. The public key is in id_rsa.pub. Try opening that file with gedit (Open With Text Editor) and you'll see something like this:



When you create your account on git.sugarlabs.org there will be a place where you can add your public SSH key. To do that use **Select All** from the **Edit** menu in gedit, then **Copy** and **Paste** into the field provided on the web form.

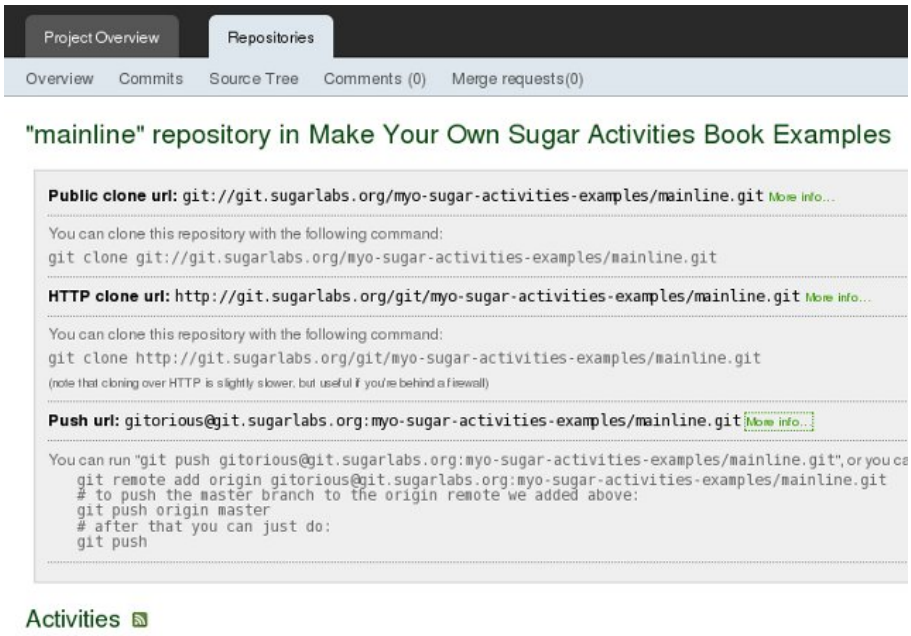
CREATE A NEW PROJECT

I'm going to create a new Project in Git for the examples for this book. I need to log in with my new account and click the **New Project** link we saw earlier. I get this form, which I have started filling in:



The **Title** is used on the website, the **Slug** is a shortened version of the title without spaces used to name the Git repository. **Categories** are optional. **License** is GPL v2 for my projects. You can choose from any of the licenses in the list for your own Projects, and you can change the license entry later if you want to. You will also need to enter a **Description** for your project.

Once you have this set up you'll be able to click on the mainline entry for the Project (like we did with Read Etexts before) and see something like this:



The next step is to convert our project files into a local Git repository, add the files to it, then push it to the repository on git.sugarlabs.org. We need to do this because you cannot **clone** an empty repository, and our remote repository is currently empty. To get around that problem we'll push the local repository out to the new remote repository we just created, then clone the remote one and delete our existing project and its Git repository. From then on we'll do all our work in the cloned repository.

This process may remind you of the Edward Albee quote, "Sometimes a person has to go a very long distance out of his way to come back a short distance correctly". Fortunately we only need to do it once per project. Enter the commands shown below in **bold** after making you project directory the current one:

```
git init
Initialized empty Git repository in
/home/jim/olpc/bookexamples/.git/
git add *.py
git add activity
git add MANIFEST
git add .gitignore
git commit -a -m "Create repository and load"
[master (root-commit) 727bfe8] Create repository and load
 9 files changed, 922 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 MANIFEST
 create mode 100755 ReadEtexts.py
 create mode 100644 ReadEtextsActivity.py
 create mode 100644 ReadEtextsActivity2.py
 create mode 100644 activity/activity.info
 create mode 100644 activity/read-etexts.svg
 create mode 100755 setup.py
 create mode 100644 toolbar.py
```

I have made an empty local Git repository with **git init**, then I've used **git add** to add the important files to it. (In fact **git add** doesn't actually add anything itself; it just tells Git to add the file on the next **git commit**). Finally **git commit** with the options shown will actually put the latest version of these files in my new local repository.

To push this local repository to git.sugarlabs.org we use the commands from the web page:

```
git remote add origin \
gitorious@git.sugarlabs.org:\
myo-sugar-activities-examples/mainline.git
git push origin master
Counting objects: 17, done.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 7.51 KiB, done.
Total 15 (delta 3), reused 0 (delta 0)
To gitorious@git.sugarlabs.org:myo-sugar-activities-examples/
mainline.git
   2cb3a1e..700789d  master -> master
=> Syncing Gitorious...
Heads up: head of  changed to
700789d3333a7257999d0a69bdcafb840e6adc09 on master
Notify cia.vc of 727bfe819d5b7b70f4f2b31d02f5562709284ac4  on
myo-sugar-activities-examples
Notify cia.vc of 700789d3333a7257999d0a69bdcafb840e6adc09  on
myo-sugar-activities-examples
[OK]
rm *
rm activity -rf
rm .git -rf
cd ~
rm Activity/ReadEtextsII
mkdir olpc
cd olpc
mkdir bookexamples
cd bookexamples
git clone \
git://git.sugarlabs.org/\
myo-sugar-activities-examples/mainline.git
Initialized empty Git repository in
/home/jim/olpc/bookexamples/mainline/.git/
remote: Counting objects: 18, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 18 (delta 3), reused 0 (delta 0)
Receiving objects: 100% (18/18), 8.53 KiB, done.
Resolving deltas: 100% (3/3), done.
```

The lines in **bold** are the commands to enter, and everything else is messages that Git sends to the console. I've split some of the longer Git commands with the backslash (\) to make them fit better on the printed page, and wrapped some output lines that would normally print on one line for the same reason. It probably isn't clear what we're doing here and why, so let's take it step by step:

- The first command **git remote add origin** tells the remote Git repository that we are going to send it stuff from our local repository.
- The second command **git push origin master** actually sends your local Git repository to the remote one and its contents will be copied in. When you enter this command you will be asked to enter the SSH pass phrase you created in the last section. GNOME will remember this phrase for you and enter it for every Git command afterwards so you don't need to. It will keep doing this until you log out or turn off the computer.
- The next step is to delete our existing files and our local Git repository (which is contained in the hidden directory .git). The **rm .git -rf** means "Delete the directory .git and everything in it". **rm** is a Unix command, not part of Git. If you like you can delete your existing files *after* you create the cloned repository in the next step. Note the command **rm Activity/ReadEtextsIl**, which deletes the symbolic link to our old project that we created by running **./setup.py dev**. We'll need to go to our new cloned project directory and run that again before we can test our Activity again.
- Now we do the **git clone** command from the web page. This takes the remote Git repository we just added our MANIFEST file to and makes a new local repository in directory **/*yourhome*/olpc/bookexamples/mainline**.

Finally we have a local repository we can use. Well, not quite. We can commit our code to it but we cannot push anything back to the remote repository because our local repository isn't configured correctly yet.

What we need to do is edit the file **config** in directory **.git** in **/*yourhome*/olpc/bookexamples/mainline**. We can use gedit to do that. We need to change the **url=** entry to point to the **Push url** shown on the mainline web page. When we're done our **config** file should look like this:

```
[core]
 repositoryformatversion = 0
 filemode = true
 bare = false
 logallrefupdates = true
[remote "origin"]
         url = gitorious@git.sugarlabs.org:
myo-sugar-activities-examples/mainline.git
 fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
 remote = origin
 merge = refs/heads/master
```

The line in **bold** is the only one that gets changed. It is split here to make it fit on the printed page. In your own files it should all be one line with no spaces between the colon(:) that ends the first line and the beginning of the second line.

From now on anyone who wants to work on our project can get a local copy of the Git repository by doing this from within the directory where he wants the repository to go:

```
git clone git://git.sugarlabs.org/\
myo-sugar-activities-examples/mainline.git
```

He'll have to change his **.git/config** file just like we did, then he'll be ready to go.

EVERYDAY USE OF GIT

While getting the repositories set up to begin with is a chore, daily use is not. There are only a few commands you'll need to work with. When we left off we had a repository in **/*yourhome*/olpc/bookexamples/mainline** with our files in it. We will need to add any new files we create too.

We use the **git add** command to tell Git that we want to use Git to store a particular file. This doesn't actually store anything, it just tells Git our intentions. The format of the command is simply:

```
git add file_or_directory_name
```

There are files we *don't* want to add to Git, to begin with those files that end in **.pyc**. If we never do a **git add** on them they'll never get added, but Git will constantly ask us why we aren't adding them. Fortunately there is a way to tell Git that we really, really don't want to add those files. We need to create a file named **.gitignore** using gedit and put in entries like this:

```
*.pyc
*.e4p
*.zip
dist
locale
.eric4project/
.ropeproject/
```

These entries will also ignore project files used by Eric and zip files containing ebooks, The **dist** entry refers to the dist directory, which is where packaged Activities go, and the **locale** entry refers to a locale directory that will be created when we package the Activity. Neither of these directories belong in the Git repository. Once we have the .gitignore file created in the mainline directory we can add it to the repository:

```
git add .gitignore
git commit -a -m "Add .gitignore file"
```

From now on Git will no longer ask us to add .pyc or other unwanted files that match our patterns. If there are other files we don't want in the repository we can add them to .gitignore either as full file names or directory names or as patterns like *.pyc.

In addition to adding files to Git we can remove them too:

```
git rm filename
```

Note that this just tells Git that from now on it will not be keeping track of a given filename, and that will take effect at the next commit. Old versions of the file are still in the repository.

If you want to see what changes will be applied at the next commit run this:

```
git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will
#   be committed)
#
# modified:   ReadEtextsActivity.py
#
no changes added to commit (use "git add" and/or
"git commit -a")
```

Finally, to put your latest changes in the repository use this:

```
git commit -a -m "Change use of instance directory to tmp"
Created commit a687b27: Change use of instance
directory to tmp
 1 files changed, 2 insertions(+), 2 deletions(-)
```

If you leave off the -m an editor will open up and you can type in a comment, then save and exit. Unfortunately by default the editor is **vi**, an old text mode editor that is not friendly like gedit.

When we have all our changes done we can send them to the central repository using **git push**:

```
git push
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
To gitorious@git.sugarlabs.org:
myo-sugar-activities-examples/mainline.git
   700789d..a687b27  master -> master
=> Syncing Gitorious...
Heads up: head of  changed to
a687b27e2f034e5a17d2ca2fe9f2787c7f633e64 on master
Notify cia.vc of a687b27e2f034e5a17d2ca2fe9f2787c7f633e64
on myo-sugar-activities-examples
[OK]
```

We can get the latest changes from other developers by doing **git pull**:

```
git pull
remote: Counting objects: 17, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 3), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
From gitorious@git.sugarlabs.org:
myo-sugar-activities-examples/mainline
   2cb3a1e..700789d  master     -> origin/master
Updating 2cb3a1e..700789d
Fast forward
 .gitignore               |    6 +
 MANIFEST                 |  244 +--------------------------
---------
 ReadEtexts.py            |  182 ++++++++++++++++++++++++++++
 ReadEtextsActivity.py    |  182 ++++++++++++++++++++++++++++
 ReadEtextsActivity2.py   |  311 ++++++++++++++++++++++++++++++++++
++++++++++++++++++
 activity/activity.info   |    9 ++
 activity/read-etexts.svg |   71 ++++++++++
 setup.py                 |   21 +++
 toolbar.py               |  136 ++++++++++++++++++++
 9 files changed, 921 insertions(+), 241 deletions(-)
 create mode 100644 .gitignore
 create mode 100755 ReadEtexts.py
 create mode 100644 ReadEtextsActivity.py
 create mode 100644 ReadEtextsActivity2.py
 create mode 100644 activity/activity.info
 create mode 100644 activity/read-etexts.svg
 create mode 100755 setup.py
 create mode 100644 toolbar.py
```

12. GOING INTERNATIONAL WITH POOTLE

INTRODUCTION

The goal of Sugar Labs and One Laptop Per Child is to educate all the children of the world, and we can't do that with Activities that are only available in one language. It is equally true that making separate versions of each Activity for every language is not going to work, and expecting Activity developers to be fluent in many languages is not realistic either. We need a way for Activity developers to be able to concentrate on creating Activities and for those who can translate to just do that. Fortunately, this is possible and the way it's done is by using *gettext*.

GETTING TEXT WITH GETTEXT

You should remember that our latest code example made use of an odd import:

```
from gettext import gettext as _
```

The "_()" function was used in statements like this:

```
        self.back.set_tooltip(_('Back'))
```

At the time I explained that this odd looking function was used to translate the word "Back" into other languages, so that when someone looks at the Back button's tool tip he'll see the text in his own language. I also said that if it was not possible to translate this text the user would see the word "Back" untranslated. In this chapter we'll learn more about how this works and what we have to do to support the volunteers who translate these text strings into other languages.

The first thing you need to learn is how to properly format the text strings to be translated. This is an issue when the text strings are actual sentences containing information. For example, you might write such a message this way:

```
    message = _("User ") + username + \
        _(" has joined the chat room.")
```

This would work, but you've made things difficult for the translator. He has two separate strings to translate and no clue that they belong together. It is much better to do this:

```
    message = _("User %s has joined the chat room.") % \
        username
```

If you know both statements give the same resulting string then you can easily see why a translator would prefer the second one. Use this technique whenever you need a message that has some information inserted into it. When you use it, try and limit yourself to only one format code (the %s) per string. If you use more than one it can cause problems for the translator.

GOING TO POT

Assuming that every string of text a user might be shown by our Activity is passed through "_()" the next step is to generate a pot file. You can do this by running setup.py with a special option:

```
./setup.py genpot
```

This creates a directory called **po** and puts a file ***ActivityName*.pot** in that directory. In the case of our example project *ActivityName* is **ReadEtextsII**. This is the contents of that file:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the
# PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2010-01-06 18:31-0600\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: activity/activity.info:2
msgid "Read ETexts II"
msgstr ""

#: toolbar.py:34
msgid "Back"
msgstr ""

#: toolbar.py:40
msgid "Forward"
msgstr ""

#: toolbar.py:115
msgid "Zoom out"
msgstr ""

#: toolbar.py:120
msgid "Zoom in"
msgstr ""

#: toolbar.py:130
msgid "Fullscreen"
msgstr ""

#: ReadEtextsActivity2.py:34
msgid "Edit"
msgstr ""

#: ReadEtextsActivity2.py:38
msgid "Read"
msgstr ""

#: ReadEtextsActivity2.py:46
msgid "View"
msgstr ""
```

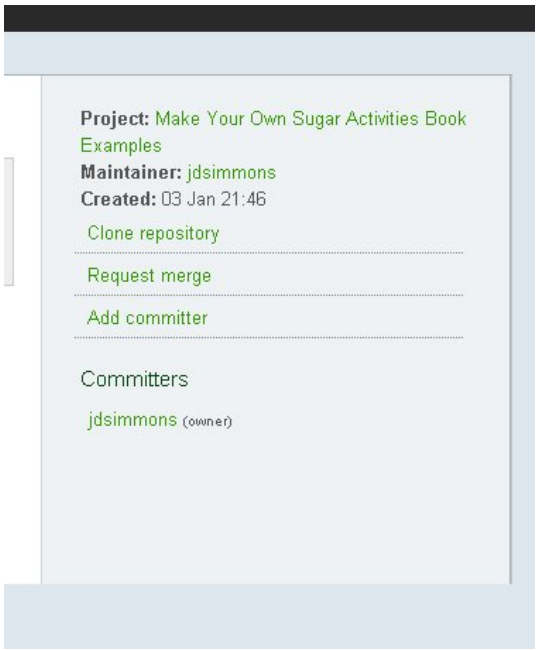
This file contains an entry for every text string in our Activity (as msgid) and a place to put a translation of that string (msgstr). Copies of this file will be made by the Pootle server for every language desired, and the msgstr entries will be filled in by volunteer translators.

GOING TO POOTLE

Before any of that can happen we need to get our POT file into Pootle. The first thing we need to do is get the new directory into our Git repository and push it out to Gitorious. You should be familiar with the needed commands by now:

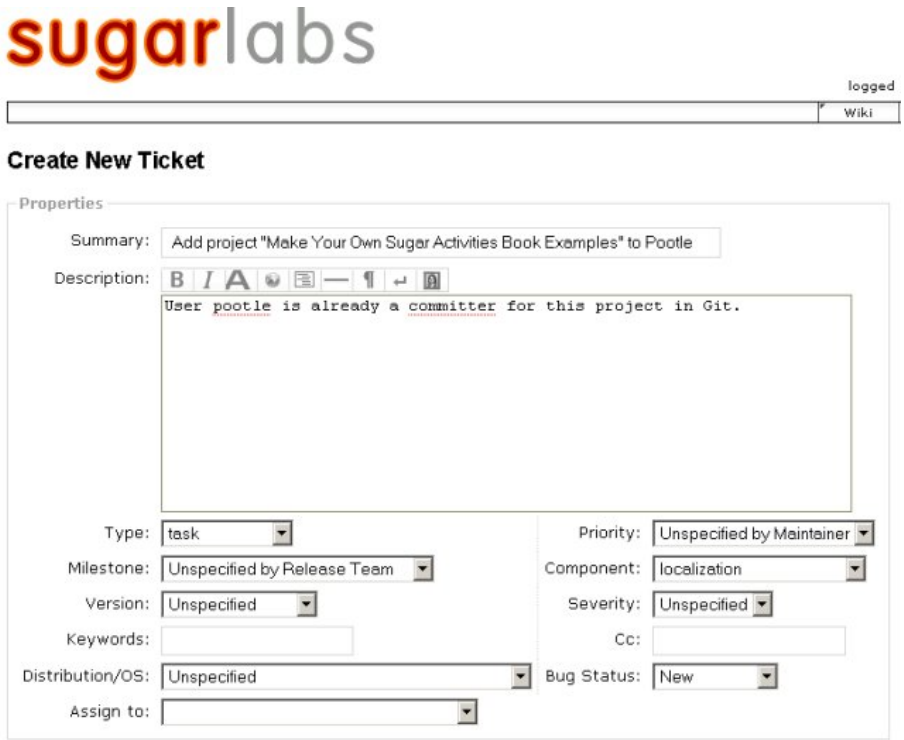
```
git add po
git commit -a -m "Add POT file"
git push
```

Next we need to give the user "pootle" commit authority to our Git project. Go to git.sugarlabs.org, sign in, and find your Project page and click on the mainline link. You should see this on the page that takes you to:



Click on the **Add committer** link and type in the name **pootle** in the form that takes you to. When you come back to this page **pootle** will be listed under Committers.

Your next step is to go to web site http://bugs.sugarlabs.org and register for a user id. When you get that open up a ticket something like this:



The **Component** entry *localization* should be used, along with **Type** *task*.

Believe it or not, this is all you need to do to get your Activity set up to be translated.

PAY NO ATTENTION TO THAT MAN BEHIND THE CURTAIN

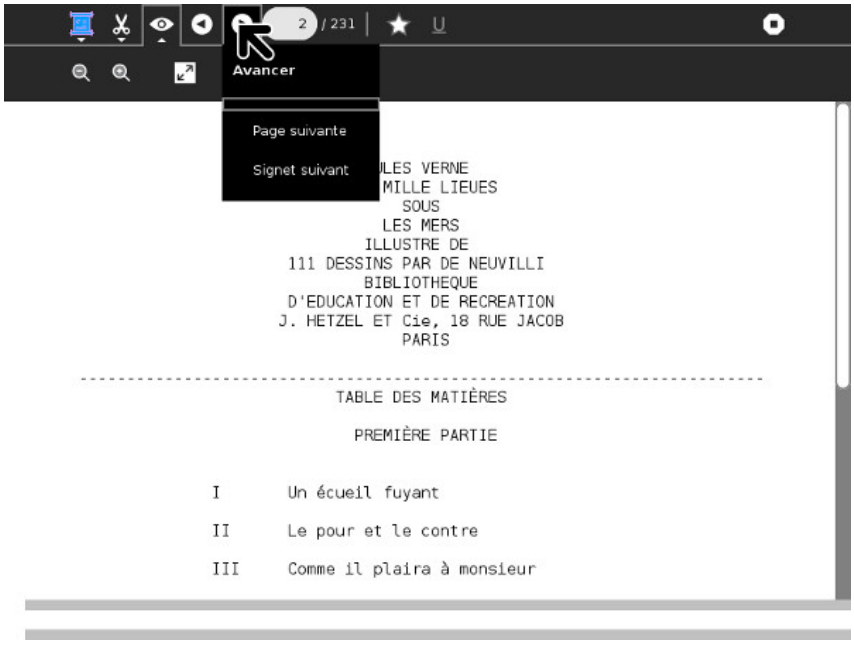
After this you'll need to do a few things to get translations from Pootle into your Activity.

- When you add text strings (labels, error messages, etc.) to your Activity always use the **_()** function with them so they can be translated.
- After adding new strings always run **./setup.py genpot** to recreate the POT file.
- After that commit and push your changes to Gitorious.
- Every so often, and especially before releasing a new version, do a **git pull**. If there are any localization files added to Gitorious this will bring them to you.

Localization with Pootle will create a large number of files in your project, some in the **po** directory and others in a new directory called **locale**. These will be included in the .xo file that you will use to distribute your Activity. The locale directory does not belong in the Git repository, however. It will be generated fresh each time you package your Activity.

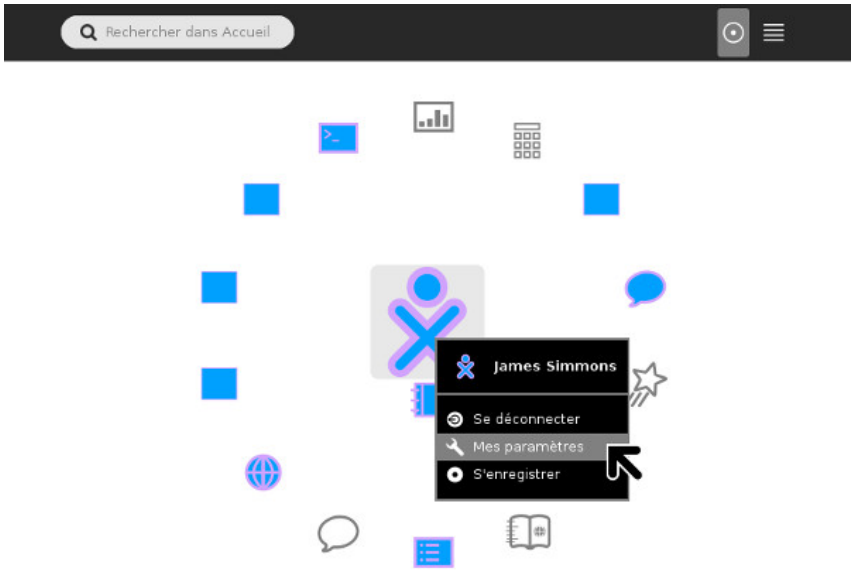
C'EST MAGNIFIQUE!

Here is a screen shot of the French language version of **Read Etexts** reading Jules Verne's novel *Vingt Mille Lieues Sour Le Mer*:

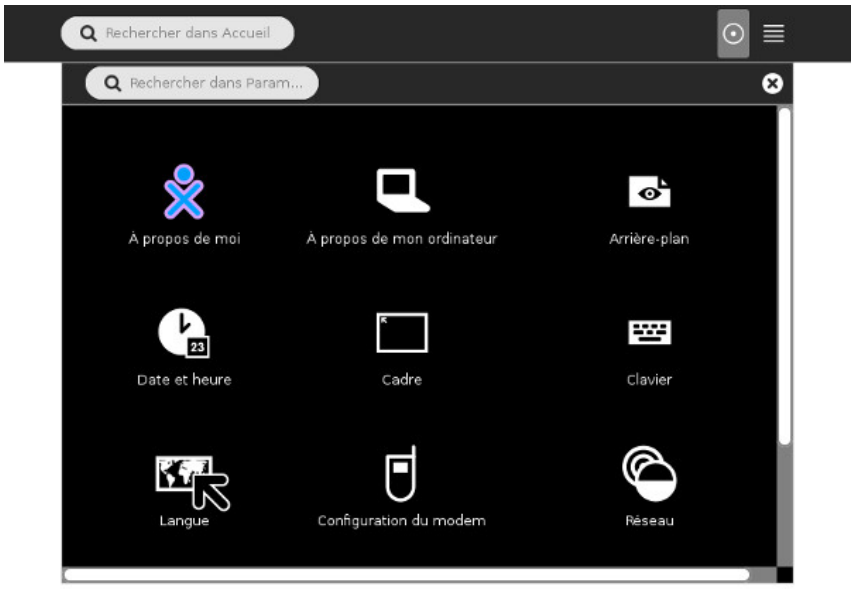


There is reason to believe that the book is in French too.

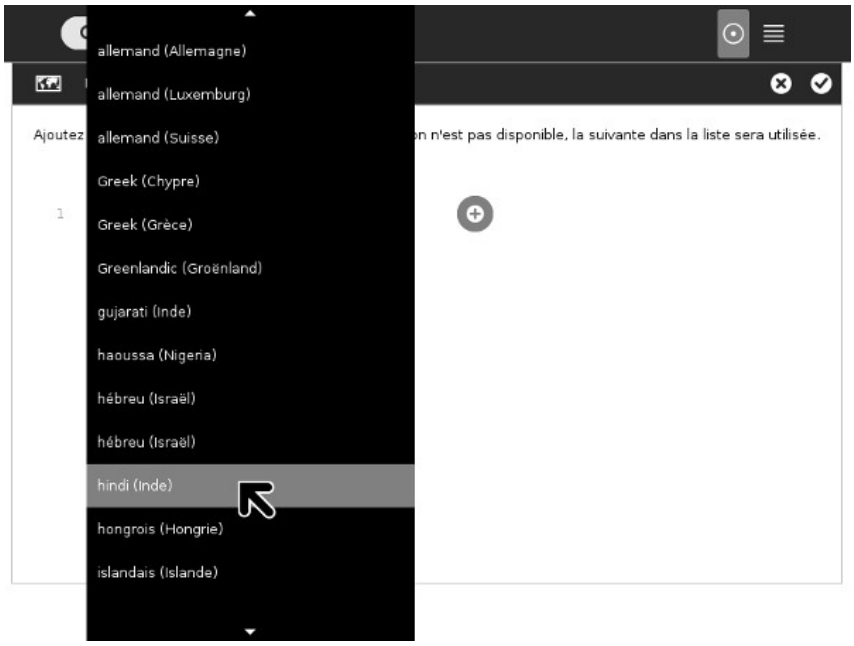
If you want to see how your own Activity looks in different languages (and you have localization files from Pootle for your Activity) all you need to do is bring up the **Settings** menu option in Sugar (shown here in French):



Then choose the **Language** icon like this:



And finally, pick your language. You will be prompted to restart Sugar.
After you do you'll see everything in your new chosen language.



And that's all there is to it!

13. DISTRIBUTE YOUR ACTIVITY

CHOOSE A LICENSE

Before you give your Activity to anyone you need to choose a license that it will be distributed under. Buying software is like buying a book. There are certain rights you have with a book and others you don't have. If you buy a copy of *The DaVinci Code* you have the right to read it, to loan it out, to sell it to a used bookstore, or to burn it. You do *not* have the right to make copies of it or to make a movie out of it. Software is the same way, but often worse. Those long license agreements we routinely accept by clicking a button might not allow you to sell the software when you're done with it, or even give it away. If you sell your computer you may find that the software you bought is only good for that computer, and only while you are the owner of the computer. (You can get good deals on reconditioned computers with no operating system installed for that very reason).

If you are in the business of selling software you might have to hire a lawyer to draw up a license agreement, but if you're giving away software there are several standard licenses you can choose from for free. The most popular by far is called the *General Public License*, or GPL. Like the licenses Microsoft uses it allows the people who get your program to do some things with it but not others. What makes it interesting is not what it allows them to do (which is pretty much anything they like) but what it forbids them to do.

If someone distributes a program licensed under the GPL they are also required to make the source code of the program available to anyone who wants it. That person may do as he likes with the code, with one important restriction: if he distributes a program based on that code he must *also* license that code using the GPL. This makes it impossible for someone to take a GPL licensed work, improve it, and sell it to someone without giving him the source code to the new version.

While the GPL is not the only license available for Activities to be distributed on http://activities.sugarlabs.org all the licenses require that anyone getting the Activity also gets the complete source code for it. You've already taken care of that requirement by putting your source code in Gitorious. If you used any code from an existing Activity licensed with the GPL you *must* license your own code the same way. If you used a significant amount of code from this book (which is also GPL licensed) you may be required to use the GPL too.

Is licensing something you should worry about? Not really. The only reason you'd want to use a license other than the GPL is if you wanted to sell your Activity instead of give it away. Consider what you'd have to do to make that possible:

- You'd have to use some language other than Python so you could give someone the program without giving them the source code.
- You would have to have your own source code repository not available to the general public and make arrangements to have the data backed up regularly.
- You would have to have your own website to distribute the Activity. The website would have to be set up to accept payments somehow.
- You would have to advertise this website somehow or nobody would know your Activity existed.
- You would have to have a lawyer draw up a license for your Activity.
- You would have to come up with some mechanism to keep your customers from giving away copies of your Activity.
- You would have to create an Activity so astoundingly clever that nobody else could make something similar and give it away.
- You would have to deal with the fact that your "customers" would be children with no money or credit cards.

In summary, activities.sugarlabs.org is not the *iPhone App Store*. It is a place where programmers share and build upon each other's work and give the results to children for free. The GPL encourages that to happen, and I recommend that you choose that for your license.

ADD LICENSE COMMENTS TO YOUR PYTHON CODE

At the top of each Python source file in your project (except **setup.py**, which is already commented) put comments like this:

```
# filename    Program description
#
# Copyright (C) 2010 Your Name Here
#
# This program is free software; you can redistribute it
# and/or modify it under the terms of the GNU General
# Public License as published by the Free Software
# Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will
# be useful, but WITHOUT ANY WARRANTY; without even
# the implied warranty of MERCHANTABILITY or FITNESS FOR
# A PARTICULAR PURPOSE.  See the GNU General Public
# License for more details.
#
# You should have received a copy of the GNU General
# Public License along with this program; if not, write
# to the Free Software Foundation, Inc., 51 Franklin
# St, Fifth Floor, Boston, MA 02110-1301  USA
```

If the code is based on someone else's code you should mention that as a courtesy.

CREATE AN .XO FILE

Make certain that activity.info has the version number you want to give your Activity (currently it must be a positive integer) and run this command:

```
./setup.py dist_xo
```

This will create a **dist** directory if one does not exist and put a file named something like **ReadETextsII-1.xo** in it. The "1" indicates version 1 of the Activity.

If you did everything right this **.xo** file should be ready to distribute. You can copy it to a thumb drive and install it on an XO laptop or onto another thumb drive running *Sugar on a Stick*. You probably should do that before distributing it any further. I like to live with new versions of my Activities for a week or so before putting them on activities.sugarlabs.org.

Now would be a good time to add **dist** and **locale** to your **.gitignore** file (if you haven't done that already), then commit it and push it to Gitorious. You don't want to have copies of your .xo files in Git. Another good thing to do at this point would be to tag your Git repository with the version number so you can identify which code goes with which version.

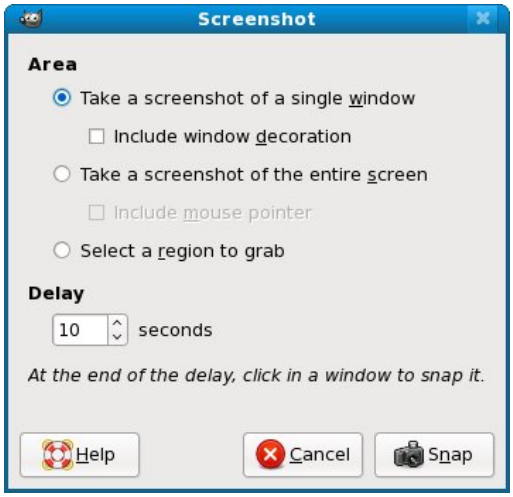
```
git tag -m "Release 1" v1 HEAD
git push --tags
```

ADD YOUR ACTIVITY TO ASLO

When you're ready to post the .xo file on ASLO you'll create an account as you did with the other websites. When you've logged in there you'll see a **Tools** link in the upper right corner of the page. Click on that and you'll see a popup menu with an option for **Developer Hub**, which you should click on. That will take you to the pages where you can add new Activities. The first thing it asks for when setting up a new Activity is what license you will use. After that you should have no problem getting your Activity set up.

You will need to create an Activity icon as a .gif file and create screen shots of your Activity in action. You can do both of these things with *The GIMP* (GNU Image Manipulation Program). For the icon all you need to do is open the .svg file with The GIMP and **Save As** a .gif file.

For the screen shots use sugar-emulator to display your Activity in action, then use the **Screenshot** option from the **Create** submenu of the **File** menu with these options:



This tells GIMP to wait 10 seconds, then take a screenshot of the window you click on with the mouse. You'll know that the 10 seconds are up because the mouse pointer will change shape to a plus (+) sign. You also tell it *not* to include the window decoration (which means the window title bar and border). Since windows in Sugar do not have decorations eliminating the decorations used by sugar-emulator will give you a screenshot that looks exactly like a Sugar Activity in action.

Every Activity needs one screenshot, but you can have more if you like. Screenshots help sell the Activity and instruct those who will use it on what the Activity can do. Unfortunately, ASLO cannot display pictures in a predictable sequence, so it is not suited to displaying steps to perform.

Another thing you'll need to provide is a home page for your Activity. The one for **Read Etexts** is here:

http://wiki.sugarlabs.org/go/Activities/Read_Etexts

Yes, one more website to get an account for. Once you do you can specify a link with */go/Activities/some_name* and when you click on that link the Wiki will create a page for you. The software used for the Wiki is *MediaWiki*, the same as used for *Wikipedia*. Your page does not need to be as elaborate as mine is, but you definitely should provide a link to your source code in Gitorious.

14. DEBUGGING SUGAR ACTIVITIES

INTRODUCTION

No matter how careful you are it is reasonably likely that your Activity will not work perfectly the first time you try it out. Debugging a Sugar Activity is a bit different than debugging a standalone program. When you test a standalone program you just run the program itself. If there are syntax errors in the code you'll see the error messages on the console right away, and if you're running under the **Eric** IDE the offending line of code will be selected in the editor so you can correct it and keep going.

With Sugar it's a bit different. It's the Sugar environment, not Eric, that runs your program. If there are syntax errors in your code you won't see them right away. Instead, the blinking Activity icon you see when your Activity starts up will just keep on blinking for several minutes and then will just go away, and your Activity won't start up. The only way you'll see the error that caused the problem will be to use the **Log Activity**. If your program has no syntax errors but does have logic errors you won't be able to step through your code with a debugger to find them. Instead, you'll need to use some kind of logging to trace through what's happening in your code, and again use the Log Activity to view the trace messages. Now would be a good time to repeat some advice I gave before:

MAKE A STANDALONE VERSION OF YOUR PROGRAM FIRST

Whatever your Activity does, it's a good bet that 80% of it could be done by a standalone program which would be much less tedious to debug. If you can think of a way to make your Activity runnable as either an Activity or a standalone Python program then by all means do it.

USE PYLINT, PYCHECKER, OR PYFLAKES

One of the advantages of a compiled language like **C** over an interpreted language like Python is that the compiler does a complete syntax check of the code before converting it to machine language. If there are syntax errors the compiler gives you informative error messages and stops the compile. There is a utility call **lint** which C programmers can use to do even more thorough checks than the compiler would do and find questionable things going on in the code.

Python does not have a compiler but it does have several lint-like utilities you can run on your code before you test it. These utilities are **pyflakes**, **pychecker**, and **pylint**. Any Linux distribution should have all three available.

PyFlakes

Here is an example of using PyFlakes:

```
pyflakes minichat.py
minichat.py:25: 'COLOR_BUTTON_GREY' imported but unused
```

```
minichat.py:28: 'XoColor' imported but unused
minichat.py:29: 'Palette' imported but unused
minichat.py:29: 'CanvasInvoker' imported but unused
```

PyFlakes seems to do the least checking of the three, but it does find errors like these above that a human eye would miss.

PyChecker

Here is PyChecker in action:

```
pychecker ReadEtextsActivity.py
Processing ReadEtextsActivity...
/usr/lib/python2.5/site-packages/dbus/_dbus.py:251:
DeprecationWarning: The dbus_bindings module is not public
API and will go away soon.

Most uses of dbus_bindings are applications catching
the exception dbus.dbus_bindings.DBusException.
You should use dbus.DBusException instead (this is
compatible with all dbus-python versions since 0.40.2).

If you need additional public API, please contact
the maintainers via <dbus@lists.freedesktop.org>.

  import dbus.dbus_bindings as m

Warnings...

/usr/lib/python2.5/site-packages/sugar/activity/activity.py:847:
Parameter (ps) not used
/usr/lib/python2.5/site-packages/sugar/activity/activity.py:992:
Parameter (event) not used
/usr/lib/python2.5/site-packages/sugar/activity/activity.py:992:
Parameter (widget) not used
/usr/lib/python2.5/site-packages/sugar/activity/activity.py:996:
Parameter (widget) not used

/usr/lib/python2.5/site-packages/sugar/graphics/window.py:157:
No class attribute (_alert) found
/usr/lib/python2.5/site-packages/sugar/graphics/window.py:164:
Parameter (window) not used
/usr/lib/python2.5/site-packages/sugar/graphics/window.py:188:
Parameter (widget) not used
/usr/lib/python2.5/site-packages/sugar/graphics/window.py:200:
Parameter (event) not used
/usr/lib/python2.5/site-packages/sugar/graphics/window.py:200:
Parameter (widget) not used

ReadEtextsActivity.py:62: Parameter (widget) not used

4 errors suppressed, use -#/--limit to increase the number
of errors displayed
```

PyChecker not only checks your code, it checks the code you import, including Sugar code.

PyLint

Here is PyLint, the most thorough of the three:

```
pylint ReadEtextsActivity.py
No config file found, using default configuration
************* Module ReadEtextsActivity
C:177: Line too long (96/80)
C:  1: Missing docstring
C: 27: Operator not preceded by a space
page=0
    ^
C: 27: Invalid name "page" (should match
(([A-Z_][A-Z0-9_]*)|(__.*__))$)
C: 30:ReadEtextsActivity: Missing docstring
C:174:ReadEtextsActivity.read_file: Invalid name "zf" (should
match [a-z_][a-z0-9_]{2,30}$)
W: 30:ReadEtextsActivity: Method 'write_file' is abstract
in class 'Activity' but is not overridden
R: 30:ReadEtextsActivity: Too many ancestors (12/7)
W: 33:ReadEtextsActivity.__init__: Using the global statement
R: 62:ReadEtextsActivity.keypress_cb:
Too many return statements (7/6)
C: 88:ReadEtextsActivity.page_previous: Missing docstring
W: 89:ReadEtextsActivity.page_previous:
```

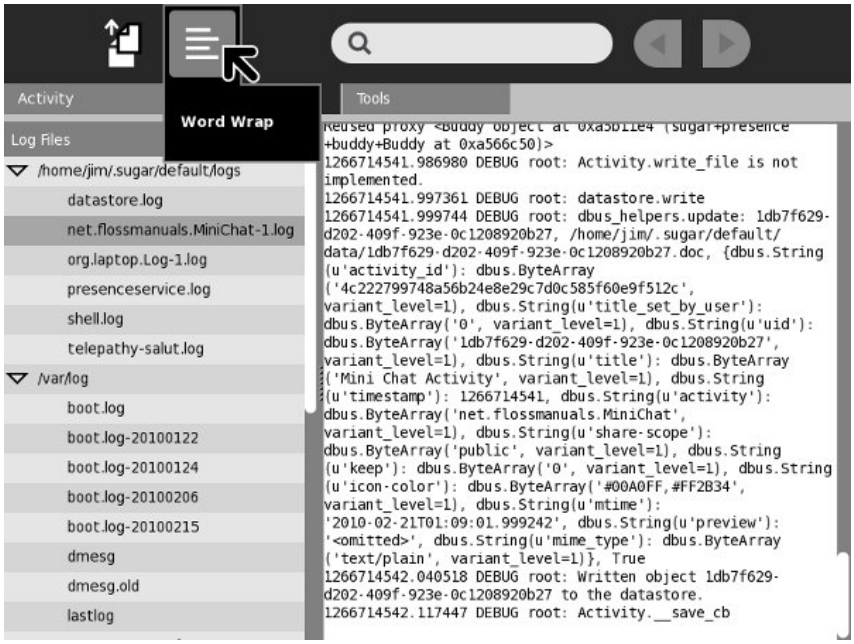
```
Using the global statement
C: 90:ReadEtextsActivity.page_previous:
Operator not preceded by a space
        page=page-1
            ^
C: 91:ReadEtextsActivity.page_previous:
Operator not preceded by a space
        if page < 0: page=0
                     ^
C: 91:ReadEtextsActivity.page_previous: More than one
statement on a single line
C: 96:ReadEtextsActivity.page_next: Missing docstring
W: 97:ReadEtextsActivity.page_next: Using the global
statement
C: 98:ReadEtextsActivity.page_next: Operator not preceded
by a space
        page=page+1
            ^
C: 99:ReadEtextsActivity.page_next: More than one
statement on a single line
C:104:ReadEtextsActivity.font_decrease: Missing docstring
C:112:ReadEtextsActivity.font_increase: Missing docstring
C:118:ReadEtextsActivity.scroll_down: Missing docstring
C:130:ReadEtextsActivity.scroll_up: Missing docstring
C:142:ReadEtextsActivity.show_page: Missing docstring
W:143:ReadEtextsActivity.show_page: Using global for
'PAGE_SIZE' but no assigment is done
W:143:ReadEtextsActivity.show_page: Using global for
'current_word' but no assigment is done
W:157:ReadEtextsActivity.save_extracted_file: Redefining
name 'zipfile' from outer scope (line 21)
C:163:ReadEtextsActivity.save_extracted_file: Invalid
name "f" (should match [a-z_][a-z0-9_]{2,30}$)
W:171:ReadEtextsActivity.read_file: Using global
for 'PAGE_SIZE' but no assigment is done
C:177:ReadEtextsActivity.read_file: Invalid name
"currentFileName" (should match [a-z_][a-z0-9_]{2,30}$)
C:179:ReadEtextsActivity.read_file: Invalid name
"currentFileName" (should match [a-z_][a-z0-9_]{2,30}$)
C:197:ReadEtextsActivity.make_new_filename: Missing
docstring
R:197:ReadEtextsActivity.make_new_filename: Method could be
a function
R: 30:ReadEtextsActivity: Too many public methods (350/20)
W:174:ReadEtextsActivity.read_file: Attribute
'zf' defined outside __init__
W:181:ReadEtextsActivity.read_file: Attribute
'etext_file' defined outside __init__
W:175:ReadEtextsActivity.read_file: Attribute
'book_files' defined outside __init__
W:182:ReadEtextsActivity.read_file: Attribute
'page_index' defined outside __init__

... A bunch of tables appear here ...

Global evaluation
-----------------
Your code has been rated at 7.52/10 (previous run: 7.52/10)
```

PyLint is the toughest on your code and your ego. It not only tells you about syntax errors, it tells you everything someone might find fault with in your code. This includes style issues that won't affect how your code runs but will affect how readable it is to other programmers.

THE LOG ACTIVITY

When you start testing your Activities the Log Activity will be like your second home. It displays a list of log files in the left pane and when you select one it will display the contents of the file in the right pane. Every time you run your Activity a new log file is created for it, so you can compare the log you got this time with what you got on previous runs. The **Edit** toolbar is especially useful. It contains a button to show the log file with lines wrapped (which is not turned on by default but probably should be). It has another button to copy selections from the log to the clipboard, which will be handy if you want to show log messages to other developers.

The **Tools** toolbar has a button to delete log files. I've never found a reason to use it. Log files go away on their own when you shut down sugar-emulator.



Here is what the Log Activity looks like showing a syntax error in your code:

LOGGING

Without a doubt the oldest debugging technique there is would be the simple print statement. If you have a running program that misbehaves because of logic errors and you can't step through the code in a debugger to figure out what's happening you might print statements in your code. For instance, if you aren't sure that a method is ever getting executed you might put a statement like this as the first line of the method:

```
def my_method():
    print 'my_method() begins'
```

You can include data in your print statements too. Suppose you need to know how many times a loop is run. You could do this:

```
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line,
                'iso-8859-1')
            linecount = linecount + 1
            print 'linecount=', linecount
```

The output of these print statements can be seen in the Log Activity. When you're finished debugging your program you would remove these statements.

An old programming book I read once made the case for leaving the statements in the finished program. The authors felt that using these statements for debugging and them removing them is a bit like wearing a parachute when the plane is on the ground and taking it off when it's airborne. If the program is out in the world and has problems you might well wish you had those statements in the code so you could help the user and yourself figure out what's going on. On the other hand, print statements aren't free. They do take time to run and they fill up the log files with junk. What we need are print statements that you can turn on an off.

The way you can do this is with Python Standard Logging. In the form used by most Activities it looks like this:

```
        self._logger = logging.getLogger(
            'read-etexts-activity')
```

These statements would go in the *__init__()* method of your Activity. Every time you want to do a print() statement you would do this instead:

```
    def _shared_cb(self, activity):
        self._logger.debug('My activity was shared')
        self.initiating = True
        self._sharing_setup()

        self._logger.debug(
              'This is my activity: making a tube...')
        id = self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            OfferDBusTube(SERVICE, {})

    def _sharing_setup(self):
        if self._shared_activity is None:
            self._logger.error(
                  'Failed to share or join activity')
            return
```

Notice that there are two kinds of logging going on here: **debug** and **error**. These are error levels. Every statement has one, and they control which log statements are run and which are ignored. There are several levels of error logging, from lowest severity to highest:

```
self._logger.debug("debug message")
self._logger.info("info message")
self._logger.warn("warn message")
self._logger.error("error message")
self._logger.critical("critical message")
```

When you set the error level in your program to one of these values you get messages with that level and higher. You can set the level in your program code like this:

```
self._logger.setLevel(logging.DEBUG)
```

You can also set the logging level outside your program code using an **environment variable**. For instance, in Sugar .82 and lower you can start sugar-emulator like this:

```
SUGAR_LOGGER_LEVEL=debug sugar-emulator
```

You can do the same thing with .84 and later, but there is a more convenient way. Edit the file **~/.sugar/debug** and uncomment the line that sets the SUGAR_LOGGER_LEVEL. Whatever value you have for SUGAR_LOGGER_LEVEL in ~/.sugar/debug will override the one set by the environment variable, so either change the setting in the file or use the environment variable, but don't do both.

THE ANALYZE ACTIVITY

Another Activity you may find yourself using at some point is **Analyze**. This is more likely to be used to debug Sugar itself than to debug your Activity. If, for instance, your collaboration test environment doesn't seem to be working this Activity might help you or someone else figure out why.

I don't have a lot to say about this Activity here, but you should be aware that it exists.

ADVANCED TOPICS

15. MAKING SHARED ACTIVITIES
16. ADDING TEXT TO SPEECH
17. FUN WITH THE JOURNAL
18. MAKING ACTIVITIES USING PYGAME

15. MAKING SHARED ACTIVITIES

INTRODUCTION

One of the distinctive features of Sugar is how many Activities support being used by more than one person at a time. More and more computers are being used as a communications medium. The latest computer games don't just pit the player against the computer; they create a world where players compete against each other. Websites like *Facebook* are increasingly popular because they allow people to interact with each other and even play games. It is only natural that educational software should support these kinds of interactions.

I have a niece that is an enthusiastic member of the *Club Penguin* website created by Disney. When I gave her Sugar on a Stick Blueberry as an extra Christmas gift I demonstrated the Neighborhood view and told her that Sugar would make her whole computer like *Club Penguin*. She thought that was a pretty cool idea. I felt pretty cool saying it.

RUNNING SUGAR AS MORE THAN ONE USER

Before you write any piece of software you need to give some thought to how you will test it. In the case of a shared Activity you might think you'd need more than one computer available to do testing, but those who designed Sugar did give some thought to testing shared Activities and gave us ways to test them using only one computer. These methods have been evolving so there are slight variations in how you test depending on the version of Sugar you're using. The first thing you have to know is how to run multiple copies of Sugar as different users.

To test Sugar as multiple users on one computer you need to create a second Linux user and run your sugar-runners as two separate Linux users. In the GNOME environment there is an option **Users and Groups** in the **Administration** submenu of the **System** menu which will enable you to set up a second user. Before it comes up it will prompt you for the administrative (root) password you created when you first set up Linux.

Creating the second user is simple enough, but how do you go about being logged in as two different users at the same time? It's actually pretty simple. You need to open a terminal window and type this:

```
ssh -XY jausten@localhost
```

where "jausten" is the userid of the second user. You will be asked to verify that the computer at "localhost" should be trusted. Since "localhost" just means that you are using the network to connect to another account on the same computer it is safe to answer "yes". Then you will be prompted to enter her password, and from then on everything you do in that terminal window will be done as her. You can launch sugar-emulator from that terminal and the first time you do it will prompt you for icon colors. Unlike on the XO laptop, you will not be prompted for a name, Instead the name associated with the userid you're running under will be the name you'll use in Sugar. You won't be able to change it.

If you're testing with sugar-build you will need a separate install of sugar-build for each user. (In other words, you'll need to log in as each user and do each install in that user's home directory). These additional installs will go quickly because you installed all the dependencies the first time.

CONNECTING TO OTHER USERS

Sugar uses software called **Telepathy** that implements an instant messaging protocol called **XMPP** (*Extended Messaging and Presence Protocol*). This protocol used to be called **Jabber**. In essence Telepathy lets you put an instant messaging client in your Activity. You can use this to send messages from user to user, execute methods remotely, and do file transfers.

There are actually two ways that Sugar users can join together in a network:

Salut

If two computer users are connected to the same segment of a network they should be able to find each other and share Activities. If you have a home network where everyone uses the same router you can share with others on that network. This is sometimes called *Link-Local XMPP*. The Telepathy software that makes this possible is called **Salut**.

The XO laptop has special hardware and software to support *Mesh Networking*, where XO laptops that are near each other can automatically start networking with each other without needing a router. As far as Sugar is concerned, it doesn't matter what kind of network you have. Wired or wireless, Mesh or not, they all work.

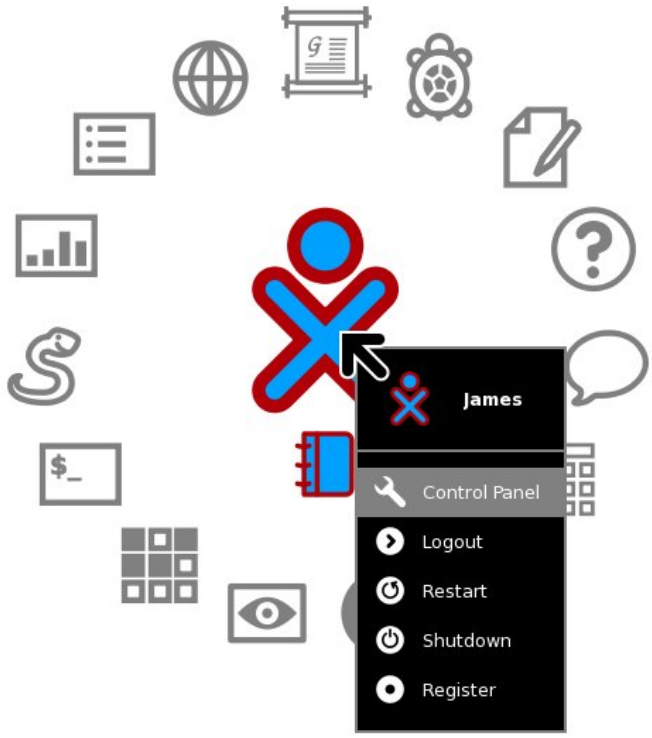
Jabber Server

The other way to connect to other users is by going through a Jabber Server. The advantage of using a Jabber server is you can contact and share Activities with people outside your own network. These people might even be on the other side of the world. Jabber allows Activities in different networks to connect when both networks are protected by firewalls. The part of Telepathy that works with a Jabber server is called **Gabble**.

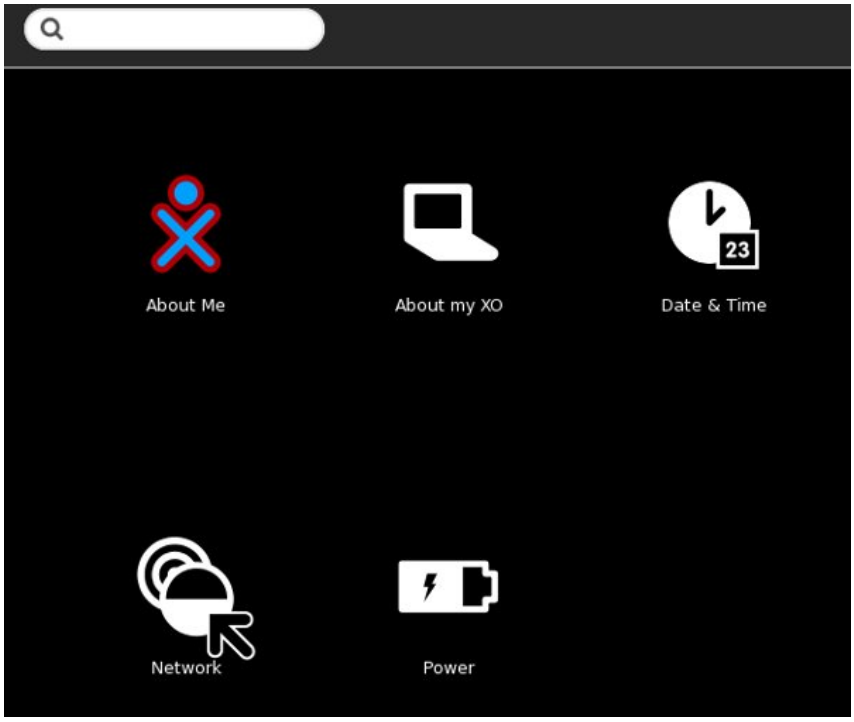
Generally you should use Salut for testing if at all possible. This simplifies testing and doesn't use up resources on a Jabber server.

It does not matter if your Activity connects to others using Gabble or Salut. In fact, the Activity has no idea which it is using. Those details are hidden from the Activity by Telepathy. Any Activity that works with Salut will work with Gabble and vice versa.

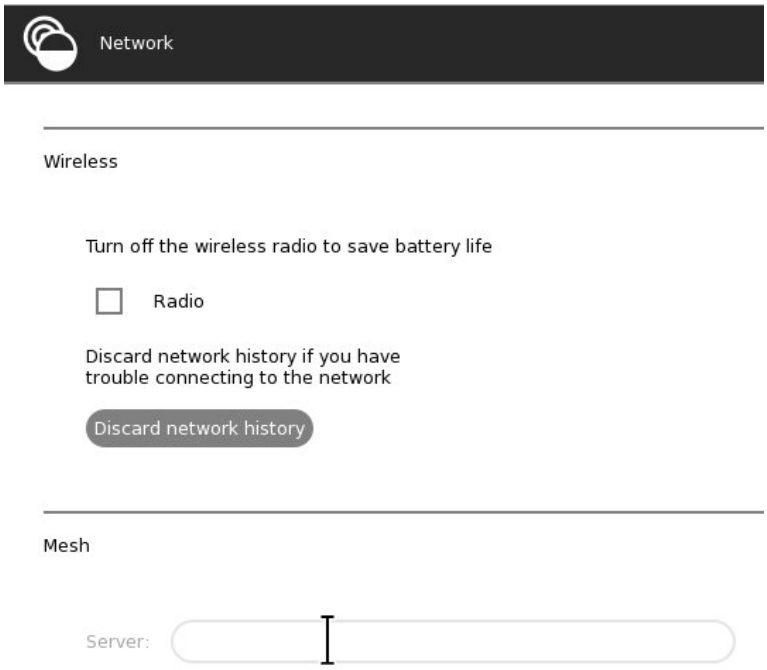
To set up sugar-emulator to use Salut go to the Sugar control panel:



In Sugar .82 this menu option is **Control Panel**. In later versions it is **My Settings**.



Click on the **Network** icon.



The **Server** field in this screen should be empty to use Salut. You can use the backspace key to remove any entry there.

You will need to follow these steps for every Sugar user that will take part in your test.

If for some reason you wish to test your Activity using a Jabber server the OLPC Wiki maintains a list of publicly available servers at http://wiki.laptop.org/go/Community_Jabber_Servers.

Once you have either Salut or a Jabber server set up in both instances of Sugar that you are running you should look at the Neighborhood view of both to see if they can detect each other, and perhaps try out the **Chat** Activity between the two. If you have that working you're ready to try programming a shared Activity.

THE MINICHAT ACTIVITY

Just as we took the **Read Etexts** Activity and stripped it down to the basics we're going to do the same to the **Chat** Activity to create a new Activity called **MiniChat**. The real Chat Activity has a number of features that we don't need to demonstrate shared Activity messaging:

- Chat can connect one to one with a conventional **XMPP** client. This may be useful for Chat but would not be needed or desirable for most shared Activities.

- If you include a URL in a Chat message the user interface enables you to click on the URL make a Journal entry for that URL. You can then use the Journal to open it with the **Browse** Activity. (This is necessary because activities cannot launch each other). Pretty cool, but not needed to demonstrate how to make a shared Activity.
- The chat session is stored in the Journal. When you resume a Chat entry from the Journal it restores the messages from your previous chat session into the user interface. We already know how to save things to the Journal and restore things from the Journal, so MiniChat won't do this.

The resulting code is about half as long as the original. I made a couple of other changes too:

- The text entry field is above the chat messages, instead of below. This makes it easier to do partial screenshots of the Activity in action.
- I took the class **TextChannelWrapper** and put it in its own file. I did this because the class looked like it might be useful for other projects.

The code and all supporting files for **MiniChat** are in the **MiniChat_gtk3** directory of the Git repository. You'll need to run

```
./setup.py dev
```

on the project to make it ready to test. The **activity.info** looks like this:

```
[Activity]
name = Mini Chat
bundle_id = net.flossmanuals.MiniChat
icon = chat
exec = sugar-activity minichat.MiniChat
show_launcher = yes
activity_version = 1
license = GPLv2+
```

Here is the code for **textchannel.py**:

```
import logging

from telepathy.client import Connection, Channel
from telepathy.interfaces import (
    CHANNEL_INTERFACE, CHANNEL_INTERFACE_GROUP,
    CHANNEL_TYPE_TEXT, CONN_INTERFACE_ALIASING)
from telepathy.constants import (
    CHANNEL_GROUP_FLAG_CHANNEL_SPECIFIC_HANDLES,
    CHANNEL_TEXT_MESSAGE_TYPE_NORMAL)

class TextChannelWrapper(object):
    """Wrap a telepathy Text Channel to make
    usage simpler."""
    def __init__(self, text_chan, conn):
        """Connect to the text channel"""
        self._activity_cb = None
        self._activity_close_cb = None
        self._text_chan = text_chan
        self._conn = conn
        self._logger = logging.getLogger(
            'minichat-activity.TextChannelWrapper')
        self._signal_matches = []
        m = self._text_chan[CHANNEL_INTERFACE].\
            connect_to_signal(
            'Closed', self._closed_cb)
        self._signal_matches.append(m)

    def send(self, text):
        """Send text over the Telepathy text channel."""
        # XXX Implement CHANNEL_TEXT_MESSAGE_TYPE_ACTION
        if self._text_chan is not None:
            self._text_chan[CHANNEL_TYPE_TEXT].Send(
                CHANNEL_TEXT_MESSAGE_TYPE_NORMAL, text)

    def close(self):
        """Close the text channel."""
```

```
        self._logger.debug('Closing text channel')
        try:
            self._text_chan[CHANNEL_INTERFACE].Close()
        except:
            self._logger.debug('Channel disappeared!')
            self._closed_cb()

    def _closed_cb(self):
        """Clean up text channel."""
        self._logger.debug('Text channel closed.')
        for match in self._signal_matches:
            match.remove()
        self._signal_matches = []
        self._text_chan = None
        if self._activity_close_cb is not None:
            self._activity_close_cb()

    def set_received_callback(self, callback):
        """Connect the function callback to the signal.

        callback -- callback function taking buddy
        and text args
        """
        if self._text_chan is None:
            return
        self._activity_cb = callback
        m = self._text_chan[CHANNEL_TYPE_TEXT].\
            connect_to_signal(
            'Received', self._received_cb)
        self._signal_matches.append(m)

    def handle_pending_messages(self):
        """Get pending messages and show them as
        received."""
        for id, timestamp, sender, type, flags, text \
            in self._text_chan[
            CHANNEL_TYPE_TEXT].ListPendingMessages(
            False):
            self._received_cb(id, timestamp, sender,
                type, flags, text)

    def _received_cb(self, id, timestamp, sender,
        type, flags, text):
        """Handle received text from the text channel.

        Converts sender to a Buddy.
        Calls self._activity_cb which is a callback
        to the activity.
        """
        if self._activity_cb:
            buddy = self._get_buddy(sender)
            self._activity_cb(buddy, text)
            self._text_chan[
                CHANNEL_TYPE_TEXT].
                AcknowledgePendingMessages([id])
        else:
            self._logger.debug(
                'Throwing received message on the floor'
                ' since there is no callback connected. See '
                'set_received_callback')

    def set_closed_callback(self, callback):
        """Connect a callback for when the text channel
        is closed.

        callback -- callback function taking no args

        """
        self._activity_close_cb = callback

    def _get_buddy(self, cs_handle):
        """Get a Buddy from a (possibly channel-specific)
        handle."""
        # XXX This will be made redundant once Presence
        # Service provides buddy resolution
        from sugar.presence import presenceservice
        # Get the Presence Service
        pservice = presenceservice.get_instance()
        # Get the Telepathy Connection
        tp_name, tp_path = \
            pservice.get_preferred_connection()
        conn = Connection(tp_name, tp_path)
        group = self._text_chan[CHANNEL_INTERFACE_GROUP]
        my_csh = group.GetSelfHandle()
        if my_csh == cs_handle:
            handle = conn.GetSelfHandle()
        elif group.GetGroupFlags() & \
            CHANNEL_GROUP_FLAG_CHANNEL_SPECIFIC_HANDLES:
```

```
            handle = group.GetHandleOwners([cs_handle])[0]
        else:
            handle = cs_handle

            # XXX: deal with failure to get the handle owner
            assert handle != 0

        return pservice.get_buddy_by_telepathy_handle(
            tp_name, tp_path, handle)
```

Here is the code for **minichat.py**:

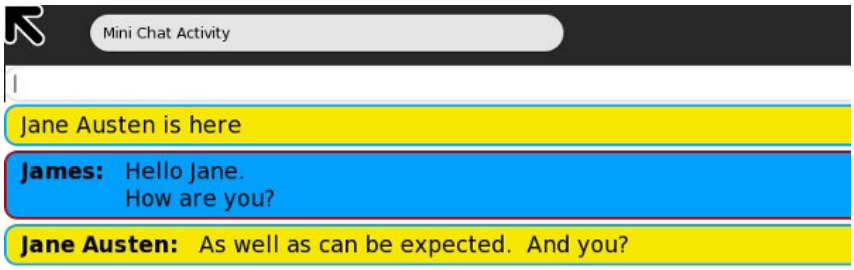
```
from gi.repository import GObject
from gettext import gettext as _
from gi.repository import Gtk
from gi.repository import Gdk
from gi.repository import Pango
import logging
from sugar3.activity.activity import Activity, SCOPE_PRIVATE
from sugar3.activity.widgets import ActivityToolbar, StopButton
from sugar3.graphics.alert import NotifyAlert
from sugar3.presence.presenceservice import PresenceService
from sugar3.graphics.style import (Color, COLOR_BLACK, COLOR_WHITE,
    COLOR_BUTTON_GREY, FONT_BOLD, FONT_NORMAL)
from sugar3.graphics.xocolor import XoColor
from sugar3.graphics.palette import Palette

from textchannel import TextChannelWrapper

logger = logging.getLogger('minichat-activity')

class MiniChat(Activity):
    def __init__(self, handle):
        Activity.__init__(self, handle)

        toolbox = ActivityToolbar(self)

        stop_button = StopButton(self)
        stop_button.show()
        toolbox.insert(stop_button, -1)

        self.set_toolbar_box(toolbox)
        toolbox.show()

        self.scroller = Gtk.ScrolledWindow()
        self.scroller.set_vexpand(True)

        root = self.make_root()
        self.set_canvas(root)
        root.show_all()
        self.entry.grab_focus()

        self.pservice = PresenceService()
        self.owner = self.pservice.get_owner()

        # Track last message, to combine several messages:
        self._last_msg = None
        self._last_msg_sender = None
        self.text_channel = None

        if self.shared_activity:
            # we are joining the activity
            self.connect('joined', self._joined_cb)
            if self.get_shared():
                # we have already joined
                self._joined_cb()
        else:
            # we are creating the activity
            if not self.metadata or self.metadata.get('share-scope',
                    SCOPE_PRIVATE) == SCOPE_PRIVATE:
                # if we are in private session
                self._alert(_('Off-line'), _('Share, or invite
someone.'))
            self.connect('shared', self._shared_cb)

    def _shared_cb(self, activity):
        logger.debug('Chat was shared')
        self._setup()

    def _joined_cb(self, activity):
        """Joined a shared activity."""
        if not self.shared_activity:
            return
        logger.debug('Joined a shared chat')
        for buddy in self.shared_activity.get_joined_buddies():
            self._buddy_already_exists(buddy)
        self._setup()
```

```
    def _setup(self):
        self.text_channel = TextChannelWrapper(
            self.shared_activity.telepathy_text_chan,
            self.shared_activity.telepathy_conn)
        self.text_channel.set_received_callback(self._received_cb)
        self._alert(_('On-line'), _('Connected'))
        self.shared_activity.connect('buddy-joined',
self._buddy_joined_cb)
        self.shared_activity.connect('buddy-left', self._buddy_left_cb)
        self.entry.set_sensitive(True)
        self.entry.grab_focus()

    def _received_cb(self, buddy, text):
        """Show message that was received."""
        if buddy:
            nick = buddy.nick
        else:
            nick = '???'
        logger.debug('Received message from %s: %s', nick, text)
        self.add_text(buddy, text)

    def _alert(self, title, text=None):
        alert = NotifyAlert(timeout=5)
        alert.props.title = title
        alert.props.msg = text
        self.add_alert(alert)
        alert.connect('response', self._alert_cancel_cb)
        alert.show()

    def _alert_cancel_cb(self, alert, response_id):
        self.remove_alert(alert)

    def _buddy_joined_cb (self, activity, buddy):
        """Show a buddy who joined"""
        if buddy == self.owner:
            return
        if buddy:
            nick = buddy.nick
        else:
            nick = '???'
        self.add_text(buddy, buddy.nick+' '+_('joined the chat'),
            status_message=True)

    def _buddy_left_cb (self, activity, buddy):
        """Show a buddy who joined"""
        if buddy == self.owner:
            return
        if buddy:
            nick = buddy.nick
        else:
            nick = '???'
        self.add_text(buddy, buddy.nick+' '+_('left the chat'),
            status_message=True)

    def _buddy_already_exists(self, buddy):
        """Show a buddy already in the chat."""
        if buddy == self.owner:
            return
        if buddy:
            nick = buddy.nick
        else:
            nick = '???'
        self.add_text(buddy, buddy.nick+' '+_('is here'),
            status_message=True)

    def make_root(self):
        vbox = Gtk.VBox()

        self.conversation = Gtk.VBox()
        self.conversation.show_all()
        self.scroller.add_with_viewport(self.conversation)
        self.scroller.override_background_color(Gtk.StateType.NORMAL, \
                                    Gdk.RGBA(*COLOR_WHITE.get_rgba()))
        self.entry = Gtk.Entry()
        self.entry.modify_bg(Gtk.StateType.INSENSITIVE,
                             COLOR_WHITE.get_gdk_color())
        self.entry.modify_base(Gtk.StateType.INSENSITIVE,
                               COLOR_WHITE.get_gdk_color())
        self.entry.set_sensitive(False)

        setattr(self.entry, "nick", "???")
        self.entry.connect('activate', self.entry_activate_cb)

        vbox.pack_start(self.entry, False, False, 0)
        vbox.pack_end(self.scroller, True, True, 0)

        vbox.show()
```

```
        box = Gtk.VBox(homogeneous=False)
        box.pack_end(vbox, False, True, 0)
        box.show_all()

        return box

    def add_text(self, buddy, text, status_message=False):
        """Display text on screen, with name and colors.

        buddy -- buddy object
        text -- string, what the buddy said
        status_message -- boolean
            False: show what buddy said
            True: show what buddy did

        Gtk layout:

        .------------- rb ---------------.
        | +name_vbox+ +----msg_vbox----+ |
        | |         | |                | |
        | | nick:   | | +------------+ | |
        | |         | | |    Text    | | |
        | +---------+ | +------------+ | |
        |             +----------------+ |
        `--------------------------------'

        """
        if buddy:
            nick = buddy.props.nick
            color = buddy.props.color
            try:
                color_stroke_html, color_fill_html = color.split(',')
            except ValueError:
                color_stroke_html, color_fill_html = ('#000000',
'#888888')
            # Select text color based on fill color:
            color_fill_rgba = Color(color_fill_html).get_rgba()
            color_fill_gray = (color_fill_rgba[0] + color_fill_rgba[1]
+
                               color_fill_rgba[2])/3
            color_stroke =
Gdk.RGBA(*Color(color_stroke_html).get_rgba())
            color_fill = Gdk.RGBA(*color_fill_rgba)
            if color_fill_gray < 0.5:
                text_color = Gdk.RGBA(*COLOR_WHITE.get_rgba())
            else:
                text_color = Gdk.RGBA(*COLOR_BLACK.get_rgba())
        else:
            nick = '???'  # XXX: should be '' but leave for debugging

        logger.debug('Nick: %s' % nick)

        # Check for Right-To-Left languages:
        if Pango.find_base_dir(nick, -1) == Pango.Direction.RTL:
            lang_rtl = True
        else:
            lang_rtl = False

        logger.debug('lang_rtl: %s' % str(lang_rtl))

        # Check if new message box or add text to previous:
        new_msg = True
        if self._last_msg_sender:
            if not status_message:
                if buddy == self._last_msg_sender:
                    # Add text to previous message
                    new_msg = False

        if not new_msg:
            rb = self._last_msg
            msg_vbox = rb.get_children()[1]

        else: # Its a new_msg, we need to create a new rb
            # We are using an EventBox to get the border effect on rb
            eb = Gtk.EventBox()
            eb.override_background_color(Gtk.StateType.NORMAL,
color_stroke)

            rb = Gtk.HBox()
            rb.override_background_color(Gtk.StateType.NORMAL,
color_fill)
            rb.set_border_width(1)
            eb.add(rb)

            logger.debug('rb: %s' % str(rb))
            self._last_msg = rb
            self._last_msg_sender = buddy
```

```
            if not status_message:
                name = Gtk.TextView()
                text_buffer = name.get_buffer()
                text_buffer.set_text(nick + ':   ')
                name.override_font(FONT_BOLD.get_pango_desc())
                name.override_color(Gtk.StateType.NORMAL, text_color)
                name.override_background_color(Gtk.StateType.NORMAL, \
                     color_fill)
                name.set_border_width(5)
                name.show()
                name_vbox = Gtk.VBox()
                name_vbox.add(name)
                rb.pack_start(name_vbox, False, True, 0)

            msg_vbox = Gtk.VBox()
            rb.pack_start(msg_vbox, True, True, 0)

        if status_message:
            self._last_msg_sender = None

        if text:
            if not new_msg:
                msg = msg_vbox.get_children()[0]
                text_buffer = msg.get_buffer()
                text_buffer.set_text(text_buffer.get_text( \
                    text_buffer.get_start_iter(),
                    text_buffer.get_end_iter(), True) + "\n" + text)
            else:
                msg = Gtk.TextView()
                text_buffer = msg.get_buffer()
                text_buffer.set_text(text)
                msg.show()
                msg.set_editable(False)
                msg.set_border_width(5)
                msg.set_justification(Gtk.Justification.LEFT)
                msg.override_font(FONT_NORMAL.get_pango_desc())
                msg.override_color(Gtk.StateType.NORMAL, text_color)
                msg.override_background_color(Gtk.StateType.NORMAL, \
                    color_fill)
                msg.set_wrap_mode(Gtk.WrapMode.WORD_CHAR)
                msg_vbox.pack_start(msg, True, True, 0)

        if new_msg:
            self.conversation.pack_start(eb, False, True, 0)
        eb.show_all()
        rb.show_all()

    def entry_activate_cb(self, entry):
        text = entry.get_text()
        entry.set_text('')
        logger.debug('Entry: %s' % text)
        if text:
            self.add_text(self.owner, text)
            if self.text_channel:
                self.text_channel.send(text)
            else:
                logger.debug('Tried to send message but text channel '
                    'not connected.')
```

And this is what the Activity looks like in action:



Try launching more than one copy of sugar-runner, with this Activity installed in each. You'll need to maintain separate copies of the code for each user. In your own projects using a central Git repository at git.sugarlabs.org will make this easy. You just do a git push to copy your changes to the central repository and a git pull to copy them to your second userid. The second userid can use the public URL. There's no need to set up SSH for any user other than the primary one.

Both users of a shared Activity must have the Activity installed. On the other hand, depending on how the Activity is written, two different versions of an Activity may be able to communicate with one another. If the messages they exchange are in the same format there should be no problem.

Once you have both instances of sugar-runner going you can launch MiniChat on one and invite the second user to Join the Chat session. You can do both with the Neighborhood panes of each instance. Making the invitation looks like this:

Accepting it looks like this:

After you've played with **MiniChat** for awhile come back and we'll discuss the secrets of using Telepathy to create a shared Activity.

KNOW WHO YOUR BUDDIES ARE

XMPP, as we said before, is the **Extended Messaging and Presence Protocol**. ⁞ **Presence** is just what it sounds like; it handles letting you know who is available to share your Activity, as well as what other Activities are available to share. There are two ways to share your Activity. The first one is when you change the **Share with:** pulldown on the standard toolbar so it reads **My Neighborhood** instead of **Private**. That means anyone on the network can share your Activity. The other way to share is to go to the Neighborhood view and invite someone specific to share. The person getting the invitation has no idea of the invitation was specifically for him or broadcast to the Neighborhood. The technical term for persons sharing your Activity is **Buddies**. The place where Buddies meet and collaborate is called an **MUC** or **Multi User Chatroom**.

The code used by our Activity for inviting Buddies and joining the Activity as a Buddy is in the *__init__()* method:

```
        if self._shared_activity:
            # we are joining the activity
            self.connect('joined', self._joined_cb)
            if self.get_shared():
                # we have already joined
                self._joined_cb()
        else:
            # we are creating the activity
            if not self.metadata or self.metadata.get(
                'share-scope',
                SCOPE_PRIVATE) == SCOPE_PRIVATE:
                # if we are in private session
                self._alert(_('Off-line'),
                    _('Share, or invite someone.'))
            self.connect('shared', self._shared_cb)

    def _shared_cb(self, activity):
        logger.debug('Chat was shared')
        self._setup()

    def _joined_cb(self, activity):
        """Joined a shared activity."""
        if not self._shared_activity:
            return
        logger.debug('Joined a shared chat')
        for buddy in \
            self._shared_activity.get_joined_buddies():
            self._buddy_already_exists(buddy)
        self._setup()

    def _setup(self):
        self.text_channel = TextChannelWrapper(
            self._shared_activity.telepathy_text_chan,
            self._shared_activity.telepathy_conn)
        self.text_channel.set_received_callback(
            self._received_cb)
        self._alert(_('On-line'), _('Connected'))
        self._shared_activity.connect('buddy-joined',
            self._buddy_joined_cb)
        self._shared_activity.connect('buddy-left',
            self._buddy_left_cb)
        self.entry.set_sensitive(True)
        self.entry.grab_focus()
```

There are two ways to launch an Activity: as the first user of an Activity or by joining an existing Activity. The first line above in **bold** determines whether we are joining or are the first user of the Activity. If so we ask for the *_joined_cb()* method to be run when the 'joined' event occurs. This method gets a buddy list from the *_shared_activity* object and creates messages in the user interface informing the user that these buddies are already in the chat room. Then it runs the *_setup()* method.

If we are not joining an existing Activity then we check to see if we are currently sharing the Activity with anyone. If we aren't we pop up a message telling the user to invite someone to chat. We also request that when the 'shared' even happens the *_shared_cb()* method should run. This method just runs the *_setup()* method.

The *_setup()* method creates a **TextChannelWrapper** object using the code in **textchannel.py**. It also tells the _shared_activity object that it wants some callback methods run when new buddies join the Activity and when existing buddies leave the Activity. Everything you need to know about your buddies can be found in the code above, except how to send messages to them. For that we use the **Text Channel**. There is no need to learn about the Text Channel in great detail because the TextChannelWrapper class does everything you'll ever need to do with the TextChannel and hides the details from you.

```
    def entry_activate_cb(self, entry):
        text = entry.props.text
        logger.debug('Entry: %s' % text)
        if text:
            self.add_text(self.owner, text)
            entry.props.text = ''
            if self.text_channel:
                self.text_channel.send(text)
            else:
                logger.debug(
                    'Tried to send message but text '
                    'channel not connected.')
```

The *add_text()* method is of interest. It takes the owner of the message and figures out what colors belong to that owner and displays the message in those colors. In the case of messages sent by the Activity it gets the owner like this in the *__init__()* method:

```
        self.owner = self._pservice.get_owner()
```

In the case of received messages it gets the buddy the message came from:

```
    def _received_cb(self, buddy, text):
        """Show message that was received."""
        if buddy:
                nick = buddy.props.nick
        else:
            nick = '???'
        logger.debug('Received message from %s: %s',
            nick, text)
        self.add_text(buddy, text)
```

But what if we want to do more than just send text messages back and forth? What do we use for that?

IT'S A SERIES OF TUBES!

No, not the Internet. Telepathy has a concept called **Tubes** which describes the way instances of an Activity can communicate together. What Telepathy does is take the Text Channel and build Tubes on top of it. There are two kinds of Tubes:

- D-Bus Tubes
- Stream Tubes

A **D-Bus Tube** is used to enable one instance of an Activity to call methods in the Buddy instances of the Activity. A **Stream Tube** is used for sending data over **Sockets**, for instance for copying a file from one instance of an Activity to another. A Socket is a way of communicating over a network using Internet Protocols. For instance the HTTP protocol used by the World Wide Web is implemented with Sockets. In the next example we'll use HTTP to transfer books from one instance of **Read Etexts III** to another.

READ ETEXTS III, NOW WITH BOOK SHARING!

The Git repository with the code samples for this book has a file named **ReadEtextsActivity3.py** in the **Making_Shared_Activities** directory which looks like this:

```
import os
import re
import logging
import time
import zipfile
from gi.repository import Gtk
from gi.repository import Pango
from gi.repository import Gdk
import dbus
from gi.repository import GObject
import telepathy
from sugar3.activity import activity
from sugar3.graphics import style
from sugar3.graphics.toolbutton import ToolButton
from sugar3.graphics.toolbarbox import ToolbarButton
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3.activity.widgets import ActivityToolbarButton
from sugar3.activity.widgets import StopButton
from sugar3.activity.widgets import EditToolbar
from sugar3 import network
from sugar3.datastore import datastore
from sugar3.graphics.alert import NotifyAlert
from toolbar import ViewToolbar
from gettext import gettext as _

page=0
PAGE_SIZE = 45
TOOLBAR_READ = 2

logger = logging.getLogger('read-etexts2-activity')

class ReadHTTPRequestHandler(network.ChunkedGlibHTTPRequestHandler):
    """HTTP Request Handler for transferring document while
collaborating.

    RequestHandler class that integrates with Glib mainloop. It writes
    the specified file to the client in chunks, returning control to
the
    mainloop between chunks.

    """
    def translate_path(self, path):
        """Return the filepath to the shared document."""
        return self.server.filepath


class ReadHTTPServer(network.GlibTCPServer):
    """HTTP Server for transferring document while collaborating."""
    def __init__(self, server_address, filepath):
        """Set up the GlibTCPServer with the ReadHTTPRequestHandler.

        filepath -- path to shared document to be served.
        """
        self.filepath = filepath
        network.GlibTCPServer.__init__(self, server_address,
                                       ReadHTTPRequestHandler)


class ReadURLDownloader(network.GlibURLDownloader):
    """URLDownloader that provides content-length and content-type."""

    def get_content_length(self):
        """Return the content-length of the download."""
        if self._info is not None:
            return int(self._info.headers.get('Content-Length'))

    def get_content_type(self):
        """Return the content-type of the download."""
        if self._info is not None:
            return self._info.headers.get('Content-type')
        return None

READ_STREAM_SERVICE = 'read-etexts-activity-http'

class EditToolBarButton(ToolbarButton):
    def __init__(self, toolbar, **kwargs):
        ToolbarButton.__init__(self, page=toolbar, **kwargs)
        self.set_tooltip(_('Edit'))

class ReadEtextsActivity(activity.Activity):
    def __init__(self, handle):
        "The entry point to the Activity"
        global page
```

```
        activity.Activity.__init__(self, handle)

        self.fileserver = None
        self.object_id = handle.object_id

        toolbar_box = ToolbarBox()

        activity_button = ActivityToolbarButton(self)
        toolbar_box.toolbar.insert(activity_button, 0)
        activity_button.show()

        self.edit_toolbar = EditToolbar()
        self.edit_toolbar.undo.props.visible = False
        self.edit_toolbar.redo.props.visible = False
        self.edit_toolbar.separator.props.visible = False
        self.edit_toolbar.copy.set_sensitive(False)
        self.edit_toolbar.copy.connect('clicked',
self.edit_toolbar_copy_cb)
        self.edit_toolbar.paste.props.visible = False
        edit_toolbar_button = ToolbarButton(
            page=self.edit_toolbar,
            icon_name='toolbar-edit')
        self.edit_toolbar.show()
        toolbar_box.toolbar.insert(edit_toolbar_button, -1)
        edit_toolbar_button.show()

        view_toolbar = ViewToolbar()
        view_toolbar.connect('go-fullscreen',
                self.view_toolbar_go_fullscreen_cb)
        view_toolbar.zoom_in.connect('clicked', self.zoom_in_cb)
        view_toolbar.zoom_out.connect('clicked', self.zoom_out_cb)
        view_toolbar.show()
        view_toolbar_button = ToolbarButton(
            page=view_toolbar,
            icon_name='toolbar-view')
        toolbar_box.toolbar.insert(view_toolbar_button, -1)
        view_toolbar_button.show()

        self.back = ToolButton('go-previous')
        self.back.set_tooltip(_('Back'))
        self.back.props.sensitive = False
        self.back.connect('clicked', self.go_back_cb)
        toolbar_box.toolbar.insert(self.back, -1)
        self.back.show()

        self.forward = ToolButton('go-next')
        self.forward.set_tooltip(_('Forward'))
        self.forward.props.sensitive = False
        self.forward.connect('clicked', self.go_forward_cb)
        toolbar_box.toolbar.insert(self.forward, -1)
        self.forward.show()

        num_page_item = Gtk.ToolItem()
        self.num_page_entry = Gtk.Entry()
        self.num_page_entry.set_text('0')
        self.num_page_entry.set_alignment(1)
        self.num_page_entry.connect('insert-text',
                                    self.num_page_entry_insert_text_cb)
        self.num_page_entry.connect('activate',
                                    self.num_page_entry_activate_cb)
        self.num_page_entry.set_width_chars(4)
        num_page_item.add(self.num_page_entry)
        self.num_page_entry.show()
        toolbar_box.toolbar.insert(num_page_item, -1)
        num_page_item.show()

        total_page_item = Gtk.ToolItem()
        self.total_page_label = Gtk.Label()

        self.total_page_label.set_markup("<span foreground='#FFF'" \
                                         " size='14000'></span>")

        self.total_page_label.set_text(' / 0')
        total_page_item.add(self.total_page_label)
        self.total_page_label.show()
        toolbar_box.toolbar.insert(total_page_item, -1)
        total_page_item.show()

        separator = Gtk.SeparatorToolItem()
        separator.props.draw = False
        separator.set_expand(True)
        toolbar_box.toolbar.insert(separator, -1)
        separator.show()

        stop_button = StopButton(self)
        stop_button.props.accelerator = '<Ctrl><Shift>Q'
        toolbar_box.toolbar.insert(stop_button, -1)
        stop_button.show()
```

```
        self.set_toolbar_box(toolbar_box)
        toolbar_box.show()

        self.scrolled_window = Gtk.ScrolledWindow()
        self.scrolled_window.set_policy(Gtk.PolicyType.NEVER,
Gtk.PolicyType.AUTOMATIC)

        self.textview = Gtk.TextView()
        self.textview.set_editable(False)
        self.textview.set_cursor_visible(False)
        self.textview.set_left_margin(50)
        self.textview.connect("key_press_event", self.keypress_cb)

        self.progressbar = Gtk.ProgressBar()
        self.progressbar.set_fraction(0.0)

        self.scrolled_window.add(self.textview)
        self.textview.show()
        self.scrolled_window.show()

        vbox = Gtk.VBox()
        vbox.pack_start(self.progressbar,  False,  False,  10)
        vbox.pack_start(self.scrolled_window, True, True, 0)
        self.set_canvas(vbox)
        vbox.show()

        page = 0
        self.clipboard = Gtk.Clipboard.get(Gdk.SELECTION_CLIPBOARD)
        self.textview.grab_focus()
        self.font_desc = Pango.FontDescription("sans %d" %
style.zoom(10))
        self.textview.modify_font(self.font_desc)

        buffer = self.textview.get_buffer()
        self.markset_id = buffer.connect("mark-set", self.mark_set_cb)
        self.unused_download_tubes = set()
        self.want_document = True
        self.download_content_length = 0
        self.download_content_type = None
        # Status of temp file used for write_file:
        self.tempfile = None
        self.close_requested = False
        self.connect("shared", self.shared_cb)
        self.is_received_document = False

        if self.shared_activity and handle.object_id == None:
            # We're joining, and we don't already have the document.
            if self.get_shared():
                # Already joined for some reason, just get the
document
                self.joined_cb(self)
            else:
                # Wait for a successful join before trying to get the
                # document
                self.connect("joined", self.joined_cb)

    def num_page_entry_insert_text_cb(self, entry, text, length,
position):
        if not re.match('[0-9]', text):
            entry.emit_stop_by_name('insert-text')
            return True
        return False

    def num_page_entry_activate_cb(self, entry):
        global page
        if entry.props.text:
            new_page = int(entry.props.text) - 1
        else:
            new_page = 0

        if new_page >= self.total_pages:
            new_page = self.total_pages - 1
        elif new_page < 0:
            new_page = 0

        self.current_page = new_page
        self.set_current_page(new_page)
        self.show_page(new_page)
        entry.props.text = str(new_page + 1)
        self.update_nav_buttons()
        page = new_page

    def update_nav_buttons(self):
        current_page = self.current_page
        self.back.props.sensitive = current_page > 0
        self.forward.props.sensitive = \
            current_page < self.total_pages - 1
```

```
        self.num_page_entry.props.text = str(current_page + 1)
        self.total_page_label.props.label = \
            ' / ' + str(self.total_pages)

    def set_total_pages(self, pages):
        self.total_pages = pages

    def set_current_page(self, page):
        self.current_page = page
        self.update_nav_buttons()

    def keypress_cb(self, widget, event):
        "Respond when the user presses one of the arrow keys"
        keyname = Gdk.keyval_name(event.keyval)
        print keyname
        if keyname == 'plus':
            self.font_increase()
            return True
        if keyname == 'minus':
            self.font_decrease()
            return True
        if keyname == 'Page_Up' :
            self.page_previous()
            return True
        if keyname == 'Page_Down':
            self.page_next()
            return True
        if keyname == 'Up' or keyname == 'KP_Up' \
                or keyname == 'KP_Left':
            self.scroll_up()
            return True
        if keyname == 'Down' or keyname == 'KP_Down' \
                or keyname == 'KP_Right':
            self.scroll_down()
            return True
        return False

    def go_back_cb(self, button):
        self.page_previous()

    def go_forward_cb(self, button):
        self.page_next()

    def page_previous(self):
        global page
        page=page-1
        if page < 0: page=0
        self.set_current_page(page)
        self.show_page(page)
        v_adjustment = self.scrolled_window.get_vadjustment()
        v_adjustment.set_value(v_adjustment.get_upper() - \
                               v_adjustment.get_page_size())

    def page_next(self):
        global page
        page=page+1
        if page >= len(self.page_index): page=0
        self.set_current_page(page)
        self.show_page(page)
        v_adjustment = self.scrolled_window.get_vadjustment()
        v_adjustment.set_value(v_adjustment.get_lower())

    def zoom_in_cb(self,  button):
        self.font_increase()

    def zoom_out_cb(self,  button):
        self.font_decrease()

    def font_decrease(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size - 1
        if font_size < 1:
            font_size = 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def font_increase(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size + 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def mark_set_cb(self, textbuffer, iter, textmark):

        if textbuffer.get_has_selection():
            begin, end = textbuffer.get_selection_bounds()
            self.edit_toolbar.copy.set_sensitive(True)
```

```
        else:
            self.edit_toolbar.copy.set_sensitive(False)

    def edit_toolbar_copy_cb(self, button):
        textbuffer = self.textview.get_buffer()
        textbuffer.copy_clipboard(self.clipboard)

    def view_toolbar_go_fullscreen_cb(self, view_toolbar):
        self.fullscreen()

    def scroll_down(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.get_value() == v_adjustment.get_upper() - \
                v_adjustment.get_page_size():
            self.page_next()
            return
        if v_adjustment.get_value() < v_adjustment.get_upper() - \
                v_adjustment.get_page_size():
            new_value = v_adjustment.get_value() +
v_adjustment.step_increment
            if new_value > v_adjustment.get_upper() -
v_adjustment.get_page_size():
                new_value = v_adjustment.get_upper() -
v_adjustment.get_page_size()
            v_adjustment.set_value(new_value)

    def scroll_up(self):
        v_adjustment = self.scrolled_window.get_vadjustment()
        if v_adjustment.get_value() == v_adjustment.get_lower():
            self.page_previous()
            return
        if v_adjustment.get_value() > v_adjustment.get_lower():
            new_value = v_adjustment.get_value() - \
                v_adjustment.step_increment
            if new_value < v_adjustment.get_lower():
                new_value = v_adjustment.get_lower()
            v_adjustment.set_value(new_value)

    def show_page(self, page_number):
        global PAGE_SIZE, current_word
        position = self.page_index[page_number]
        self.etext_file.seek(position)
        linecount = 0
        label_text = '\n\n\n'
        textbuffer = self.textview.get_buffer()
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line, 'iso-8859-1')
            linecount = linecount + 1
        label_text = label_text + '\n\n\n'
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        filebytes = zipfile.read(filename)
        outfn = self.make_new_filename(filename)
        if (outfn == ''):
            return False
        f = open(os.path.join(self.get_activity_root(), 'tmp', outfn),
'w')
        try:
            f.write(filebytes)
        finally:
            f.close()

    def get_saved_page_number(self):
        global page
        title = self.metadata.get('title', '')
        if title == ''  or not title[len(title)- 1].isdigit():
            page = 0
        else:
            i = len(title) - 1
            newPage = ''
            while (title[i].isdigit() and i > 0):
                newPage = title[i] + newPage
                i = i - 1
            if title[i] == 'P':
                page = int(newPage) - 1
            else:
                # not a page number; maybe a volume number.
                page = 0

    def save_page_number(self):
        global page
        title = self.metadata.get('title', '')
        if title == ''  or not title[len(title)- 1].isdigit():
            title = title + ' P' +  str(page + 1)
```

```
        else:
            i = len(title) - 1
            while (title[i].isdigit() and i > 0):
                i = i - 1
            if title[i] == 'P':
                title = title[0:i] + 'P' + str(page + 1)
            else:
                title = title + ' P' + str(page + 1)
        self.metadata['title'] = title

    def read_file(self, filename):
        "Read the Etext file"
        global PAGE_SIZE,  page

        tempfile = os.path.join(self.get_activity_root(),  'instance',
\
                'tmp%i' % time.time())
        os.link(filename,  tempfile)
        self.tempfile = tempfile

        if zipfile.is_zipfile(filename):
            self.zf = zipfile.ZipFile(filename, 'r')
            self.book_files = self.zf.namelist()
            self.save_extracted_file(self.zf, self.book_files[0])
            currentFileName = os.path.join(self.get_activity_root(), \
                    'tmp',  self.book_files[0])
        else:
            currentFileName = filename

        self.etext_file = open(currentFileName,"r")
        self.page_index = [ 0 ]
        pagecount = 0
        linecount = 0
        while self.etext_file:
            line = self.etext_file.readline()
            if not line:
                break
            linecount = linecount + 1
            if linecount >= PAGE_SIZE:
                position = self.etext_file.tell()
                self.page_index.append(position)
                linecount = 0
                pagecount = pagecount + 1
        if filename.endswith(".zip"):
            os.remove(currentFileName)
        self.get_saved_page_number()
        self.show_page(page)
        self.set_total_pages(pagecount + 1)
        self.set_current_page(page)

        # We've got the document, so if we're a shared activity, offer
it
        if self.get_shared():
            self.watch_for_tubes()
            self.share_document()

    def make_new_filename(self, filename):
        partition_tuple = filename.rpartition('/')
        return partition_tuple[2]

    def write_file(self, filename):
        "Save meta data for the file."
        if self.is_received_document:
            # This document was given to us by someone, so we have
            # to save it to the Journal.
            self.etext_file.seek(0)
            filebytes = self.etext_file.read()
            f = open(filename, 'wb')
            try:
                f.write(filebytes)
            finally:
                f.close()
        elif self.tempfile:
            if self.close_requested:
                os.link(self.tempfile,  filename)
                logger.debug("Removing temp file %s because we will
close", \
                             self.tempfile)
                os.unlink(self.tempfile)
                self.tempfile = None
        else:
            # skip saving empty file
            raise NotImplementedError

        self.metadata['activity'] = self.get_bundle_id()
        self.save_page_number()

    def can_close(self):
```

```
        self.close_requested = True
        return True

    def joined_cb(self, also_self):
        """Callback for when a shared activity is joined.

        Get the shared document from another participant.
        """
        self.watch_for_tubes()
        GObject.idle_add(self.get_document)

    def get_document(self):
        if not self.want_document:
            return False

        # Assign a file path to download if one doesn't exist yet
        if not self._jobject.file_path:
            path = os.path.join(self.get_activity_root(), 'instance',
                                'tmp%i' % time.time())
        else:
            path = self._jobject.file_path

        # Pick an arbitrary tube we can try to download the document
from
        try:
            tube_id = self.unused_download_tubes.pop()
        except (ValueError, KeyError), e:
            logger.debug('No tubes to get the document from right now:
%s',
                         e)
            return False

        # Avoid trying to download the document multiple times at once
        self.want_document = False
        GObject.idle_add(self.download_document, tube_id, path)
        return False

    def download_document(self, tube_id, path):
        chan = self._shared_activity.telepathy_tubes_chan
        iface = chan[telepathy.CHANNEL_TYPE_TUBES]
        addr = iface.AcceptStreamTube(tube_id,
                telepathy.SOCKET_ADDRESS_TYPE_IPV4,
                telepathy.SOCKET_ACCESS_CONTROL_LOCALHOST, 0,
                utf8_strings=True)
        logger.debug('Accepted stream tube: listening address is %r', \
                     addr)
        assert isinstance(addr, dbus.Struct)
        assert len(addr) == 2
        assert isinstance(addr[0], str)
        assert isinstance(addr[1], (int, long))
        assert addr[1] > 0 and addr[1] < 65536
        port = int(addr[1])

        self.progressbar.show()
        getter = ReadURLDownloader("http://%s:%d/document"
                                   % (addr[0], port))
        getter.connect("finished", self.download_result_cb, tube_id)
        getter.connect("progress", self.download_progress_cb, tube_id)
        getter.connect("error", self.download_error_cb, tube_id)
        logger.debug("Starting download to %s...", path)
        getter.start(path)
        self.download_content_length = getter.get_content_length()
        self.download_content_type = getter.get_content_type()
        return False

    def download_progress_cb(self, getter, bytes_downloaded, tube_id):
        if self.download_content_length > 0:
            logger.debug("Downloaded %u of %u bytes from tube %u...",
                         bytes_downloaded,
self.download_content_length,
                         tube_id)
        else:
            logger.debug("Downloaded %u bytes from tube %u...",
                         bytes_downloaded, tube_id)
        total = self.download_content_length
        self.set_downloaded_bytes(bytes_downloaded,  total)
        Gdk.threads_enter()
        while Gtk.events_pending():
            Gtk.main_iteration()
        Gdk.threads_leave()

    def set_downloaded_bytes(self, bytes,  total):
        fraction = float(bytes) / float(total)
        self.progressbar.set_fraction(fraction)
        logger.debug("Downloaded percent",  fraction)

    def clear_downloaded_bytes(self):
        self.progressbar.set_fraction(0.0)
```

```
        logger.debug("Cleared download bytes")

    def download_error_cb(self, getter, err, tube_id):
        self.progressbar.hide()
        logger.debug("Error getting document from tube %u: %s",
                     tube_id, err)
        self.alert(_('Failure'), _('Error getting document from tube'))
        self.want_document = True
        self.download_content_length = 0
        self.download_content_type = None
        GObject.idle_add(self.get_document)

    def download_result_cb(self, getter, tempfile, suggested_name,
tube_id):
        if self.download_content_type.startswith('text/html'):
            # got an error page instead
            self.download_error_cb(getter, 'HTTP Error', tube_id)
            return

        del self.unused_download_tubes

        self.tempfile = tempfile
        file_path = os.path.join(self.get_activity_root(), 'instance',
                                 '%i' % time.time())
        logger.debug("Saving file %s to datastore...", file_path)
        os.link(tempfile, file_path)
        self._jobject.file_path = file_path
        datastore.write(self._jobject, transfer_ownership=True)

        logger.debug("Got document %s (%s) from tube %u",
                     tempfile, suggested_name, tube_id)
        self.is_received_document = True
        self.read_file(tempfile)
        self.save()
        self.progressbar.hide()

    def shared_cb(self, activityid):
        """Callback when activity shared.

        Set up to share the document.

        """
        # We initiated this activity and have now shared it, so by
        # definition we have the file.
        logger.debug('Activity became shared')
        self.watch_for_tubes()
        self.share_document()

    def share_document(self):
        """Share the document."""
        h = hash(self._activity_id)
        port = 1024 + (h % 64511)
        logger.debug('Starting HTTP server on port %d', port)
        self.fileserver = ReadHTTPServer(("", port),
            self.tempfile)

        # Make a tube for it
        chan = self._shared_activity.telepathy_tubes_chan
        iface = chan[telepathy.CHANNEL_TYPE_TUBES]
        self.fileserver_tube_id =
iface.OfferStreamTube(READ_STREAM_SERVICE,
                {},
                telepathy.SOCKET_ADDRESS_TYPE_IPV4,
                ('127.0.0.1', dbus.UInt16(port)),
                telepathy.SOCKET_ACCESS_CONTROL_LOCALHOST, 0)

    def watch_for_tubes(self):
        """Watch for new tubes."""
        tubes_chan = self._shared_activity.telepathy_tubes_chan


tubes_chan[telepathy.CHANNEL_TYPE_TUBES].connect_to_signal('NewTube',
            self.new_tube_cb)
        tubes_chan[telepathy.CHANNEL_TYPE_TUBES].ListTubes(
            reply_handler=self.list_tubes_reply_cb,
            error_handler=self.list_tubes_error_cb)

    def new_tube_cb(self, tube_id, initiator, tube_type, service,
params,
                    state):
        """Callback when a new tube becomes available."""
        logger.debug('New tube: ID=%d initator=%d type=%d service=%s '
                     'params=%r state=%d', tube_id, initiator,
tube_type,
                     service, params, state)
        if service == READ_STREAM_SERVICE:
            logger.debug('I could download from that tube')
            self.unused_download_tubes.add(tube_id)
```

```
            # if no download is in progress, let's fetch the document
            if self.want_document:
                GObject.idle_add(self.get_document)

    def list_tubes_reply_cb(self, tubes):
        """Callback when new tubes are available."""
        for tube_info in tubes:
            self.new_tube_cb(*tube_info)

    def list_tubes_error_cb(self, e):
        """Handle ListTubes error by logging."""
        logger.error('ListTubes() failed: %s', e)

    def alert(self, title, text=None):
        alert = NotifyAlert(timeout=20)
        alert.props.title = title
        alert.props.msg = text
        self.add_alert(alert)
        alert.connect('response', self.alert_cancel_cb)
        alert.show()

    def alert_cancel_cb(self, alert, response_id):
        self.remove_alert(alert)
        self.textview.grab_focus()
```

The contents of **activity.info** are these lines:

```
[Activity]
name = ReadEtexts III
bundle_id = net.flossmanuals.ReadETextsActivity3
icon = read-etexts
exec = sugar-activity ReadEtextsActivity3.ReadEtextsActivity
show_launcher = no
mime_types = text/plain;application/zip
activity_version = 1
license = GPLv2+
```

To try it out, download a *Project Gutenberg* book to the Journal, open it with this latest **Read Etexts III**, then share it with a second user who has the program installed but not running. She should accept the invitation to join that appears in her Neighborhood view. When she does Read Etexts II will start up and copy the book from the first user over the network and load it. The Activity will first show a blank screen, but then a progress bar will appear just under the toolbar and indicate the progress of the copying. When it is finished the first page of the book will appear.

So how does it work? Let's look at the code. The first points of interest are the class definitions that appear at the beginning: **ReadHTTPRequestHandler**, **ReadHTTPServer**, and **ReadURLDownloader**. These three classes extend (that is to say, inherit code from) classes provided by the **sugar.network** package. These classes provide an **HTTP client** for receiving the book and an **HTTP Server** for sending the book.

This is the code used to send a book:

```
    def shared_cb(self, activityid):
        """Callback when activity shared.

        Set up to share the document.

        """
        # We initiated this activity and have now shared it,
        # so by definition we have the file.
        logger.debug('Activity became shared')
        self.watch_for_tubes()
        self.share_document()

    def share_document(self):
        """Share the document."""
        h = hash(self._activity_id)
        port = 1024 + (h % 64511)
        logger.debug(
            'Starting HTTP server on port %d', port)
        self.fileserver = ReadHTTPServer(("", port),
            self.tempfile)

        # Make a tube for it
        chan = self._shared_activity.telepathy_tubes_chan
```

```
        iface = chan[telepathy.CHANNEL_TYPE_TUBES]
        self.fileserver_tube_id = iface.OfferStreamTube(
            READ_STREAM_SERVICE,
            {},
            telepathy.SOCKET_ADDRESS_TYPE_IPV4,
            ('127.0.0.1', dbus.UInt16(port)),
            telepathy.SOCKET_ACCESS_CONTROL_LOCALHOST,
            0)
```

You will notice that a hash of the *_activity_id* is used to get a port number. That port is used for the HTTP server and is passed to Telepathy, which offers it as a **Stream Tube**. On the receiving side we have this code:

```
    def joined_cb(self, also_self):
        """Callback for when a shared activity is joined.

        Get the shared document from another participant.
        """
        self.watch_for_tubes()
        gobject.idle_add(self.get_document)

    def get_document(self):
        if not self.want_document:
            return False

        # Assign a file path to download if one doesn't
        # exist yet
        if not self._jobject.file_path:
            path = os.path.join(self.get_activity_root(),
                'instance',
                'tmp%i' % time.time())
        else:
            path = self._jobject.file_path

        # Pick an arbitrary tube we can try to download the
        # document from
        try:
            tube_id = self.unused_download_tubes.pop()
        except (ValueError, KeyError), e:
            logger.debug(
                'No tubes to get the document from '
                'right now: %s',
                e)
            return False

        # Avoid trying to download the document multiple
        # times at once
        self.want_document = False
        gobject.idle_add(self.download_document,
            tube_id, path)
        return False

    def download_document(self, tube_id, path):
        chan = self._shared_activity.telepathy_tubes_chan
        iface = chan[telepathy.CHANNEL_TYPE_TUBES]
        addr = iface.AcceptStreamTube(tube_id,
            telepathy.SOCKET_ADDRESS_TYPE_IPV4,
            telepathy.SOCKET_ACCESS_CONTROL_LOCALHOST,
            0,
            utf8_strings=True)
        logger.debug(
            'Accepted stream tube: listening address is %r',
            addr)
        assert isinstance(addr, dbus.Struct)
        assert len(addr) == 2
        assert isinstance(addr[0], str)
        assert isinstance(addr[1], (int, long))
        assert addr[1] > 0 and addr[1] < 65536
        port = int(addr[1])

        self.progressbar.show()
        getter = ReadURLDownloader(
            "http://%s:%d/document"
            % (addr[0], port))
        getter.connect("finished",
            self.download_result_cb, tube_id)
        getter.connect("progress",
            self.download_progress_cb, tube_id)
        getter.connect("error",
            self.download_error_cb, tube_id)
        logger.debug(
            "Starting download to %s...", path)
        getter.start(path)
        self.download_content_length = \
            getter.get_content_length()
```

```
        self.download_content_type = \
            getter.get_content_type()
        return False

    def download_progress_cb(self, getter,
        bytes_downloaded, tube_id):
        if self.download_content_length > 0:
            logger.debug(
                "Downloaded %u of %u bytes from tube %u...",
                bytes_downloaded,
                self.download_content_length,
                tube_id)
        else:
            logger.debug(
                "Downloaded %u bytes from tube %u...",
                bytes_downloaded, tube_id)
        total = self.download_content_length
        self.set_downloaded_bytes(bytes_downloaded,
            total)
        gtk.gdk.threads_enter()
        while gtk.events_pending():
            gtk.main_iteration()
        gtk.gdk.threads_leave()

    def download_error_cb(self, getter, err, tube_id):
        self.progressbar.hide()
        logger.debug(
            "Error getting document from tube %u: %s",
            tube_id, err)
        self.alert(_('Failure'),
            _('Error getting document from tube'))
        self.want_document = True
        self.download_content_length = 0
        self.download_content_type = None
        gobject.idle_add(self.get_document)

    def download_result_cb(self, getter, tempfile,
        suggested_name, tube_id):
        if self.download_content_type.startswith(
            'text/html'):
            # got an error page instead
            self.download_error_cb(getter,
                'HTTP Error', tube_id)
            return

        del self.unused_download_tubes

        self.tempfile = tempfile
        file_path = os.path.join(self.get_activity_root(),
            'instance',
            '%i' % time.time())
        logger.debug(
            "Saving file %s to datastore...", file_path)
        os.link(tempfile, file_path)
        self._jobject.file_path = file_path
        datastore.write(self._jobject,
            transfer_ownership=True)

        logger.debug(
            "Got document %s (%s) from tube %u",
            tempfile, suggested_name, tube_id)
        self.is_received_document = True
        self.read_file(tempfile)
        self.save()
        self.progressbar.hide()
```

Telepathy gives us the address and port number associated with a Stream Tube and we set up the HTTP Client to read from it. The client reads the file in chunks and calls *download_progress_cb()* after every chunk so we can update a progress bar to show the user how the download is progressing. There are also callback methods for when there is a download error and for when the download is finished,

The **ReadURLDownloader** class is not only useful for transferring files over Stream Tubes, it can also be used to interact with websites and web services. My Activity **Get Internet Archive Books** uses this class for that purpose.

The one remaining piece is the code which handles getting Stream Tubes to download the book from. In this code, adapted from the **Read** Activity, as soon as an instance of an Activity receives a book it turns around and offers to share it, thus the Activity may have several possible Tubes it could get the book from:

```
READ_STREAM_SERVICE = 'read-etexts-activity-http'

    ...

    def watch_for_tubes(self):
        """Watch for new tubes."""
        tubes_chan = self._shared_activity.\
            telepathy_tubes_chan

        tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            connect_to_signal(
            'NewTube',
            self.new_tube_cb)
        tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            ListTubes(
            reply_handler=self.list_tubes_reply_cb,
            error_handler=self.list_tubes_error_cb)

    def new_tube_cb(self, tube_id, initiator,
        tube_type, service, params, state):
        """Callback when a new tube becomes available."""
        logger.debug(
            'New tube: ID=%d initator=%d type=%d service=%s '
            'params=%r state=%d', tube_id, initiator,
            tube_type,
            service, params, state)
        if service == READ_STREAM_SERVICE:
            logger.debug('I could download from that tube')
            self.unused_download_tubes.add(tube_id)
            # if no download is in progress,
            # let's fetch the document
            if self.want_document:
                gobject.idle_add(self.get_document)

    def list_tubes_reply_cb(self, tubes):
        """Callback when new tubes are available."""
        for tube_info in tubes:
            self.new_tube_cb(*tube_info)

    def list_tubes_error_cb(self, e):
        """Handle ListTubes error by logging."""
        logger.error('ListTubes() failed: %s', e)
```

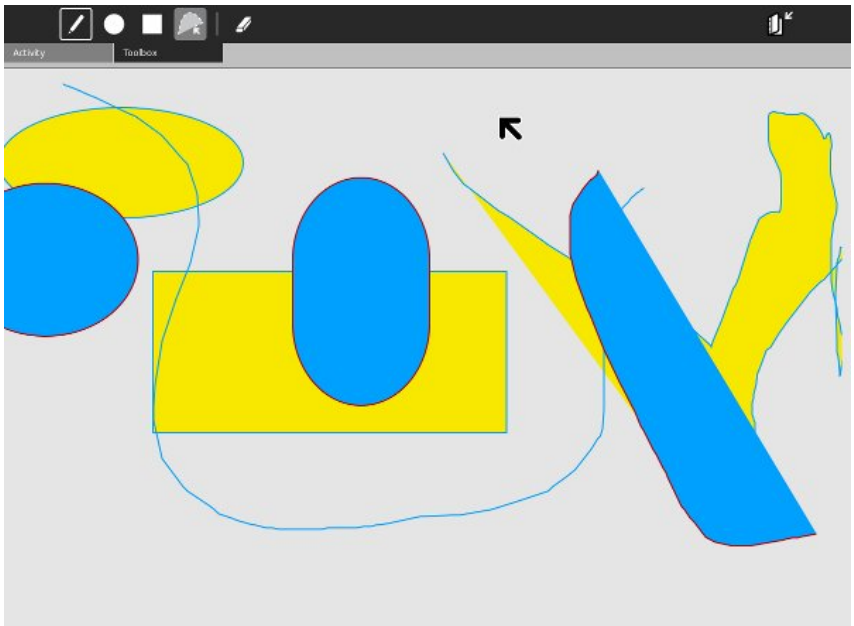
The **READ_STREAM_SERVICE** constant is defined near the top of the source file.

USING D-BUS TUBES

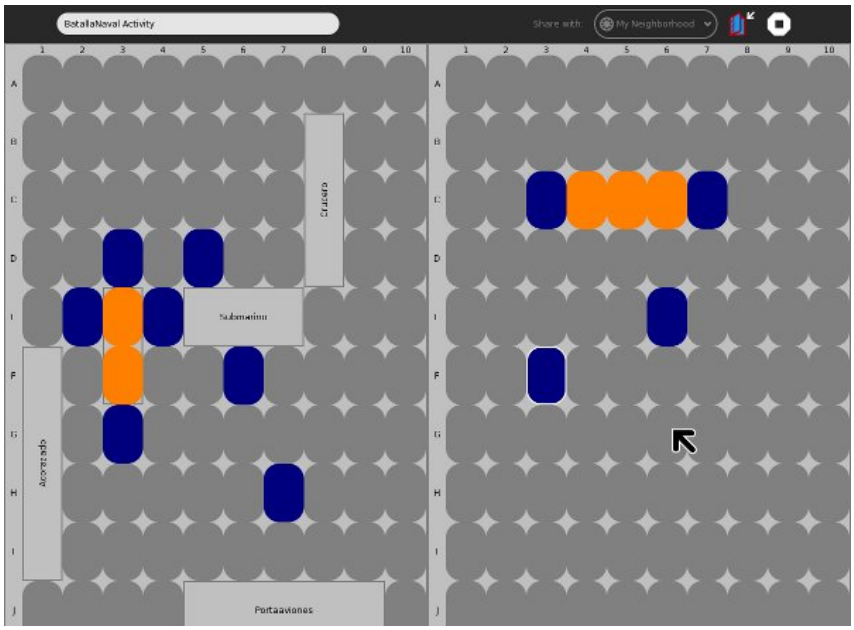
D-Bus is a method of supporting **IPC**, or **Inter-Process Communication**, that was created for the GNOME desktop environment. The idea of IPC is to allow two running programs to communicate with each other and execute each other's code. GNOME uses D-Bus to provide communication between the desktop environment and the programs running in it, and also between GNOME and the operating system. A **D-Bus Tube** is how Telepathy makes it possible for an instance of an Activity running on one computer to execute methods in another instance of the same Activity running on a different computer. Instead of just sending simple text messages back and forth or doing file transfers, your Activities can be truly shared. That is, your Activity can allow many people to work on the same task together.

I have never written an Activity that uses D-Bus Tubes myself, but many others have. We're going to take a look at code from two of them: **Scribble** by Sayamindu Dasgupta and **Batalla Naval**, by Gerard J. Cerchio and Andrés Ambrois, which was written for the Ceibal Jam.

Scribble is a drawing program that allows many people to work on the same drawing at the same time. Instead of allowing you to choose what colors you will draw with, it uses the background and foreground colors of your Buddy icon (the XO stick figure) to draw with. That way, with many people drawing shapes it's easy to know who drew what. If you join the Activity in progress Scribble will update your screen so your drawing matches everyone else's screen. Scribble in action looks like this:



Batalla Naval is a version of the classic game *Battleship*. Each player has two grids: one for placing his own ships (actually the computer places the ships for you) and another blank grid representing the area where your opponent's ships are. You can't see his ships and he can't see yours. You click on the opponent's grid (on the right) to indicate where you want to aim an artillery shell. When you do the corresponding square will light up in both your grid and your opponent's own ship grid. If the square you picked corresponds to a square where your opponent has placed a ship that square will show up in a different color. The object is to find the squares containing your opponent's ships before he finds yours. The game in action looks like this:



I suggest that you download the latest code for these two Activities from Gitorious using these commands:

```
mkdir scribble
cd scribble
git clone git://git.sugarlabs.org/scribble/mainline.git
cd ..
mkdir batallanaval
cd batallanaval
git clone git://git.sugarlabs.org/batalla-naval/mainline.git
```

You'll need to do some setup work to get these running in sugar-runner. Scribble requires the **goocanvas** GTK component and the Python bindings that go with it. These may not be installed by default but you should be able to install them using **Add/Remove Programs** from the **System** menu in GNOME. Batalla Naval is missing **setup.py**, but that's easily fixed since every setup.py is identical. Copy the one from the book examples into the **mainline/BatallaNaval.activity** directory and run **./setup.py dev** on both Activities.

These Activities use different strategies for collaboration. Scribble creates lines of Python code which it passes to all Buddies and the Buddies use **exec** to run the commands. This is the code used for drawing a circle:

```
    def process_item_finalize(self, x, y):
        if self.tool == 'circle':
            self.cmd = "goocanvas.Ellipse(
                parent=self._root,
                center_x=%d,
                center_y=%d, radius_x = %d,
                radius_y = %d,
                fill_color_rgba = %d,
                stroke_color_rgba = %d,
                title = '%s')" % (self.item.props.center_x,
                self.item.props.center_y,
                self.item.props.radius_x,
                self.item.props.radius_y,
                self._fill_color,
                self._stroke_color, self.item_id)
...

    def process_cmd(self, cmd):
        #print 'Processing cmd :' + cmd
        exec(cmd)
        #FIXME: Ugly hack, but I'm too lazy to
        # do this nicely

        if len(self.cmd_list) > 0:
```

```
            self.cmd_list += (';' + cmd)
        else:
            self.cmd_list = cmd
```

The **cmd_list** variable is used to create a long string containing all of the commands executed so far. When a new Buddy joins the Activity she is sent this variable to execute so that her drawing area has the same contents as the other Buddies have.

This is an interesting approach but you could do the same thing with the TextChannel so it isn't the best use of D-Bus Tubes. Batalla Naval's use of D-Bus is more typical.

HOW D-BUS TUBES WORK, MORE OR LESS

D-Bus enables you to have two running programs send messages to each other. The programs have to be running on the same computer. Sending a message is sort of a roundabout way of having one program run code in another. A program defines the kind of messages it is willing to receive and act on. In the case of Batalla Naval it defines a message "tell me what square you want to fire a shell at and I'll figure out if part of one of my ships is in that square and tell you." The first program doesn't actually run code in the second one, but the end result is similar. D-Bus Tubes is a way of making D-Bus able to send messages like this to a program running on another computer.

Think for a minute about how you might make a program on one computer run code in a running program on a different computer. You'd have to use the network, of course. Everyone is familiar with sending data over a network, but in this case you would have to send program code over the network. You would need to be able to tell the running program on the second computer what code you wanted it to run. You would have to send it a method call and all the parameters you needed to pass into the method, and you'd need a way to get a return value back.

Isn't that kind of like what **Scribble** is doing in the code we just looked at? Maybe we could make our code do something like that?

Of course if you did that then every program you wanted to run code in remotely would have to be written to deal with that. If you had a bunch of programs you wanted to do that with you'd have to have some way of letting the programs know which requests were meant for it. It would be nice if there was a program running on each machine that dealt with making the network connections, converting method calls to data that could be sent over the network and then converting the data back into method calls and running them, plus sending any return values back. This program should be able to know which program you wanted to run code in and see that the method call is run there. The program should run all the time, and it would be really good if it made running a method on a remote program as simple as running a method in my own program.

As you might guess, what I've just described is more or less what D-Bus Tubes are. There are articles explaining how it works in detail but it is not necessary to know how it works to use it. You do need to know about a few things, though. First, you need to know how to use D-Bus Tubes to make objects in your Activity available for use by other instances of that Activity running elsewhere.

An Activity that needs to use D-Bus Tubes needs to define what sorts of messages it is willing to act on, in effect what specific methods in in the program are available for this use. All Activities that use D-Bus Tubes have constants like this:

```
SERVICE = "org.randomink.sayamindu.Scribble"
IFACE = SERVICE
PATH = "/org/randomink/sayamindu/Scribble"
```

These are the constants used for the **Scribble** Activity. The first constant, named SERVICE, represents the **bus name** of the Activity. This is also called a **well-known name** because it uses a **reversed domain name** as part of the name. In this case Sayamindu Dasgupta has a website at http://sayamindu.randomink.org so he reverses the dot-separated words of that URL to create the first part of his bus name. It is not necessary to own a domain name before you can create a bus name. You can use org.sugarlabs.ActivityName if you like. The point is that the bus name must be unique, and by convention this is made easier by starting with a reversed domain name.

The PATH constant represents the **object path**. It looks like the bus name with slashes separating the words rather than periods. For most Activities that is exactly what it should be, but it is possible for an application to expose more than one object to D-Bus and in that case each object exposed would have its own unique name, by convention words separated by slashes.

The third constant is IFACE, which is the **interface name**. An interface is a collection of related methods and **signals**, identified by a name that uses the same convention used by the bus name. In the example above, and probably in most Activities using a D-Bus Tube, the interface name and the bus name are identical.

So what is a signal? A signal is like a method but instead of one running program calling a method in one other running program, a signal is **broadcast**. In other words, instead of executing a method in just one program it executes the same method in many running programs, in fact in every running program that has that method that it is connected to through the D-Bus. A signal can pass data into a method call but it can't receive anything back as a return value. It's like a radio station that broadcasts music to anyone that is tuned in. The flow of information is one way only.

Of course a radio station often receives phone calls from its listeners. A disc jockey might play a new song and invite listeners to call the station and say what they thought about it. The phone call is two way communication between the disc jockey and the listener, but it was initiated by a request that was broadcast to all listeners. In the same way your Activity might use a signal to invite all listeners (Buddies) to use a method to call it back, and that method can both supply and receive information.

In D-Bus methods and signals have **signatures**. A signature is a description of the parameters passed into a method or signal including its **data types**. Python is not a **strongly typed** language. In a strongly typed language every variable has a data type that limits what it can do. Data types include such things as **strings**, **integers**, **long integers**, **floating point numbers**, **booleans**, etc. Each one can be used for a specific purpose only. For instance a boolean can only hold the values **True** and **False**, nothing else. A string can be used to hold strings of characters, but even if those characters represent a number you cannot use a string for calculations. Instead you need to convert the string into one of the numeric data types. An integer can hold integers up to a certain size, and a long integer can hold much larger integers, A floating point number is a number with a decimal point in scientific notation. It is almost useless for business arithmetic, which requires rounded results.

In Python you can put anything into any variable and the language itself will figure out how to deal with it. To make Python work with D-Bus, which requires strongly typed variables that Python doesn't have, you need a way to tell D-Bus what types the variables you pass into a method should have. You do this by using a signature string as an argument to the method or signal. Methods have two strings: an **in_signature** and an **out_signature**. Signals just have a **signature** parameter. Some examples of signature strings:

ii	Two parameters, both integers
sss	Three parameters, all strings
ixd	Three parameters, an integer, a long integer, and a double precision floating point number.
a(ssiii)	An array where each element of the array is a tuple containing two strings and three integers.

There is more information on signature strings in the dbus-python tutorial at http://dbus.freedesktop.org/doc/dbus-python/doc/tutorial.html.

INTRODUCING HELLO MESH AND FRIENDS

If you study the source code of a few shared Activities you'll conclude that many of them contain nearly identical methods, as if they were all copied from the same source. In fact, more likely than not they were. The Activity **Hello Mesh** was created to be an example of how to use D-Bus Tubes in a shared Activity. It is traditional in programming textbooks to have the first example program be something that just prints the words "Hello World" to the console or displays the same words in a window. In that tradition **Hello Mesh** is a program that doesn't do all that much. You can find the code in Gitorious at http://git.sugarlabs.org/projects/hello-mesh.

Hello Mesh is widely copied because it demonstrates how to do things that all shared Activities need to do. When you have a shared Activity you need to be able to do two things:

- Send information or commands to other instances of your Activity.
- Give Buddies joining your Activity a copy of the current state of the Activity.

It does this using two signals and one method:

- A signal called *Hello()* that someone joining the Activity sends to all participants. The *Hello()* method takes no parameters.
- A method called *World()* which instances of the Activity receiving *Hello()* send back to the sender. This method takes a text string as an argument, which is meant to represent the current state of the Activity.
- Another signal called *SendText()* which sends a text string to all participants. This represents updating the state of the shared Activity. In the case of **Scribble** this would be informing the others that this instance has just drawn a new shape.

Rather than study **Hello Mesh** itself I'd like to look at the code derived from it used in **Batalla Naval**. I have taken the liberty of running the comments, originally in Spanish, through *Google Translate* to make everything in English. I have also removed some commented-out lines of code.

This Activity does something clever to make it possible to run it either as a Sugar Activity or as a standalone Python program. The standalone program does not support sharing at all, and it runs in a Window. The class **Activity** is a subclass of **Window**, so when the code is run standalone the *init()* function in **BatallaNaval.py** gets a Window, and when the same code is run as an Activity the instance of class **BatallaNavalActivity** is passed to *init()*:

```
from sugar.activity.activity import Activity, ActivityToolbox
import BatallaNaval
from Collaboration import CollaborationWrapper

class BatallaNavalActivity(Activity):
    ''' The Sugar class called when you run this
        program as an Activity. The name of this
        class file is marked in the
        activity/activity.info file.'''

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

        self.gamename = 'BatallaNaval'

        # Create the basic Sugar toolbar
        toolbox = ActivityToolbox(self)
        self.set_toolbox(toolbox)
        toolbox.show()

        # Create an instance of the CollaborationWrapper
        # so you can share the activity.
        self.colaboracion = CollaborationWrapper(self)

        # The activity is a subclass of Window, so it
        # passses itself to the init function
        BatallaNaval.init(False, self)
```

The other clever thing going on here is that all the collaboration code is placed in its own **CollaborationWrapper** class which takes the instance of the **BatallNavalActivity** class in its constructor. This separates the collaboration code from the rest of the program. Here is the code in **CollaborationWrapper.py**:

```
import logging

from sugar.presence import presenceservice
import telepathy
from dbus.service import method, signal
# In build 656 Sugar lacks sugartubeconn
try:
  from sugar.presence.sugartubeconn import \
      SugarTubeConnection
except:
  from sugar.presence.tubeconn import TubeConnection as \
      SugarTubeConnection
from dbus.gobject_service import ExportedGObject

''' In all collaborative Activities in Sugar we are
    made aware when a player enters or leaves. So that
    everyone knows the state of the Activity we use
```

```
    the methods Hello and World. When a participant
    enters Hello sends a signal that reaches
    all participants and the participants
    respond directly using the method "World",
    which retrieves the current state of the Activity.
    After the updates are given then the signal
    Play is used by each participant to make his move.
    In short this module encapsulates the logic of
    "collaboration" with the following effect:
        - When someone enters the collaboration
          the Hello signal is sent.
        - Whoever receives the Hello signal responds
          with World
        - Every time someone makes a move he uses
          the method Play giving a signal which
          communicates to each participant
          what his move was.
'''

SERVICE = "org.ceibaljam.BatallaNaval"
IFACE = SERVICE
PATH = "/org/ceibaljam/BatallaNaval"

logger = logging.getLogger('BatallaNaval')
logger.setLevel(logging.DEBUG)

class CollaborationWrapper(ExportedGObject):
    ''' A wrapper for the collaboration methods.
        Get the activity and the necessary callbacks.
 '''

    def __init__(self, activity):
        self.activity = activity
        self.presence_service = \
            presenceservice.get_instance()
        self.owner = \
            self.presence_service.get_owner()

    def set_up(self, buddy_joined_cb, buddy_left_cb,
        World_cb, Play_cb, my_boats):
        self.activity.connect('shared',
            self._shared_cb)
        if self.activity._shared_activity:
            # We are joining the activity
            self.activity.connect('joined',
                self._joined_cb)
            if self.activity.get_shared():
                # We've already joined
                self._joined_cb()

        self.buddy_joined = buddy_joined_cb
        self.buddy_left = buddy_left_cb
        self.World_cb = World_cb
        # Called when someone passes the board state.
        self.Play_cb = Play_cb
        # Called when someone makes a move.

        # Submitted by making World on a new partner
        self.my_boats = [(b.nombre, b.orientacion,
            b.largo, b.pos[0],
            b.pos[1]) for b in my_boats]
        self.world = False
        self.entered = False

    def _shared_cb(self, activity):
        self._sharing_setup()
        self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            OfferDBusTube(
            SERVICE, {})
        self.is_initiator = True

    def _joined_cb(self, activity):
        self._sharing_setup()
        self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            ListTubes(
            reply_handler=self._list_tubes_reply_cb,
            error_handler=self._list_tubes_error_cb)
        self.is_initiator = False

    def _sharing_setup(self):
        if self.activity._shared_activity is None:
            logger.error(
                'Failed to share or join activity')
            return

        self.conn = \
            self.activity._shared_activity.telepathy_conn
        self.tubes_chan = \
```

```
            self.activity._shared_activity.telepathy_tubes_chan
        self.text_chan = \
            self.activity._shared_activity.telepathy_text_chan

        self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES].\
            connect_to_signal(
            'NewTube', self._new_tube_cb)

        self.activity._shared_activity.connect(
            'buddy-joined',
            self._buddy_joined_cb)
        self.activity._shared_activity.connect(
            'buddy-left',
            self._buddy_left_cb)

        # Optional - included for example:
        # Find out who's already in the shared activity:
        for buddy in \
            self.activity._shared_activity.\
                get_joined_buddies():
            logger.debug(
                'Buddy %s is already in the activity',
                buddy.props.nick)

    def participant_change_cb(self, added, removed):
        logger.debug(
            'Tube: Added participants: %r', added)
        logger.debug(
            'Tube: Removed participants: %r', removed)
        for handle, bus_name in added:
            buddy = self._get_buddy(handle)
            if buddy is not None:
                logger.debug(
                    'Tube: Handle %u (Buddy %s) was added',
                    handle, buddy.props.nick)
        for handle in removed:
            buddy = self._get_buddy(handle)
            if buddy is not None:
                logger.debug('Buddy %s was removed' %
                    buddy.props.nick)
        if not self.entered:
            if self.is_initiator:
                logger.debug(
                    "I'm initiating the tube, "
                    "will watch for hellos.")
                self.add_hello_handler()
            else:
                logger.debug(
                    'Hello, everyone! What did I miss?')
                self.Hello()
        self.entered = True


    # This is sent to all participants whenever we
    # join an activity
    @signal(dbus_interface=IFACE, signature='')
    def Hello(self):
        """Say Hello to whoever else is in the tube."""
        logger.debug('I said Hello.')

    # This is called by whoever receives our Hello signal
    # This method receives the current game state and
    # puts us in sync with the rest of the participants.
    # The current game state is represented by the
    # game object
    @method(dbus_interface=IFACE, in_signature='a(ssiii)',
        out_signature='a(ssiii)')
    def World(self, boats):
        """To be called on the incoming XO after
        they Hello."""
        if not self.world:
            logger.debug('Somebody called World on me')
            self.world = True   # Instead of loading
                                # the world, I am
                                # receiving play by
                                # play.
            self.World_cb(boats)
            # now I can World others
            self.add_hello_handler()
        else:
            self.world = True
            logger.debug(
                "I've already been welcomed, doing nothing")
        return self.my_boats

    @signal(dbus_interface=IFACE, signature='ii')
    def Play(self, x, y):
        """Say Hello to whoever else is in the tube."""
```

```
    logger.debug('Running remote play:%s x %s.', x, y)

def add_hello_handler(self):
    logger.debug('Adding hello handler.')
    self.tube.add_signal_receiver(self.hello_signal_cb,
        'Hello', IFACE,
        path=PATH, sender_keyword='sender')
    self.tube.add_signal_receiver(self.play_signal_cb,
        'Play', IFACE,
        path=PATH, sender_keyword='sender')

def hello_signal_cb(self, sender=None):
    """Somebody Helloed me. World them."""
    if sender == self.tube.get_unique_name():
        # sender is my bus name, so ignore my own signal
        return
    logger.debug('Newcomer %s has joined', sender)
    logger.debug(
        'Welcoming newcomer and sending them '
        'the game state')

    self.other = sender

    # I send my ships and I get theirs in return
    enemy_boats = self.tube.get_object(self.other,
        PATH).World(
        self.my_boats, dbus_interface=IFACE)

    # I call the callback World, to load the enemy ships
    self.World_cb(enemy_boats)

def play_signal_cb(self, x, y, sender=None):
    """Somebody placed a stone. """
    if sender == self.tube.get_unique_name():
        return  # sender is my bus name,
                # so ignore my own signal
    logger.debug('Buddy %s placed a stone at %s x %s',
        sender, x, y)
    # Call our Play callback
    self.Play_cb(x, y)
    # In theory, no matter who sent him

def _list_tubes_error_cb(self, e):
    logger.error('ListTubes() failed: %s', e)

def _list_tubes_reply_cb(self, tubes):
    for tube_info in tubes:
        self._new_tube_cb(*tube_info)

def _new_tube_cb(self, id, initiator, type,
    service, params, state):
    logger.debug('New tube: ID=%d initator=%d '
        'type=%d service=%s '
        'params=%r state=%d', id, initiator, '
        'type, service, params, state)
    if (type == telepathy.TUBE_TYPE_DBUS and
        service == SERVICE):
        if state == telepathy.TUBE_STATE_LOCAL_PENDING:
            self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES]
                .AcceptDBusTube(id)
        self.tube = SugarTubeConnection(self.conn,
            self.tubes_chan[telepathy.CHANNEL_TYPE_TUBES],
            id, group_iface=
                self.text_chan[telepathy.\
                    CHANNEL_INTERFACE_GROUP])
        super(CollaborationWrapper,
            self).__init__(self.tube, PATH)
        self.tube.watch_participants(
            self.participant_change_cb)

def _buddy_joined_cb (self, activity, buddy):
    """Called when a buddy joins the shared
    activity. """
    logger.debug(
        'Buddy %s joined', buddy.props.nick)
    if self.buddy_joined:
        self.buddy_joined(buddy)

def _buddy_left_cb (self, activity, buddy):
    """Called when a buddy leaves the shared
    activity. """
    if self.buddy_left:
        self.buddy_left(buddy)

def _get_buddy(self, cs_handle):
    """Get a Buddy from a channel specific handle."""
    logger.debug('Trying to find owner of handle %u...',
        cs_handle)
```

```
        group = self.text_chan[telepathy.\
            CHANNEL_INTERFACE_GROUP]
        my_csh = group.GetSelfHandle()
        logger.debug(
            'My handle in that group is %u', my_csh)
        if my_csh == cs_handle:
            handle = self.conn.GetSelfHandle()
            logger.debug('CS handle %u belongs to me, %u',
                cs_handle, handle)
        elif group.GetGroupFlags() & \
            telepathy.\
            CHANNEL_GROUP_FLAG_CHANNEL_SPECIFIC_HANDLES:
            handle = group.GetHandleOwners([cs_handle])[0]
            logger.debug('CS handle %u belongs to %u',
                cs_handle, handle)
        else:
            handle = cs_handle
            logger.debug('non-CS handle %u belongs to itself',
                handle)
            # XXX: deal with failure to get the handle owner
            assert handle != 0
        return self.presence_service.\
            get_buddy_by_telepathy_handle(
            self.conn.service_name,
            self.conn.object_path, handle)
```

Most of the code above is similar to what we've seen in the other examples, and most of it can be used as is in any Activity that needs to make D-Bus calls. For this reason we'll focus on the code that is specific to using D-Bus. The logical place to start is the *Hello()* method. There is of course nothing magic about the name "Hello". **Hello Mesh** is meant to be a "Hello World" program for using D-Bus Tubes, so by convention the words "Hello" and "World" had to be used for *something*. The *Hello()* method is broadcast to all instances of the Activity to inform them that a new instance is ready to receive information about the current state of the shared Activity. Your own Activity will probably need something similar, but you should feel free to name it something else, and if you're writing the code for a school assignment you should definitely name it something else:

```
    # This is sent to all participants whenever we
    # join an activity
    @signal(dbus_interface=IFACE, signature='')
    def Hello(self):
        """Say Hello to whoever else is in the tube."""
        logger.debug('I said Hello.')

    def add_hello_handler(self):
        logger.debug('Adding hello handler.')
        self.tube.add_signal_receiver(
            self.hello_signal_cb,
            'Hello', IFACE,
            path=PATH, sender_keyword='sender')
...

    def hello_signal_cb(self, sender=None):
        """Somebody Helloed me. World them."""
        if sender == self.tube.get_unique_name():
            # sender is my bus name,
            # so ignore my own signal
            return
        logger.debug('Newcomer %s has joined', sender)
        logger.debug(
            'Welcoming newcomer and sending them '
            'the game state')

        self.other = sender

        # I send my ships and I returned theirs
        enemy_boats = self.tube.get_object(
            self.other, PATH).World(
            self.my_boats, dbus_interface=IFACE)

        # I call the callback World, to load the enemy ships
        self.World_cb(enemy_boats)
```

The most interesting thing about this code is this line, which Python calls a **Decorator**:

```
    @signal(dbus_interface=IFACE, signature='')
```

When you put **@signal** in front of a method name it has the effect of adding the two parameters shown to the method call whenever it is invoked, in effect changing it from a normal method call to a D-Bus call for a signal. The **signature** parameter is an empty string, indicating that the method call has no parameters. The *Hello()* method does nothing at all locally but when it is received by the other instances of the Activity it causes them to execute the *World()* method, which sends back the location of their boats and gets the new participants boats in return.

Batalla Naval is apparently meant to be a demonstration program. *Battleship* is a game for two players, but there is nothing in the code to prevent more players from joining and no way to handle it if they do. Ideally you would want code to make only the first joiner an actual player and make the rest only spectators.

Next we'll look at the *World()* method:

```
    # This is called by whoever receives our Hello signal
    # This method receives the current game state and
    # puts us in sync with the rest of the participants.
    # The current game state is represented by the game
    # object
    @method(dbus_interface=IFACE, in_signature='a(ssiii)',
        out_signature='a(ssiii)')
    def World(self, boats):
        """To be called on the incoming XO after
        they Hello."""
        if not self.world:
            logger.debug('Somebody called World on me')
            self.world = True   # Instead of loading the world,
                                # I am receiving play by play.
            self.World_cb(boats)
            # now I can World others
            self.add_hello_handler()
        else:
            self.world = True
            logger.debug("I've already been welcomed, "
                "doing nothing")
        return self.my_boats
```

There is another decorator here, this one converting the *World()* method to a D-Bus call for a method. The signature is more interesting than *Hello()* had. It means an array of tuples where each tuple contains two strings and three integers. Each element in the array represents one ship and its attributes. *World_cb* is set to point to a method in **BatallaNaval.py**, (and so is *Play_cb*). If you study the *init()* code in **BatallaNaval.py** you'll see how this happens. *World()* is called in the *hello_signal_cb()* method we just looked at. It is sent to the joiner who sent *Hello()* to us.

Finally we'll look at the *Play()* signal:

```
    @signal(dbus_interface=IFACE, signature='ii')
    def Play(self, x, y):
        """Say Hello to whoever else is in the tube."""
        logger.debug('Running remote play:%s x %s.', x, y)

    def add_hello_handler(self):
...
        self.tube.add_signal_receiver(self.play_signal_cb,
            'Play', IFACE,
            path=PATH, sender_keyword='sender')
...
    def play_signal_cb(self, x, y, sender=None):
        """Somebody placed a stone. """
        if sender == self.tube.get_unique_name():
            return  # sender is my bus name, so
                    # ignore my own signal
        logger.debug('Buddy %s placed a stone at %s x %s',
            sender, x, y)
        # Call our Play callback
        self.Play_cb(x, y)
```

There are several ways you could improve this Activity. When playing against the computer in non-sharing mode the game just makes random moves. The game does not limit the players to two and make the rest of the joiners spectators. It does not make the players take turns. When a player succeeds in sinking all the other players ships nothing happens to mark the event. Finally, *gettext()* is not used for the text strings displayed by the Activity so it cannot be translated into languages other than Spanish.

In the tradition of textbooks everywhere I will leave making these improvements as an exercise for the student.

16. ADDING TEXT TO SPEECH

INTRODUCTION

Certainly one of the most popular Activities available is **Speak**, which takes the words you type in and speaks them out loud, at the same time displaying a cartoon face that seems to be speaking the words. You might be surprised to learn how little of the code in that Activity is used to get the words spoken. If your Activity could benefit from having words spoken out loud (the possibilities for educational Activities and games are definitely there) this chapter will teach you how to make it happen.

WE HAVE WAYS TO MAKE YOU TALK

A couple of ways, actually, and neither one is that painful. They are:

- Running the **espeak** program directly
- Using the **gstreamer espeak plugin**

Both approaches have their advantages. The first one is the one used by Speak. (Technically, Speak uses the gstreamer plugin if it is available, and otherwise executes espeak directly. For what Speak is doing using the gstreamer plugin isn't really needed). Executing espeak is definitely the simplest method, and may be suitable for your own Activity. Its big advantage is that you do not need to have the gstreamer plugin installed. If your Activity needs to run on something other than the latest version of Sugar this will be something to consider.

The gstreamer plugin is what is used by **Read Etexts** to do text to speech with highlighting. For this application we needed to be able to do things that are not possible by just running **espeak**. For example:

- We needed to be able to pause and resume speech, because the Activity needs to speak a whole page worth of text, not just simple phrases.
- We needed to highlight the words being spoken as they are spoken.

You might think that you could achieve these objectives by running espeak on one word at a time. If you do, don't feel bad because I thought that too. On a fast computer it sounds really awful, like HAL 9000 developing a stutter towards the end of being deactivated. On the XO no sounds came out at all.

Originally Read Etexts used **speech-dispatcher** to do what the gstreamer plugin does. The developers of that program were very helpful in getting the highlighting in Read Etexts working, but speech-dispatcher needed to be configured before you could use it which was an issue for us. (There is more than one kind of text to speech software available and speech-dispatcher supports most of them. This makes configuration files inevitable). Aleksey Lim of Sugar Labs came up with the idea of using a gstreamer plugin and was the one who wrote it. He also rewrote much of **Read Etexts** so it would use the plugin if it was available, use speech-dispatcher if not, and would not support speech if neither was available.

RUNNING ESPEAK DIRECTLY

You can run the **espeak** program from the terminal to try out its options. To see what options are available for espeak you can use the **man** command:

```
man espeak
```

This will give you a manual page describing how to run the program and what options are available. The parts of the man page that are most interesting to us are these:

```
NAME
       espeak - A multi-lingual software speech synthesizer.

SYNOPSIS
       espeak [options] [<words>]

DESCRIPTION
       espeak is a software speech synthesizer for English,
       and some other languages.

OPTIONS
       -p <integer>
              Pitch adjustment, 0 to 99, default is 50

       -s <integer>
              Speed in words per minute, default is 160

       -v <voice name>
              Use voice file of this name from
              espeak-data/voices

       --voices[=<language code>]
               Lists the available voices. If =<language code>
               is present then only those voices which are
               suitable for that language are listed.
```

Let's try out some of these options. First let's get a list of **Voices**:

```
espeak --voices
Pty Language Age/Gender VoiceName       File        Other Langs
 5  af             M  afrikaans         af
 5  bs             M  bosnian           bs
 5  ca             M  catalan           ca
 5  cs             M  czech             cs
 5  cy             M  welsh-test        cy
 5  de             M  german            de
 5  el             M  greek             el
```

```
 5  en             M  default          default
 5  en-sc          M  en-scottish      en/en-sc    (en 4)
 2  en-uk          M  english          en/en       (en 2)
... and many more ...
```

Now that we know what the names of the voices are we can try them out. How about English with a French accent?

```
espeak "Your mother was a hamster and your father \
smelled of elderberries." -v fr
```

Let's try experimenting with rate and pitch:

```
espeak "I'm sorry, Dave. I'm afraid I can't \
do that." -s 120 -p 30
```

The next thing to do is to write some Python code to run espeak. Here is a short program adapted from the code in **Speak**:

```
import re
import subprocess

PITCH_MAX = 99
RATE_MAX = 99
PITCH_DEFAULT = PITCH_MAX/2
RATE_DEFAULT = RATE_MAX/3

def speak(text, rate=RATE_DEFAULT, pitch=PITCH_DEFAULT,
    voice="default"):

    # espeak uses 80 to 370
    rate = 80 + (370-80) * int(rate) / 100

    subprocess.call(["espeak", "-p", str(pitch),
            "-s", str(rate), "-v", voice,  text],
            stdout=subprocess.PIPE)

def voices():
    out = []
    result = subprocess.Popen(["espeak", "--voices"],
        stdout=subprocess.PIPE).communicate()[0]

    for line in result.split('\n'):
        m = re.match(
            r'\s*\d+\s+([\w-]+)\s+([MF])\s+([\w_-]+)\s+(.+)',
            line)
        if not m:
            continue
        language, gender, name, stuff = m.groups()
        if stuff.startswith('mb/') or \
            name in ('en-rhotic','english_rp',
                'english_wmids'):
            # these voices don't produce sound
            continue
        out.append((language, name))

    return out

def main():
    print voices()
    speak("I'm afraid I can't do that, Dave.")
    speak("Your mother was a hamster, and your father "
        + "smelled of elderberries!",  30,  60,  "fr")

if __name__ == "__main__":
    main()
```

In the Git repository in the directory **Adding_TTS** this file is named **espeak.py**. Load this file into **Eric** and do **Run Script** from the **Start** menu to run it. In addition to hearing speech you should see this text:

[('af', 'afrikaans'), ('bs', 'bosnian'), ('ca', 'catalan'), ('cs', 'czech'), ('cy', 'welsh-test'), ('de', 'german'), ('el', 'greek'), ('en', 'default'), ('en-sc', 'en-scottish'), ('en-uk', 'english'), ('en-uk-north', 'lancashire'), ('en-us', 'english-us'), ('en-wi', 'en-westindies'), ('eo', 'esperanto'), ('es', 'spanish'), ('es-la', 'spanish-latin-american'), ('fi', 'finnish'), ('fr', 'french'), ('fr-be', 'french'), ('grc', 'greek-ancient'), ('hi', 'hindi-test'), ('hr', 'croatian'), ('hu', 'hungarian'), ('hy', 'armenian'), ('hy', 'armenian-west'), ('id', 'indonesian-test'), ('is', 'icelandic-test'), ('it', 'italian'), ('ku', 'kurdish'), ('la', 'latin'), ('lv', 'latvian'), ('mk', 'macedonian-test'), ('nl', 'dutch-test'), ('no', 'norwegian-test'), ('pl', 'polish'), ('pt', 'brazil'), ('pt-pt', 'portugal'), ('ro', 'romanian'), ('ru', 'russian_test'), ('sk', 'slovak'), ('sq', 'albanian'), ('sr', 'serbian'), ('sv', 'swedish'), ('sw', 'swahihi-test'), ('ta', 'tamil'), ('tr', 'turkish'), ('vi', 'vietnam-test'), ('zh', 'Mandarin'), ('zh-yue', 'cantonese-test')]

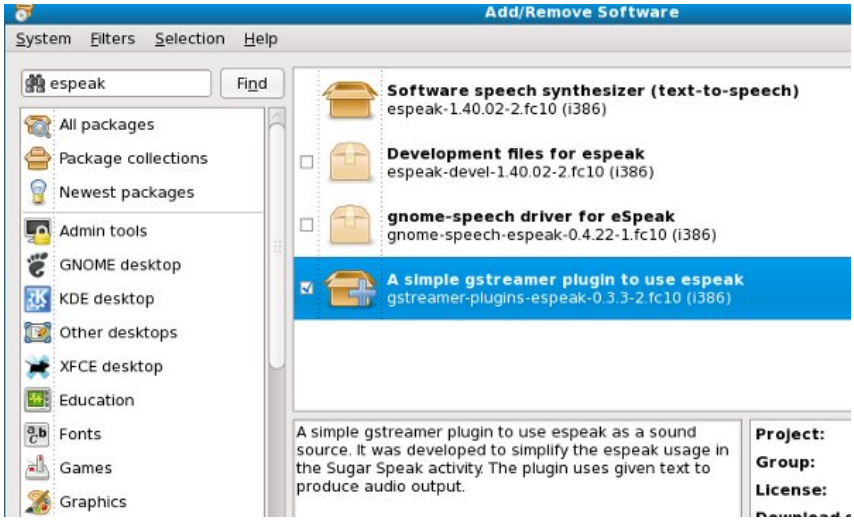
The *voices()* function returns a list of voices as one tuple per voice, and eliminates voices from the list that espeak cannot produce on its own. This list of tuples can be used to populate a drop down list.

The *speak()* function adjusts the value of **rate** so you can input a value between 0 and 99 rather than between 80 to 370. *speak()* is more complex in the Speak Activity than what we have here because in that Activity it monitors the spoken audio and generates mouth movements based on the amplitude of the voice. Making the face move is most of what the Speak Activity does, and since we aren't doing that we need very little code to make our Activity speak.

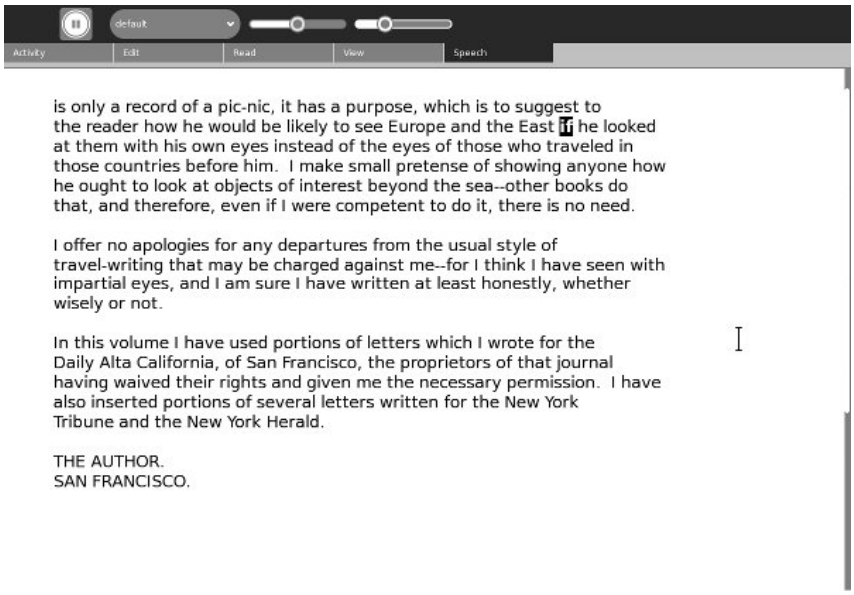
You can use **import espeak** to include this file in your own Activities.

USING THE GSTREAMER ESPEAK PLUGIN

The gstreamer espeak plugin can be installed in **Fedora 10** or later using **Add/Remove Software**.



When you have this done you should be able to download the **Read Etexts** Activity (the real one, not the simplified version we're using for the book) from ASLO and try out the **Speech** tab. You should do that now. It will look something like this:



The book used in the earlier screenshots of this manual was *Pride and Prejudice* by Jane Austen. To balance things out the rest of the screen shots will be using *The Innocents Abroad* by Mark Twain.

Gstreamer is a framework for multimedia. If you've watched videos on the web you are familiar with the concept of streaming media. Instead of downloading a whole song or a whole movie clip and then playing it, streaming means the downloading and the playing happen at the same time, with the downloading just a bit behind the playing. There are many different kinds of media files: MP3's, DivX, WMV, Real Media, and so on. For every kind of media file Gstreamer has a plugin.

Gstreamer makes use of a concept called **pipelining**. The idea is that the output of one program can become the input to another. One way to handle that situation is to put the output of the first program into a temporary file and have the second program read it. This would mean that the first program would have to finish running before the second one could begin. What if you could have both programs run at the same time and have the second program read the data as the first one wrote it out? It's possible, and the mechanism for getting data from one program to the other is called a **pipe**. A collection of programs joined together in this way is called a **pipeline**. The program that feeds data into a pipe is called a **source**, and the data that takes the data out of the pipe is called a **sink**.

The gstreamer espeak plugin uses a simple pipe: text goes into espeak at one end and sound comes out the other and is sent to your soundcard. You might think that doesn't sound much different from what we were doing before, but it is. When you just run espeak the program has to load itself into memory, speak the text you give it into the sound card, then unload itself. This is one of the reasons you can't just use espeak a word at a time to achieve speech with highlighted words. There is a short lag while the program is loading. It isn't that noticeable if you give espeak a complete phrase or sentence to speak, but if it happens for every word it is *very* noticeable. Using the gstreamer plugin we can have espeak loaded into memory all the time, just waiting for us to send some words into its input pipe. It will speak them and then wait for the next batch.

Since we can control what goes into the pipe it is possible to pause and resume speech.

The example we'll use here is a version of **Read Etexts** again, but instead of the Activity we're going to modify the standalone version. There is nothing special about the gstreamer plugin that makes it only work with Activities. Any Python program can use it. I'm only including Text to Speech as a topic in this manual because every Sugar installation includes espeak and many Activities could find it useful.

There is a in our Git repository named **speech.py** which looks like this:

```
import gst

voice = 'default'
pitch = 0

rate = -20
highlight_cb = None

def _create_pipe():
    pipeline = 'espeak name=source ! autoaudiosink'
    pipe = gst.parse_launch(pipeline)

    def stop_cb(bus, message):
        pipe.set_state(gst.STATE_NULL)

    def mark_cb(bus, message):
        if message.structure.get_name() == 'espeak-mark':
            mark = message.structure['mark']
            highlight_cb(int(mark))

    bus = pipe.get_bus()
    bus.add_signal_watch()
    bus.connect('message::eos', stop_cb)
    bus.connect('message::error', stop_cb)
    bus.connect('message::element', mark_cb)

    return (pipe.get_by_name('source'), pipe)

def _speech(source, pipe, words):
    source.props.pitch = pitch
    source.props.rate = rate
    source.props.voice = voice
    source.props.text = words;
    pipe.set_state(gst.STATE_PLAYING)

info_source, info_pipe = _create_pipe()
play_source, play_pipe = _create_pipe()

# track for marks
play_source.props.track = 2

def voices():
    return info_source.props.voices

def say(words):
    _speech(info_source, info_pipe, words)
    print words

def play(words):
    _speech(play_source, play_pipe, words)

def is_stopped():
    for i in play_pipe.get_state():
        if isinstance(i, gst.State) and \
            i == gst.STATE_NULL:
            return True
    return False

def stop():
    play_pipe.set_state(gst.STATE_NULL)

def is_paused():
    for i in play_pipe.get_state():
        if isinstance(i, gst.State) and \
            i == gst.STATE_PAUSED:
            return True
    return False

def pause():
    play_pipe.set_state(gst.STATE_PAUSED)

def rate_up():
    global rate
    rate = min(99, rate + 10)
```

```
def rate_down():
    global rate
    rate = max(-99, rate - 10)

def pitch_up():
    global pitch
    pitch = min(99, pitch + 10)

def pitch_down():
    global pitch
    pitch = max(-99, pitch - 10)

def prepare_highlighting(label_text):
    i = 0
    j = 0
    word_begin = 0
    word_end = 0
    current_word = 0
    word_tuples = []
    omitted = [' ', '\n', u'\r', '_', '[', '{', ']',\
        '}', '|', '<', '>', '*', '+', '/', '\\' ]
    omitted_chars = set(omitted)
    while i < len(label_text):
        if label_text[i] not in omitted_chars:
            word_begin = i
            j = i
            while j < len(label_text) and \
                 label_text[j] not in omitted_chars:
                 j = j + 1
                 word_end = j
                 i = j
            word_t = (word_begin, word_end, \
                 label_text[word_begin: word_end].strip())
            if word_t[2] != u'\r':
                 word_tuples.append(word_t)
        i = i + 1
    return word_tuples

def add_word_marks(word_tuples):
    "Adds a mark between each word of text."
    i = 0
    marked_up_text = '<speak> '
    while i < len(word_tuples):
        word_t = word_tuples[i]
        marked_up_text = marked_up_text + \
            '<mark name="' + str(i) + '"/>' + word_t[2]
        i = i + 1
    return marked_up_text + '</speak>'
```

There is another file named **ReadEtextsTTS.py** which looks like this:

```
import sys
import os
import zipfile
import pygtk
import gtk
import getopt
import pango
import gobject
import time
import speech

speech_supported = True

try:
    import gst
    gst.element_factory_make('espeak')
    print 'speech supported!'
except Exception, e:
    speech_supported = False
    print 'speech not supported!'

page=0
PAGE_SIZE = 45

class ReadEtextsActivity():
    def __init__(self):
        "The entry point to the Activity"
        speech.highlight_cb = self.highlight_next_word
        # print speech.voices()

    def highlight_next_word(self, word_count):
        if word_count <= len(self.word_tuples):
            word_tuple = self.word_tuples[word_count]
            textbuffer = self.textview.get_buffer()
            tag = textbuffer.create_tag()
            tag.set_property('weight', pango.WEIGHT_BOLD)
```

```
            tag.set_property( 'foreground', "white")
            tag.set_property( 'background', "black")
            iterStart = \
                textbuffer.get_iter_at_offset(word_tuple[0])
            iterEnd = \
                textbuffer.get_iter_at_offset(word_tuple[1])
            bounds = textbuffer.get_bounds()
            textbuffer.remove_all_tags(bounds[0], bounds[1])
            textbuffer.apply_tag(tag, iterStart, iterEnd)
            v_adjustment = \
                self.scrolled_window.get_vadjustment()
            max = v_adjustment.upper - \
                v_adjustment.page_size
            max = max * word_count
            max = max / len(self.word_tuples)
            v_adjustment.value = max
        return True

    def keypress_cb(self, widget, event):
        "Respond when the user presses one of the arrow keys"
        global done
        global speech_supported
        keyname = gtk.gdk.keyval_name(event.keyval)
        if keyname == 'KP_End' and speech_supported:
            if speech.is_paused() or speech.is_stopped():
                speech.play(self.words_on_page)
            else:
                speech.pause()
            return True
        if keyname == 'plus':
            self.font_increase()
            return True
        if keyname == 'minus':
            self.font_decrease()
            return True
        if speech_supported and speech.is_stopped() == False \
            and speech.is_paused == False:
            # If speech is in progress, ignore other keys.
            return True
        if keyname == '7':
            speech.pitch_down()
            speech.say('Pitch Adjusted')
            return True
        if keyname == '8':
            speech.pitch_up()
            speech.say('Pitch Adjusted')
            return True
        if keyname == '9':
            speech.rate_down()
            speech.say('Rate Adjusted')
            return True
        if keyname == '0':
            speech.rate_up()
            speech.say('Rate Adjusted')
            return True
        if keyname == 'KP_Right':
            self.page_next()
            return True
        if keyname == 'Page_Up' or keyname == 'KP_Up':
            self.page_previous()
            return True
        if keyname == 'KP_Left':
            self.page_previous()
            return True
        if keyname == 'Page_Down' or keyname == 'KP_Down':
            self.page_next()
            return True
        if keyname == 'Up':
            self.scroll_up()
            return True
        if keyname == 'Down':
            self.scroll_down()
            return True
        return False

    def page_previous(self):
        global page
        page=page-1
        if page < 0: page=0
        self.show_page(page)
        v_adjustment = \
            self.scrolled_window.get_vadjustment()
        v_adjustment.value = v_adjustment.upper - \
            v_adjustment.page_size

    def page_next(self):
        global page
        page=page+1
```

```
        if page >= len(self.page_index): page=0
        self.show_page(page)
        v_adjustment = \
            self.scrolled_window.get_vadjustment()
        v_adjustment.value = v_adjustment.lower

    def font_decrease(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size - 1
        if font_size < 1:
            font_size = 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def font_increase(self):
        font_size = self.font_desc.get_size() / 1024
        font_size = font_size + 1
        self.font_desc.set_size(font_size * 1024)
        self.textview.modify_font(self.font_desc)

    def scroll_down(self):
        v_adjustment = \
            self.scrolled_window.get_vadjustment()
        if v_adjustment.value == v_adjustment.upper - \
            v_adjustment.page_size:
            self.page_next()
            return
        if v_adjustment.value < v_adjustment.upper - \
            v_adjustment.page_size:
            new_value = v_adjustment.value + \
                v_adjustment.step_increment
            if new_value > v_adjustment.upper - \
                v_adjustment.page_size:
                new_value = v_adjustment.upper - \
                    v_adjustment.page_size
            v_adjustment.value = new_value

    def scroll_up(self):
        v_adjustment = \
            self.scrolled_window.get_vadjustment()
        if v_adjustment.value == v_adjustment.lower:
            self.page_previous()
            return
        if v_adjustment.value > v_adjustment.lower:
            new_value = v_adjustment.value - \
                v_adjustment.step_increment
            if new_value < v_adjustment.lower:
                new_value = v_adjustment.lower
            v_adjustment.value = new_value

    def show_page(self, page_number):
        global PAGE_SIZE, current_word
        position = self.page_index[page_number]
        self.etext_file.seek(position)
        linecount = 0
        label_text = ''
        textbuffer = self.textview.get_buffer()
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + \
                unicode(line, 'iso-8859-1')
            linecount = linecount + 1
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)
        self.word_tuples = \
            speech.prepare_highlighting(label_text)
        self.words_on_page = \
            speech.add_word_marks(self.word_tuples)

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        filebytes = zipfile.read(filename)
        f = open("/tmp/" + filename, 'w')
        try:
            f.write(filebytes)
        finally:
            f.close()

    def read_file(self, filename):
        "Read the Etext file"
        global PAGE_SIZE

        if zipfile.is_zipfile(filename):
            self.zf = zipfile.ZipFile(filename, 'r')
            self.book_files = self.zf.namelist()
            self.save_extracted_file(self.zf, \
                self.book_files[0])
            currentFileName = "/tmp/" + self.book_files[0]
```

```
        else:
            currentFileName = filename

        self.etext_file = open(currentFileName,"r")
        self.page_index = [ 0 ]
        linecount = 0
        while self.etext_file:
            line = self.etext_file.readline()
            if not line:
                break
            linecount = linecount + 1
            if linecount >= PAGE_SIZE:
                position = self.etext_file.tell()
                self.page_index.append(position)
                linecount = 0
        if filename.endswith(".zip"):
            os.remove(currentFileName)

    def delete_cb(self, widget, event, data=None):
        speech.stop()
        return False

    def destroy_cb(self, widget, data=None):
        speech.stop()
        gtk.main_quit()

    def main(self, file_path):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("delete_event", self.delete_cb)
        self.window.connect("destroy", self.destroy_cb)
        self.window.set_title("Read Etexts Activity")
        self.window.set_size_request(800, 600)
        self.window.set_border_width(0)
        self.read_file(file_path)
        self.scrolled_window = gtk.ScrolledWindow(
            hadjustment=None, vadjustment=None)
        self.textview = gtk.TextView()
        self.textview.set_editable(False)
        self.textview.set_left_margin(50)
        self.textview.set_cursor_visible(False)
        self.textview.connect("key_press_event",
            self.keypress_cb)
        self.font_desc = pango.FontDescription("sans 12")
        self.textview.modify_font(self.font_desc)
        self.show_page(0)
        self.scrolled_window.add(self.textview)
        self.window.add(self.scrolled_window)
        self.textview.show()
        self.scrolled_window.show()
        self.window.show()
        gtk.main()

if __name__ == "__main__":
    try:
        opts, args = getopt.getopt(sys.argv[1:], "")
        ReadEtextsActivity().main(args[0])
    except getopt.error, msg:
        print msg
        print "This program has no options"
        sys.exit(2)
```

The program **ReadEtextsTTS** has only a few changes to make it enabled for speech. The first one checks for the existence of the gstreamer plugin:

```
speech_supported = True

try:
    import gst
    gst.element_factory_make('espeak')
    print 'speech supported!'
except Exception, e:
    speech_supported = False
    print 'speech not supported!'
```

This code detects whether the plugin is installed by attempting to import for the Python library associated with it named "gst". If the import fails it throws an **Exception** and we catch that Exception and use it to set a variable named **speech_supported** to **False**. We can check the value of this variable in other places in the program to make a program that works with Text To Speech if it is available and without it if it is not. Making a program work in different environments by doing these kinds of checks is called *degrading gracefully*. Catching exceptions on imports is a common technique in Python to achieve this. If you want your Activity to run on older versions of Sugar you may find yourself using it.

The next bit of code we're going to look at highlights a word in the textview and scrolls the textview to keep the highlighted word visible.

```
class ReadEtextsActivity():
    def __init__(self):
        "The entry point to the Activity"
        speech.highlight_cb = self.highlight_next_word
        # print speech.voices()

    def highlight_next_word(self, word_count):
        if word_count < len(self.word_tuples):
            word_tuple = self.word_tuples[word_count]
            textbuffer = self.textview.get_buffer()
            tag = textbuffer.create_tag()
            tag.set_property('weight', pango.WEIGHT_BOLD)
            tag.set_property( 'foreground', "white")
            tag.set_property( 'background', "black")
            iterStart = \
                textbuffer.get_iter_at_offset(word_tuple[0])
            iterEnd = \
                textbuffer.get_iter_at_offset(word_tuple[1])
            bounds = textbuffer.get_bounds()
            textbuffer.remove_all_tags(bounds[0], bounds[1])
            textbuffer.apply_tag(tag, iterStart, iterEnd)
            v_adjustment = \
                self.scrolled_window.get_vadjustment()
            max = v_adjustment.upper - v_adjustment.page_size
            max = max * word_count
            max = max / len(self.word_tuples)
            v_adjustment.value = max
        return True
```

In the *__init__()* method we assign a variable called *highlight_cb* in **speech.py** with a method called *highlight_next_word()*. This gives **speech.py** a way to call that method every time a new word in the textview needs to be highlighted.

The next line will print the list of tuples containing Voice names to the terminal if you uncomment it. We aren't letting the user change voices in this application but it would not be difficult to add that feature.

The code for the method that highlights the words follows. What it does is look in a list of tuples that contain the starting and ending offsets of every word in the textarea's text buffer. The caller of this method passes in a word number (for instance the first word in the buffer is word 0, the second is word 1, and so on). The method looks up that entry in the list, gets its starting and ending offsets, removes any previous highlighting, then highlights the new text. In addition to that it figures out what fraction of the total number of words the current word is and scrolls the textviewer enough to make sure that word is visible.

Of course this method works best on pages without many blank lines, which fortunately is most pages. It does not work so well on title pages. An experienced programmer could probably come up with a more elegant and reliable way of doing this scrolling. Let me know what you come up with.

Further down we see the code that gets the keystrokes the user enters and does speech-related things with them:

```
    def keypress_cb(self, widget, event):
        "Respond when the user presses one of the arrow keys"
        global done
        global speech_supported
        keyname = gtk.gdk.keyval_name(event.keyval)
        if keyname == 'KP_End' and speech_supported:
            if speech.is_paused() or speech.is_stopped():
                speech.play(self.words_on_page)
            else:
                speech.pause()
            return True
        if speech_supported and speech.is_stopped() == False \
            and speech.is_paused == False:
            # If speech is in progress, ignore other keys.
            return True
        if keyname == '7':
            speech.pitch_down()
            speech.say('Pitch Adjusted')
            return True
        if keyname == '8':
            speech.pitch_up()
            speech.say('Pitch Adjusted')
            return True
        if keyname == '9':
            speech.rate_down()
            speech.say('Rate Adjusted')
            return True
        if keyname == '0':
            speech.rate_up()
            speech.say('Rate Adjusted')
            return True
```

As you can see, the functions we're calling are all in the file **speech.py** that we imported. You don't have to fully understand how these functions work to make use of them in your own Activities. Notice that the code as written prevents the user from changing pitch or rate while speech is in progress. Notice also that there are two different methods in speech.py for doing speech. **play()** is the method for doing text to speech with word highlighting. **say()** is for saying short phrases produced by the user interface, in this case "Pitch adjusted" and "Rate adjusted". Of course if you put code like this in your Activity you would use the _() function so these phrases could be translated into other languages.

There is one more bit of code we need to do text to speech with highlighting: we need to prepare the words to be spoken to be highlighted in the textviewer.

```
    def show_page(self, page_number):
        global PAGE_SIZE, current_word
        position = self.page_index[page_number]
        self.etext_file.seek(position)
        linecount = 0
        label_text = ''
        textbuffer = self.textview.get_buffer()
        while linecount < PAGE_SIZE:
            line = self.etext_file.readline()
            label_text = label_text + unicode(line, \
                'iso-8859-1')
            linecount = linecount + 1
        textbuffer.set_text(label_text)
        self.textview.set_buffer(textbuffer)
        self.word_tuples = \
            speech.prepare_highlighting(label_text)
        self.words_on_page = \
            speech.add_word_marks(self.word_tuples)
```

The beginning of this method reads a page's worth of text into a string called label_text and puts it into the textview's buffer. The last two lines splits the text into words, leaving in punctuation, and puts each word and its beginning and ending offsets into a tuple. The tuples are added to a List.

speech.add_word_marks() converts the words in the List to a document in *SSML* (*Speech Synthesis Markup Language*) format. SSML is a standard for adding tags (sort of like the tags used to make web pages) to text to tell speech software what to do with the text. We're just using a very small part of this standard to produce a marked up document with a mark between each word, like this:

```
<speak>
    <mark name="0"/>The<mark name="1"/>quick<mark name-"2"/>
    brown<mark name="3"/>fox<mark name="4"/>jumps
</speak>
```

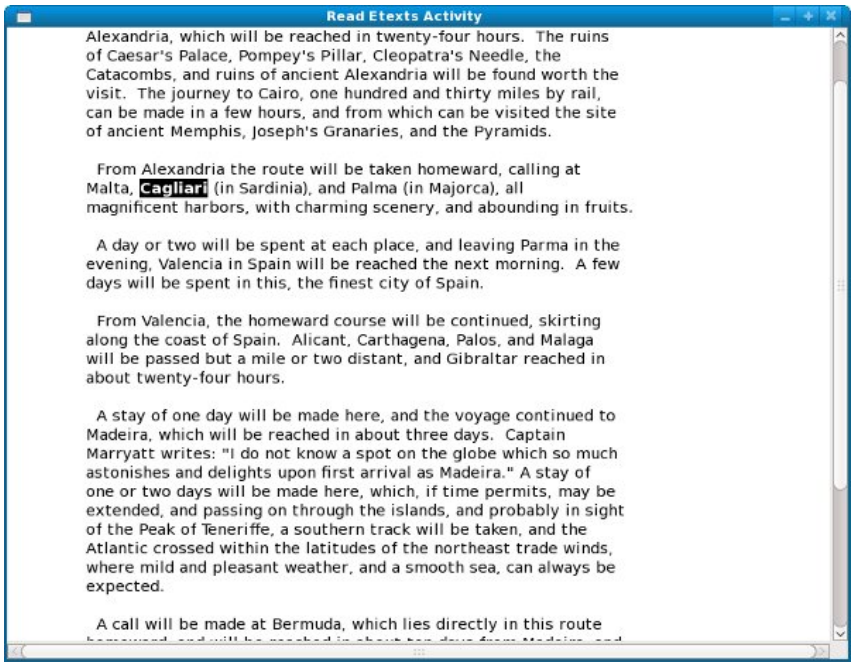
When espeak reads this file it will do a *callback* into our program every time it reads one of the mark tags. The callback will contain the number of the word in the **word_tuples** List which it will get from the **name** attribute of the **mark** tag. In this way the method being called will know which word to highlight. The advantage of using the mark name rather than just highlighting the next word in the textviewer is that if espeak should fail to do one of the callbacks the highlighting won't be thrown out of sync. This was a problem with speech-dispatcher.

A callback is just what it sounds like. When one program calls another program it can pass in a function or method of its own that it wants the second program to call when something happens.

To try out the new program run

```
./ReadEtextsTTS.py bookfile
```

from the Terminal. You can adjust pitch and rate up and down using the keys **7**, **8**, **9**, and **0** on the top row of the keyboard. It should say "Pitch Adjusted" or "Rate Adjusted" when you do that. You can start, pause, and resume speech with highlighting by using the **End** key on the keypad. (On the XO laptop the "game" keys are mapped to what is the numeric keypad on a normal keyboard. This makes these keys handy for use when the XO is folded into tablet mode and the keyboard is not available). You cannot change pitch or rate while speech is in progress. Attempts to do that will be ignored. The program in action looks like this:



That brings us to the end of the topic of Text to Speech. If you're like to see more, the Git repository for this book has a few more sample programs that use the gstreamer espeak plugin. These examples were created by the author of the plugin and demonstrate some other ways you can use it. There's even a "choir" program that demonstrates multiple voices speaking at the same time.

17. FUN WITH THE JOURNAL

INTRODUCTION

By default every Activity creates and reads one Journal entry. Most Activities don't need to do any more with the Journal than that, and if your Activity is like that you won't need the information in this chapter. Chances are that someday you will want to do more than that, so if you do keep reading.

First let's review what the Journal is. The Journal is a collection of files that each have **metadata** (data about data) associated with them. Metadata is stored as text strings and includes such things as the **Title**, **Description**, **Tags**, **MIME Type**, and a screen shot of the Activity when it was last used.

Your Activity cannot read and write these files directly. Instead Sugar provides an API (Application Programming Interface) that gives you an indirect way to add, delete and modify entries in the Journal, as well as a way to search Journal entries and make a list of entries that meet the search criteria.

The API we'll use is in the **datastore** package. After version .82 of Sugar this API was rewritten, so we'll need to learn how to support both versions in the same Activity.

If you've read this far you've seen several examples where Sugar started out doing one thing and then changed to do the same thing a better way but still provided a way to create Activities that would work with either the old or the new way. You may be wondering if it is normal for a project to do this. As a professional programmer I can tell you that doing tricks like this to maintain backward compatibility is extremely common, and Sugar does no more of this than any other project. There are decisions made by Herman Hollerith when he tabulated the 1890 census using punched cards that computer programmers must live with to this day.

INTRODUCING SUGAR COMMANDER

I am a big fan of the concept of the Journal but not so much of the user interface that Sugar uses to navigate through it and maintain it. My biggest gripe against it is that it represents the contents of thumb drives and SD cards as if the files on these were also Journal entries. My feeling is that files and directories are one thing and the Journal is another, and the user interface should recognize that.

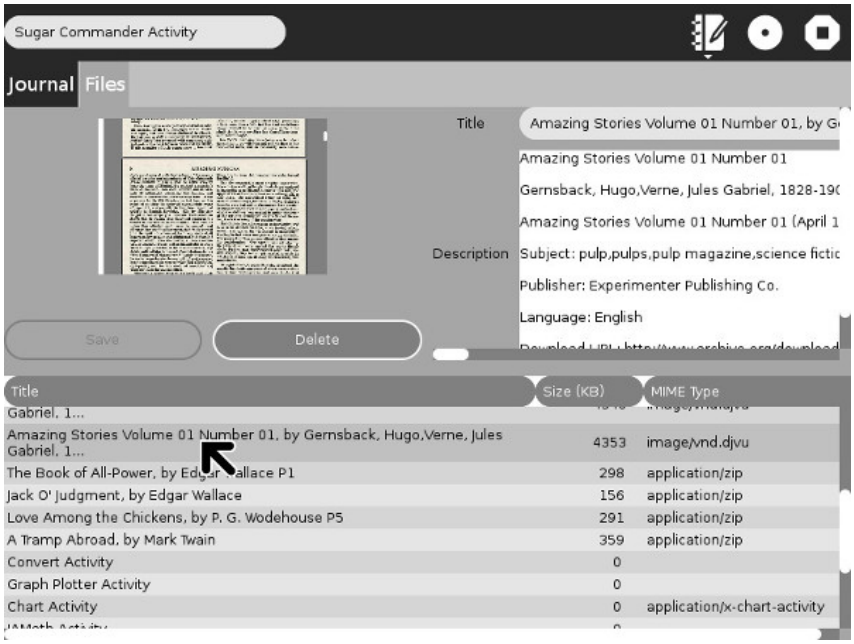
In the early days of Sugar the Journal was and was not an Activity. It inherited code from the Activity class just like any other Activity, and it was written in Python and used the same datastore API that other Activities used. However, it was run in a special way that gave it powers and abilities far beyond those of an ordinary Activity. In particular it could do two things:

- It could write to files on external media like thumb drives and SD cards.
- It alone could be used to resume Journal entries using other Activities.

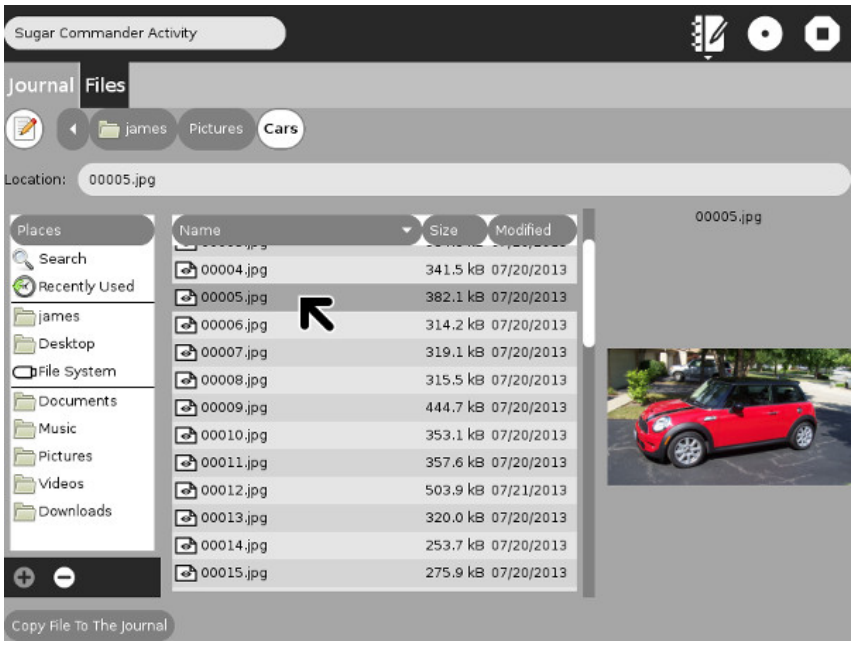
In recent versions of Sugar the Journal window no longer extends the Activity class, but it does still use the same datastore API that Activities use. In earlier versions of this book I called the Journal interface the Journal Activity, but it isn't correct to call it that anymore.

I wanted to write a Journal Activity that does everything the original did but has a user interface more to my own taste, but the Sugar security model won't allow that. Recently I came to the conclusion that a more mild-mannered version of the Journal Activity might be useful. Just as Kal-El sometimes finds it more useful to be Clark Kent than Superman, my own Activity might be a worthy alternative to the built-in Journal window when super powers are not needed.

My Activity, which I call **Sugar Commander**, has two tabs. One represents the Journal and looks like this:



This tab lets you browse through the Journal sorted by Title or MIME Type, select entries and view their details, update Title, Description or Tags, and delete entries you no longer want. The other tab shows files and folders and looks like this:



This tab lets you browse through the files and folders or the regular file system, including thumb drives and SD cards. You can select a file and make a Journal entry out of it by pushing the button at the bottom of the screen.

This Activity has very little code and still manages to do everything an ordinary Activity can do with the Journal. You can download the Git repository using this command:

```
git clone git://git.sugarlabs.org/sugar-commander/\
mainline.git
```

There is only one source file, **sugarcommander.py**:

```
import logging
import os
import time
from sugar3.activity import activity
from sugar3.activity.widgets import ActivityToolbar, StopButton
from gi.repository import Gtk
from gi.repository import Gdk
from gi.repository import cairo
from gi.repository import Pango
from gi.repository import PangoCairo
from gi.repository import GdkPixbuf
from sugar3.activity import widgets
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3 import mime
from sugar3.datastore import datastore
from sugar3.graphics.alert import NotifyAlert
from sugar3.graphics import style
from gettext import gettext as _
import pygame
import zipfile
from gi.repository import GObject
import dbus

COLUMN_TITLE = 0
COLUMN_SIZE = 1
COLUMN_MIME = 2
COLUMN_JOBJECT = 3

ARBITRARY_LARGE_HEIGHT = 10000
JPEG_QUALITY = 80

DS_DBUS_SERVICE = 'org.laptop.sugar.DataStore'
DS_DBUS_INTERFACE = 'org.laptop.sugar.DataStore'
DS_DBUS_PATH = '/org/laptop/sugar/DataStore'

_logger = logging.getLogger('sugar-commander')
```

```
class SugarCommander(activity.Activity):
    def __init__(self, handle, create_jobject=True):
        "The entry point to the Activity"
        activity.Activity.__init__(self, handle)
        self.selected_journal_entry = None
        self.selected_path = None
        self.update_log_entries = ''
        self.close_requested = False

        canvas = Gtk.Notebook()
        canvas.props.show_border = True
        canvas.props.show_tabs = True
        canvas.show()

        self.ls_journal = Gtk.ListStore(str,  GObject.TYPE_UINT64,
str, \
           GObject.TYPE_PYOBJECT)
        self.tv_journal = Gtk.TreeView(self.ls_journal)
        self.tv_journal.set_rules_hint(True)
        self.tv_journal.set_search_column(COLUMN_TITLE)
        self.selection_journal = self.tv_journal.get_selection()
        self.selection_journal.set_mode(Gtk.SelectionMode.SINGLE)
        self.selection_journal.connect("changed",
self.selection_journal_cb)
        renderer = Gtk.CellRendererText()
        renderer.set_property('wrap-mode', Pango.WrapMode.WORD)
        renderer.set_property('wrap-width', 500)
        renderer.set_property('width', 500)
        self.col_journal = Gtk.TreeViewColumn(_('Title'), renderer,
                                              text=COLUMN_TITLE)
        self.col_journal.set_sort_column_id(COLUMN_TITLE)
        self.tv_journal.append_column(self.col_journal)

        size_renderer = Gtk.CellRendererText()
        size_renderer.set_property('width', 100)
        size_renderer.set_property('alignment', Pango.Alignment.RIGHT)
        size_renderer.set_property('xalign', 0.8)
        self.col_size = Gtk.TreeViewColumn(_('Size (KB)'),
size_renderer,
                                           text=COLUMN_SIZE)
        self.col_size.set_sort_column_id(COLUMN_SIZE)
        self.tv_journal.append_column(self.col_size)

        mime_renderer = Gtk.CellRendererText()
        mime_renderer.set_property('width', 200)
        self.col_mime = Gtk.TreeViewColumn(_('MIME Type'),
mime_renderer,
                                           text=COLUMN_MIME)
        self.col_mime.set_sort_column_id(COLUMN_MIME)
        self.tv_journal.append_column(self.col_mime)

        self.list_scroller_journal = Gtk.ScrolledWindow(
                        hadjustment=None, vadjustment=None)
        self.list_scroller_journal.set_policy(
                    Gtk.PolicyType.AUTOMATIC,
Gtk.PolicyType.AUTOMATIC)
        self.list_scroller_journal.add(self.tv_journal)

        tab1_label = Gtk.Label()
        tab1_label.set_markup("<span foreground='#FFF'" \
                        " size='14000'>" +_("Journal") + "</span>")
        tab1_label.show()
        self.tv_journal.show()
        self.list_scroller_journal.show()

        column_table = Gtk.Table(2,  2,  False)
        column_table.set_col_spacings(10)
        column_table.set_row_spacings(10)

        image_table = Gtk.Table(2,  2,  False)

        self.btn_resize = Gtk.Button(_("Resize To Width"))
        self.btn_resize.connect('button_press_event',
                                self.resize_button_press_event_cb)
        image_table.attach(self.btn_resize,  0, 1, 2, 3)

        self.resize_width_entry = Gtk.Entry()
        self.resize_width_entry.set_max_length(4)
        image_table.attach(self.resize_width_entry, 1, 2, 2, 3)
        self.resize_width_entry.set_text('600')
        self.resize_width_entry.connect('key_press_event',
                                        self.resize_key_press_event_cb)

        self.btn_save = Gtk.Button(_("Save"))
        self.btn_save.connect('button_press_event',
                              self.save_button_press_event_cb)
        image_table.attach(self.btn_save,  0, 1, 3, 4)
        self.btn_save.props.sensitive = False
```

```
        self.btn_save.show()

        self.btn_delete = Gtk.Button(_("Delete"))
        self.btn_delete.connect('button_press_event',
                                self.delete_button_press_event_cb)
        image_table.attach(self.btn_delete,  1, 2, 3, 4)
        self.btn_delete.props.sensitive = False
        self.btn_delete.show()

        self.image = Gtk.Image()
        image_table.set_col_spacings(10)
        image_table.set_row_spacings(10)
        image_table.attach(self.image, 0, 2, 0, 1)

        self.dimension_label = Gtk.Label("")
        image_table.attach(self.dimension_label,  0, 2, 1, 2)

        entry_table = Gtk.Table(2,  2,  False)
        entry_table.set_col_spacings(10)
        entry_table.set_row_spacings(10)

        title_label = Gtk.Label(_("Title"))
        entry_table.attach(title_label, 0, 1, 0, 1)
        title_label.show()

        self.title_entry = Gtk.Entry()
        entry_table.attach(self.title_entry, 1, 2, 0, 1)
        self.title_entry.connect('key_press_event',
                                 self.key_press_event_cb)
        self.title_entry.show()

        description_label = Gtk.Label(_("Description"))
        entry_table.attach(description_label, 0, 1, 1, 2)
        description_label.show()

        self.description_textview = Gtk.TextView()
        self.description_textview.set_wrap_mode(Pango.WrapMode.WORD)
        entry_table.attach(self.description_textview, 1, 2, 1, 2)
        self.description_textview.props.accepts_tab = False
        self.description_textview.connect('key_press_event',
                                          self.key_press_event_cb)
        self.description_textview.show()

        tags_label = Gtk.Label(_("Tags"))
        entry_table.attach(tags_label, 0, 1, 2, 3)
        tags_label.show()

        self.tags_textview = Gtk.TextView()
        self.tags_textview.set_wrap_mode(Pango.WrapMode.WORD)
        entry_table.attach(self.tags_textview, 1, 2, 2, 3)
        self.tags_textview.props.accepts_tab = False
        self.tags_textview.connect('key_press_event',
                                   self.key_press_event_cb)
        self.tags_textview.show()

        entry_table.show()

        scroller_image = Gtk.ScrolledWindow(
                         hadjustment=None, vadjustment=None)
        scroller_image.set_hexpand(False)
        scroller_image.set_vexpand(True)
        scroller_image.add_with_viewport(image_table)
        scroller_image.show()

        self.scroller_entry = Gtk.ScrolledWindow(
                         hadjustment=None, vadjustment=None)
        self.scroller_entry.set_hexpand(False)
        self.scroller_entry.set_vexpand(True)
        self.scroller_entry.add_with_viewport(entry_table)
        self.scroller_entry.show()

        column_table.attach(scroller_image,  0, 1, 0, 1)

        column_table.attach(self.scroller_entry,  1, 2, 0, 1)

        image_table.show()
        column_table.show()
        self.btn_resize.hide()
        self.resize_width_entry.hide()

        vbox = Gtk.VBox(homogeneous=True, spacing=5)
        vbox.pack_start(column_table,  expand=True, fill=True,
padding=0)
        vbox.pack_end(self.list_scroller_journal, expand=True,
fill=True, \
            padding=0)

        canvas.append_page(vbox,  tab1_label)
```

```
        self._filechooser = Gtk.FileChooserWidget(
            action=Gtk.FileChooserAction.OPEN)
        self._filechooser.set_current_folder("/media")
        self.copy_button = Gtk.Button(_("Copy File To The Journal"))
        self.copy_button.connect('clicked',  self.create_journal_entry)
        self.copy_button.show()
        self._filechooser.set_extra_widget(self.copy_button)
        preview = Gtk.Image()
        self._filechooser.set_preview_widget(preview)
        self._filechooser.connect("update-preview",
                                  self.update_preview_cb, preview)
        tab2_label = Gtk.Label()
        tab2_label.set_markup("<span foreground='#FFF'" \
                              " size='14000'>" + _("Files")  + "</span>")
        tab2_label.show()
        canvas.append_page(self._filechooser,  tab2_label)

        self.set_canvas(canvas)
        self.show_all()
        self.btn_resize.hide()
        self.resize_width_entry.hide()
        self.dimension_label.hide()

        toolbox = ActivityToolbar(self)

        stop_button = StopButton(self)
        stop_button.show()
        toolbox.insert(stop_button, -1)

        self.set_toolbar_box(toolbox)
        toolbox.show()

        self.load_journal_table()

        bus = dbus.SessionBus()
        remote_object = bus.get_object(DS_DBUS_SERVICE, DS_DBUS_PATH)
        _datastore = dbus.Interface(remote_object, DS_DBUS_INTERFACE)
        _datastore.connect_to_signal('Created',
self.datastore_created_cb)
        _datastore.connect_to_signal('Updated',
self.datastore_updated_cb)
        _datastore.connect_to_signal('Deleted',
self.datastore_deleted_cb)

        self.selected_journal_entry = None

    def update_preview_cb(self, file_chooser, preview):
        filename = file_chooser.get_preview_filename()
        try:
            file_mimetype = mime.get_for_file(filename)
            if file_mimetype.startswith('image/') \
                and file_mimetype != 'image/vnd.djvu':
                pixbuf =
GdkPixbuf.Pixbuf.new_from_file_at_size(filename,
                                       style.zoom(320),
style.zoom(240))
                preview.set_from_pixbuf(pixbuf)
                have_preview = True
            elif file_mimetype  == 'application/x-cbz':
                fname = self.extract_image(filename)
                pixbuf = GdkPixbuf.Pixbuf.new_from_file_at_size(fname,
                                       style.zoom(320),
style.zoom(240))
                preview.set_from_pixbuf(pixbuf)
                have_preview = True
                os.remove(fname)
            else:
                have_preview = False
        except:
            have_preview = False
        file_chooser.set_preview_widget_active(have_preview)
        return

    def key_press_event_cb(self, entry, event):
        self.btn_save.props.sensitive = True

    def resize_key_press_event_cb(self, entry, event):
        keyname = Gtk.Gdk.keyval_name(event.keyval)
        if ((keyname < '0' or keyname > '9') and keyname !=
'BackSpace'
            and keyname != 'Left' and keyname != 'Right'
            and keyname != 'KP_Left' and keyname != 'KP_Right'
            and keyname != 'Delete' and keyname != 'End'
            and keyname != 'KP_End' and keyname != 'Home'
            and keyname != 'KP_Home' and keyname != 'KP_Delete'):
            return True
        else:
```

```
            return False

    def resize_button_press_event_cb(self, entry, event):
        jobject = self.selected_journal_entry
        filename = jobject.get_file_path()
        im = pygame.image.load(filename)
        image_width, image_height = im.get_size()
        resize_to_width = int(self.resize_width_entry.get_text())
        if (image_width < resize_to_width):
            self.alert(_('Error'), \
                _('Image cannot be made larger, only smaller.'))
            return
        tempfile = os.path.join(self.get_activity_root(), 'instance',
                                'tmp%i' % time.time())
        try:
            scaled_pixbuf = GdkPixbuf.Pixbuf.new_from_file_at_size( \
                filename,resize_to_width, ARBITRARY_LARGE_HEIGHT)
            scaled_pixbuf.save(tempfile, "jpeg", \
                {"quality":"%d" % JPEG_QUALITY})
        except:
            print 'File could not be converted'
            return

        jobject.file_path = tempfile
        jobject.metadata['mime_type'] = 'image/jpeg'
        im = pygame.image.load(tempfile)
        image_width, image_height = im.get_size()
        self.dimension_label.set_text(str(image_width) + "x" +
            str(image_height))
        self.dimension_label.show()
        datastore.write(jobject, update_mtime=False,
                        reply_handler=self.datastore_write_cb,
                        error_handler=self.datastore_write_error_cb)
        title = jobject.metadata.get('title', None)
        self.update_log_entries += '\n' + _('Entry %s resized.') %
title

    def save_button_press_event_cb(self, entry, event):
        self.update_entry()

    def delete_button_press_event_cb(self, entry, event):
        datastore.delete(self.selected_journal_entry.object_id)

    def datastore_created_cb(self, uid):
        new_jobject = datastore.get(uid)
        iter = self.ls_journal.append()
        title = new_jobject.metadata['title']
        self.ls_journal.set(iter, COLUMN_TITLE, title)
        mime = new_jobject.metadata['mime_type']
        self.ls_journal.set(iter, COLUMN_MIME, mime)
        self.ls_journal.set(iter, COLUMN_JOBJECT, new_jobject)
        size = self.get_size(new_jobject) / 1024
        self.ls_journal.set(iter, COLUMN_SIZE, size)

    def datastore_updated_cb(self,  uid):
        new_jobject = datastore.get(uid)
        iter = self.ls_journal.get_iter_first()
        for row in self.ls_journal:
            jobject = row[COLUMN_JOBJECT]
            if jobject.object_id == uid:
                title = jobject.metadata['title']
                self.ls_journal.set(iter, COLUMN_TITLE, title)
                mime = jobject.metadata['mime_type']
                self.ls_journal.set(iter, COLUMN_MIME, mime)
                size = self.get_size(jobject) / 1024
                self.ls_journal.set(iter, COLUMN_SIZE, size)
            iter = self.ls_journal.iter_next(iter)
        object_id = self.selected_journal_entry.object_id
        if object_id == uid:
            self.set_form_fields(new_jobject)

    def datastore_deleted_cb(self,  uid):
        save_path = self.selected_path
        iter = self.ls_journal.get_iter_first()
        for row in self.ls_journal:
            jobject = row[COLUMN_JOBJECT]
            if jobject.object_id == uid:
                title = jobject.metadata.get('title', None)
                self.update_log_entries += '\n' + \
                    _('Entry %s deleted.') % title
                self.ls_journal.remove(iter)
                break
            iter = self.ls_journal.iter_next(iter)


        try:
            self.selection_journal.select_path(save_path)
            self.tv_journal.grab_focus()
```

```
        except:
            self.title_entry.set_text('')
            description_textbuffer =
self.description_textview.get_buffer()
            description_textbuffer.set_text('')
            tags_textbuffer = self.tags_textview.get_buffer()
            tags_textbuffer.set_text('')
            self.btn_save.props.sensitive = False
            self.btn_delete.props.sensitive = False
            self.image.clear()
            self.image.show()

    def update_entry(self):
        needs_update = False

        if self.selected_journal_entry is None:
            return

        object_id = self.selected_journal_entry.object_id
        jobject = datastore.get(object_id)

        old_title = jobject.metadata.get('title', None)
        if old_title != self.title_entry.props.text:
            jobject.metadata['title'] = self.title_entry.props.text
            jobject.metadata['title_set_by_user'] = '1'
            self.update_log_entries += '\n' + \
                _('Entry title changed to %s') %
self.title_entry.props.text
            needs_update = True

        old_tags = jobject.metadata.get('tags', None)
        new_tags = self.tags_textview.props.buffer.props.text
        if old_tags != new_tags:
            jobject.metadata['tags'] = new_tags
            self.update_log_entries += '\n' + \
                _('Entry %s tags updated.') %
self.title_entry.props.text
            needs_update = True

        old_description = jobject.metadata.get('description', None)
        new_description =
self.description_textview.props.buffer.props.text
        if old_description != new_description:
            jobject.metadata['description'] = new_description
            self.update_log_entries += '\n' + \
                 _('Entry %s description updated.') % \
                 self.title_entry.props.text
            needs_update = True

        if needs_update:
            datastore.write(jobject, update_mtime=False,
                            reply_handler=self.datastore_write_cb,

error_handler=self.datastore_write_error_cb)
        self.btn_save.props.sensitive = False

    def datastore_write_cb(self):
        pass

    def datastore_write_error_cb(self, error):
        logging.error('sugarcommander.datastore_write_error_cb: %r' %
error)

    def close(self, skip_save=False):
        activity.Activity.close(self, False)

    def read_file(self, file_path):
        """Load a file from the datastore on activity start"""
        _logger.debug('sugarcommander.read_file: %s', file_path)

    def write_file(self, filename):
        "Save meta data for the file."
        old_description = self.metadata.get('description',  \
            'Sugar Commander log:')
        new_description = old_description + self.update_log_entries
        self.metadata['description'] = new_description
        self.metadata['mime_type'] = 'text/plain'
        f = open(filename, 'w')
        try:
            f.write(new_description)
        finally:
            f.close()
        self.update_log_entries = ''

    def can_close(self):
        self.close_requested = True
        return True
```

```
    def selection_journal_cb(self, selection):
        self.btn_delete.props.sensitive = True
        tv = selection.get_tree_view()
        model = tv.get_model()
        sel = selection.get_selected()
        if sel:
            model, iter = sel
            jobject = model.get_value(iter,COLUMN_JOBJECT)
            jobject = datastore.get(jobject.object_id)
            self.selected_journal_entry = jobject
            self.set_form_fields(jobject)
            if jobject.metadata['mime_type'] .startswith('image/')  \
                and jobject.metadata['mime_type'] != 'image/vnd.djvu':
                self.btn_resize.show()
                self.resize_width_entry.show()
                filename = jobject.get_file_path()
                im = pygame.image.load(filename)
                image_width, image_height = im.get_size()
                self.dimension_label.set_text(str(image_width) + "x" +
                    str(image_height))
                self.dimension_label.show()
            else:
                self.btn_resize.hide()
                self.resize_width_entry.hide()
                self.dimension_label.hide()
            self.selected_path = model.get_path(iter)

    def set_form_fields(self, jobject):
        self.title_entry.set_text(jobject.metadata['title'])
        description_textbuffer = self.description_textview.get_buffer()
        if 'description' in jobject.metadata:

description_textbuffer.set_text(jobject.metadata['description'])
        else:
            description_textbuffer.set_text('')
        tags_textbuffer = self.tags_textview.get_buffer()
        if 'tags' in jobject.metadata:
            tags_textbuffer.set_text(jobject.metadata['tags'])
        else:
            tags_textbuffer.set_text('')
        self.create_preview(jobject.object_id)

    def create_preview(self,  object_id):
        jobject = datastore.get(object_id)

        if 'preview' in jobject.metadata:
            preview = jobject.metadata['preview']
            if preview is None or preview == '' or preview == 'None':
                if jobject.metadata['mime_type'] .startswith('image/')
and \
                    jobject.metadata['mime_type'] != 'image/vnd.djvu':
                    filename = jobject.get_file_path()
                    self.show_image(filename)
                    return
                if jobject.metadata['mime_type']  == 'application/x-
cbz':
                    filename = jobject.get_file_path()
                    fname = self.extract_image(filename)
                    self.show_image(fname)
                    os.remove(fname)
                    return
                self.show_image('xoimage.jpg')
                return

        if 'preview' in jobject.metadata and \
                len(jobject.metadata['preview']) > 4:
            preview_data = jobject.metadata['preview']
            loader = GdkPixbuf.PixbufLoader()
            loader.write(preview_data)
            scaled_buf = loader.get_pixbuf()
            loader.close()
            self.image.set_from_pixbuf(scaled_buf)
            self.image.show()
        else:
            self.image.clear()
            self.image.show()

    def load_journal_table(self):
        self.btn_save.props.sensitive = False
        self.btn_delete.props.sensitive = False
        query = {}
        ds_objects, num_objects = datastore.find(query,
properties=['uid',
            'title',  'mime_type'])

        self.ls_journal.clear()
        for i in xrange (0, num_objects, 1):
            iter = self.ls_journal.append()
```

```
            title = ds_objects[i].metadata['title']
            self.ls_journal.set(iter, COLUMN_TITLE, title)
            mime = ds_objects[i].metadata['mime_type']
            self.ls_journal.set(iter, COLUMN_MIME, mime)
            self.ls_journal.set(iter, COLUMN_JOBJECT, ds_objects[i])
            size = self.get_size(ds_objects[i]) / 1024
            self.ls_journal.set(iter, COLUMN_SIZE, size)

        v_adjustment = self.list_scroller_journal.get_vadjustment()
        v_adjustment.value = 0

    def get_size(self, jobject):
        """Return the file size for a Journal object."""
        logging.debug('get_file_size %r', jobject.object_id)
        path = jobject.get_file_path()
        if not path:
            return 0

        return os.stat(path).st_size

    def create_journal_entry(self,  widget,  data=None):
        filename = self._filechooser.get_filename()
        journal_entry = datastore.create()
        journal_entry.metadata['title'] =
self.make_new_filename(filename)
        journal_entry.metadata['title_set_by_user'] = '1'
        journal_entry.metadata['keep'] = '0'
        file_mimetype = mime.get_for_file(filename)
        if not file_mimetype is None:
            journal_entry.metadata['mime_type'] = file_mimetype
        journal_entry.metadata['buddies'] = ''
        if file_mimetype.startswith('image/')  and file_mimetype != \
            'image/vnd.djvu':
            preview = self.create_preview_metadata(filename)
        elif file_mimetype  == 'application/x-cbz':
            fname = self.extract_image(filename)
            preview = self.create_preview_metadata(fname)
            os.remove(fname)
        else:
            preview = ''
        if not preview  == '':
            journal_entry.metadata['preview'] =
dbus.ByteArray(preview)
        else:
            journal_entry.metadata['preview'] =  ''

        journal_entry.file_path = filename
        datastore.write(journal_entry)
        self.update_log_entries += '\n' + \
            _('File %s copied to the Journal.') % filename
        self.alert(_('Success'),  _('%s added to Journal.')
                   % self.make_new_filename(filename))

    def alert(self, title, text=None):
        alert = NotifyAlert(timeout=20)
        alert.props.title = title
        alert.props.msg = text
        self.add_alert(alert)
        alert.connect('response', self.alert_cancel_cb)
        alert.show()

    def alert_cancel_cb(self, alert, response_id):
        self.remove_alert(alert)

    def show_image(self, filename):
        "display a resized image in a preview"
        scaled_buf = GdkPixbuf.Pixbuf.new_from_file_at_size(filename,
                                       style.zoom(300),
style.zoom(225))
        self.image.set_from_pixbuf(scaled_buf)
        self.image.show()

    def extract_image(self,  filename):
        zf = zipfile.ZipFile(filename, 'r')
        image_files = zf.namelist()
        image_files.sort()
        file_to_extract = image_files[0]
        extract_new_filename = self.make_new_filename(file_to_extract)
        if extract_new_filename is None or extract_new_filename == '':
            # skip over directory name if the images are in a
subdirectory.
            file_to_extract = image_files[1]
            extract_new_filename =
self.make_new_filename(file_to_extract)

        if len(image_files) > 0:
            if self.save_extracted_file(zf, file_to_extract):
                fname = os.path.join(self.get_activity_root(),
```

```
'instance',
                                    extract_new_filename)
                return fname

    def save_extracted_file(self, zipfile, filename):
        "Extract the file to a temp directory for viewing"
        try:
            filebytes = zipfile.read(filename)
        except zipfile.BadZipfile, err:
            print 'Error opening the zip file: %s' % (err)
            return False
        except KeyError,  err:
            self.alert('Key Error', 'Zipfile key not found: '
                       + str(filename))
            return
        outfn = self.make_new_filename(filename)
        if (outfn == ''):
            return False
        fname = os.path.join(self.get_activity_root(), 'instance',
outfn)
        f = open(fname, 'w')
        try:
            f.write(filebytes)
        finally:
            f.close()
        return True

    def make_new_filename(self, filename):
        partition_tuple = filename.rpartition('/')
        return partition_tuple[2]

    def create_preview_metadata(self,  filename):

        file_mimetype = mime.get_for_file(filename)
        if not file_mimetype.startswith('image/'):
            return ''

        scaled_pixbuf =
GdkPixbuf.Pixbuf.new_from_file_at_size(filename,  \
                                    style.zoom(320),
style.zoom(240))
        preview_data = []

        succes, preview_data = scaled_pixbuf.save_to_bufferv('png',
[], [])
        preview_data = ''.join(preview_data)

        return preview_data
```

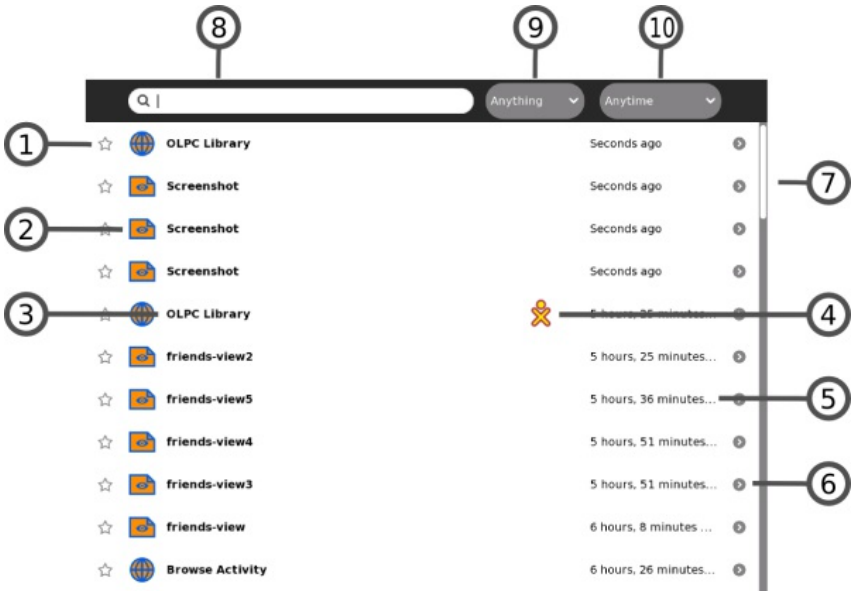
Let's look at this code one method at a time.

ADDING A JOURNAL ENTRY

We add a Journal entry when someone pushes a button on the gtk.FileChooser. This is the code that gets run:

```
    def create_journal_entry(self, widget, data=None):
        filename = self._filechooser.get_filename()
        journal_entry = datastore.create()
        journal_entry.metadata['title'] = \
            self.make_new_filename(
            filename)
        journal_entry.metadata['title_set_by_user'] = '1'
        journal_entry.metadata['keep'] = '0'
        file_mimetype = mime.get_for_file(filename)
        if not file_mimetype is None:
            journal_entry.metadata['mime_type'] = \
                file_mimetype
        journal_entry.metadata['buddies'] = ''
        if file_mimetype.startswith('image/'):
            preview = self.create_preview_metadata(filename)
        elif file_mimetype  == 'application/x-cbz':
            fname = self.extract_image(filename)
            preview = self.create_preview_metadata(fname)
            os.remove(fname)
        else:
            preview = ''
        if not preview  == '':
            journal_entry.metadata['preview'] = \
                dbus.ByteArray(preview)
        else:
            journal_entry.metadata['preview'] =  ''
        journal_entry.file_path = filename
        datastore.write(journal_entry)
```

The only thing worth commenting on here is the metadata. **title** is what appears as #3 in the picture below. **title_set_by_user** is set to 1 so that the Activity won't prompt the user to change the title when the Activity closes. **keep** refers to the little star that appears at the beginning of the Journal entry (see #1 in the picture below). Highlight it by setting this to 1, otherwise set to 0. **buddies** is a list of users that collaborated on the Journal entry, and in this case there aren't any (these show up as #4 in the picture below).



preview is an image file in the PNG format that is a screenshot of the Activity in action. This is created by the Activity itself when it is run so there is no need to make one when you add a Journal entry. You can simply use an empty string ('') for this property.

Because previews are much more visible in Sugar Commander than they are in the regular Journal Activity I decided that Sugar Commander should make a preview image for image files and comic books as soon as they are added to the Journal. To do this I made a pixbuf of the image that would fit within the scaled dimensions of 320x240 pixels and made a **dbus.ByteArray** out of it, which is the format that the Journal uses to store preview images.

mime_type describes the format of the file and is generally assigned based on the filename suffix. For instance, files ending in .html have a MIME type of 'text/html'. Python has a package called **mimetypes** that takes a file name and figures out what its MIME type should be, but Sugar provides its own package to do the same thing. For most files either one would give the correct answer, but Sugar has its own MIME types for things like Activity bundles, etc. so for best results you really should use Sugar's mime package. You can import it like this:

```
from sugar import mime
```

The rest of the metadata (icon, modified time) is created automatically.

NOT ADDING A JOURNAL ENTRY

*Note: the technique described in this section is broken in some versions of Sugar. If you use this technique your Activity will not work on those versions. The technique is documented and **should** be supported in every version of Sugar, but there is a certain risk in using it.*

In addition to the risk, there is a school of thought that Activities should always leave behind a Journal entry. The idea is that the Journal is not just a place for data, but is the equivalent of a personal journal that a child might keep about his school work. If you subscribe to that idea then every thing the child does should have a Journal entry so the child can enter notes about it.

The current version of Sugar starts Activities from the Activity ring using their most recent Journal entry by default. This makes the problem of extra journal entries left behind by Activities that don't really need journal entries less of a problem than it has been in the past.

The code in this book leaves behind a Journal entry with a log of what has been done with the Activity. You can easily fix it to not leave behind a Journal entry using the technique in this section.

Sugar Activities by default create a Journal entry using the *write_file()* method. There will be Activities that don't need to do this. For instance, **Get Internet Archive Books** downloads e-books to the Journal, but has no need for a Journal entry of its own. The same thing is true of **Sugar Commander**. You might make a game that keeps track of high scores. You could keep those scores in a Journal entry, but that would require players to resume the game from the Journal rather than just starting it up from the Activity Ring. For that reason you might prefer to store the high scores in a file in the **data** directory rather than the Journal, and not leave a Journal entry behind at all.

Sugar gives you a way to do that. First you need to specify an extra argument in your Activity's *__init__()* method like this:

```
class SugarCommander(activity.Activity):
    def __init__(self, handle, create_jobject=True):
        "The entry point to the Activity"
        activity.Activity.__init__(self, handle, False)
```

Second, you need to override the *close()* method like this:

```
    def close(self, skip_save=False):
        "Override the close method so we don't try to
        create a Journal entry."
        activity.Activity.close(self, True)
```

That's all there is to it.

LISTING OUT JOURNAL ENTRIES

If you need to list out Journal entries you can use the *find()* method of **datastore**. The find method takes an argument containing search criteria. If you want to search for image files you can search by mime-type using a statement like this:

```
        ds_objects, num_objects = datastore.find(
            {'mime_type':['image/jpeg',
            'image/gif', 'image/tiff', 'image/png']},
            properties=['uid',
            'title', 'mime_type']))
```

You can use any metadata attribute to search on. If you want to list out everything in the Journal you can use an empty search criteria like this:

```
        ds_objects, num_objects = datastore.find({},
            properties=['uid',
```

```
            'title', 'mime_type']))
```

The properties argument specifies what metadata to return for each object in the list. You should limit these to what you plan to use, but always include **uid**. One thing you should *never* include in a list is **preview**. This is an image file showing what the Activity for the Journal object looked like when it was last used. If for some reason you need this there is a simple way to get it for an individual Journal object, but you never want to include it in a list because it will slow down your Activity enormously.

Here is code in **Sugar Commander** that lists Journal entries:

```
    def load_journal_table(self):
        self.btn_save.props.sensitive = False
        self.btn_delete.props.sensitive = False
        query = {}
        ds_objects, num_objects = datastore.find(query,
properties=['uid',
            'title',  'mime_type'])

        self.ls_journal.clear()
        for i in xrange (0, num_objects, 1):
            iter = self.ls_journal.append()
            title = ds_objects[i].metadata['title']
            self.ls_journal.set(iter, COLUMN_TITLE, title)
            mime = ds_objects[i].metadata['mime_type']
            self.ls_journal.set(iter, COLUMN_MIME, mime)
            self.ls_journal.set(iter, COLUMN_JOBJECT, ds_objects[i])
            size = self.get_size(ds_objects[i]) / 1024
            self.ls_journal.set(iter, COLUMN_SIZE, size)

        v_adjustment = self.list_scroller_journal.get_vadjustment()
        v_adjustment.value = 0
```

USING JOURNAL ENTRIES

When you're ready to read a file stored in a Journal object you can use the *get_file_path()* method of the Journal object to get a file path and open it for reading, like this:

```
        fname = jobject.get_file_path()
```

One word of caution: be aware that this path does not exist until you call *get_file_path()* and will not exist long after. With the Journal you work with copies of files in the Journal, not the originals. For that reason you don't want to store the return value of *get_file_path()* for later use because later it may not be valid. Instead, store the Journal object itself and call the method right before you need the path.

Metadata entries for Journal objects generally contain strings and work the way you would expect, with one exception, which is the **preview**.

```
    def create_preview(self,  object_id):
        jobject = datastore.get(object_id)

        if jobject.metadata.has_key('preview'):
            preview = jobject.metadata['preview']
            if preview is None or preview == '' or
                preview == 'None':
                if jobject.metadata['mime_type'].startswith(
                    'image/'):
                    filename = jobject.get_file_path()
                    self.show_image(filename)
                    return
                if jobject.metadata['mime_type']  == \
                    'application/x-cbz':
                    filename = jobject.get_file_path()
                    fname = self.extract_image(filename)
                    self.show_image(fname)
                    os.remove(fname)
                    return

        if jobject.metadata.has_key('preview') and \
                len(jobject.metadata['preview']) > 4:

            if jobject.metadata['preview'][1:4] == 'PNG':
                preview_data = jobject.metadata['preview']
```

```
            else:
                import base64
                preview_data = base64.b64decode(
                    jobject.metadata['preview'])

            loader = gtk.gdk.PixbufLoader()
            loader.write(preview_data)
            scaled_buf = loader.get_pixbuf()
            loader.close()
            self.image.set_from_pixbuf(scaled_buf)
            self.image.show()
        else:
            self.image.clear()
            self.image.show()
```

We should never request **preview** as metadata to be returned in our list of Journal objects. We'll need to get a complete copy of the Journal object to get it. Since we already have a Journal object we can get the complete Journal object by getting its **object id** then requesting a new copy from the datastore using the id.

The code you would use to get a complete copy of a Journal object looks like this:

```
        object_id = jobject.object_id
        jobject = datastore.get(object_id)
```

To get a preview image use code like this:

```
        if 'preview' in jobject.metadata and \
                len(jobject.metadata['preview']) > 4:
            preview_data = jobject.metadata['preview']
            loader = GdkPixbuf.PixbufLoader()
            loader.write(preview_data)
            scaled_buf = loader.get_pixbuf()
            loader.close()
            self.image.set_from_pixbuf(scaled_buf)
            self.image.show()
```

UPDATING A JOURNAL OBJECT

The code to update a Journal object looks like this:

```
    def update_entry(self):
        needs_update = False

        if self.selected_journal_entry is None:
            return

        object_id = self.selected_journal_entry.object_id
        jobject = datastore.get(object_id)

        old_title = jobject.metadata.get('title', None)
        if old_title != self.title_entry.props.text:
            jobject.metadata['title'] = \
                self.title_entry.props.text
            jobject.metadata['title_set_by_user'] = '1'
            needs_update = True

        old_tags = jobject.metadata.get('tags', None)
        new_tags = \
            self.tags_textview.props.buffer.props.text
        if old_tags != new_tags:
            jobject.metadata['tags'] = new_tags
            needs_update = True

        old_description = \
            jobject.metadata.get('description', None)
        new_description = \
            self.description_textview.props.buffer.props.text
        if old_description != new_description:
            jobject.metadata['description'] = \
                new_description
            needs_update = True

        if needs_update:
            datastore.write(jobject, update_mtime=False,
                reply_handler=self.datastore_write_cb,
                error_handler=self.datastore_write_error_cb)
        self.btn_save.props.sensitive = False

    def datastore_write_cb(self):
```

```
        pass

    def datastore_write_error_cb(self, error):
        logging.error(
            'sugarcommander.datastore_write_error_cb:'
            ' %r' % error)
```

DELETING A JOURNAL ENTRY

The code to delete a Journal entry is this:

```
    def delete_button_press_event_cb(self, entry, event):
        datastore.delete(
            self.selected_journal_entry.object_id)
```

GETTING CALLBACKS FROM THE JOURNAL USING D-BUS

In the chapter on **Making Shared Activities** we saw how D-Bus calls sent over Telepathy Tubes could be used to send messages from an Activity running on one computer to the same Activity running on a different computer. D-Bus is not normally used that way; typically it is used to send messages between programs running on the same computer.

For example, if you're working with the Journal you can get callbacks whenever the Journal is updated. You get the callbacks whether the update was done by your Activity or elsewhere. If it is important for your Activity to know when the Journal has been updated you'll want to get these callbacks.

The first thing you need to do is define some constants and import the dbus package:

```
DS_DBUS_SERVICE = 'org.laptop.sugar.DataStore'
DS_DBUS_INTERFACE = 'org.laptop.sugar.DataStore'
DS_DBUS_PATH = '/org/laptop/sugar/DataStore'
import dbus
```

Next, in your __init__() method put code to connect to the signals and do the callbacks:

```
        bus = dbus.SessionBus()
        remote_object = bus.get_object(
            DS_DBUS_SERVICE, DS_DBUS_PATH)
        _datastore = dbus.Interface(remote_object,
            DS_DBUS_INTERFACE)
        _datastore.connect_to_signal('Created',
            self._datastore_created_cb)
        _datastore.connect_to_signal('Updated',
            self._datastore_updated_cb)
        _datastore.connect_to_signal('Deleted',
            self._datastore_deleted_cb)
```

The methods being run by the callbacks might look something like this:

```
    def datastore_created_cb(self, uid):
        new_jobject = datastore.get(uid)
        iter = self.ls_journal.append()
        title = new_jobject.metadata['title']
        self.ls_journal.set(iter,
            COLUMN_TITLE, title)
        mime = new_jobject.metadata['mime_type']
        self.ls_journal.set(iter,
            COLUMN_MIME, mime)
        self.ls_journal.set(iter,
            COLUMN_JOBJECT, new_jobject)

    def datastore_updated_cb(self,  uid):
        new_jobject = datastore.get(uid)
        iter = self.ls_journal.get_iter_first()
        for row in self.ls_journal:
            jobject = row[COLUMN_JOBJECT]
            if jobject.object_id == uid:
                title = new_jobject.metadata['title']
```

```
                self.ls_journal.set_value(iter,
                    COLUMN_TITLE, title)
                break
            iter = self.ls_journal.iter_next(iter)
        object_id = \
            self.selected_journal_entry.object_id
        if object_id == uid:
            self.set_form_fields(new_jobject)

    def datastore_deleted_cb(self,  uid):
        save_path = self.selected_path
        iter = self.ls_journal.get_iter_first()
        for row in self.ls_journal:
            jobject = row[COLUMN_JOBJECT]
            if jobject.object_id == uid:
                self.ls_journal.remove(iter)
                break
            iter = self.ls_journal.iter_next(iter)

        try:
            self.selection_journal.select_path(
                save_path)
            self.tv_journal.grab_focus()
        except:
            self.title_entry.set_text('')
            description_textbuffer = \
                self.description_textview.get_buffer()
            description_textbuffer.set_text('')
            tags_textbuffer = \
                self.tags_textview.get_buffer()
            tags_textbuffer.set_text('')
            self.btn_save.props.sensitive = False
            self.btn_delete.props.sensitive = False
            self.image.clear()
            self.image.show()
```

The **uid** passed to each callback method is the **object id** of the Journal object that has been added, updated, or deleted. If an entry is added to the Journal I get the Journal object from the datastore by its uid, then add it to the gtk.ListStore for the gtk.TreeModel I'm using to list out Journal entries. If an entry is updated or deleted I need to account for the possibility that the Journal entry I am viewing or editing may have been updated or removed. I use the uid to figure out which row in the gtk.ListStore needs to be removed or modified by looping through the entries in the gtk.ListStore looking for a match.

Now you know everything you'll ever need to know to work with the Journal.

18. MAKING ACTIVITIES USING PYGAME

INTRODUCTION

PyGame and **PyGTK** are two different ways to make a Python program with a graphical user interface. Normally you would not use both in the same program. Each of them has its own way of creating a window and each has its own way of handling events.

The base class Activity we have been using is an extension of the PyGTK Window class and uses PyGTK event handling. The toolbars all Activities use are PyGTK components. In short, any Activity written in Python must use PyGTK. Putting a PyGame program in the middle of a PyGTK program is a bit like putting a model ship in a bottle. Fortunately there is some Python code called **SugarGame** that will make it possible to do that.

Before we figure out how we'll get it in the bottle, let's have a look at our ship.

MAKING A STANDALONE GAME USING PYGAME

As you might expect, it's a good idea to make a standalone Python game using PyGame before you make an Activity out of it. I am not an experienced PyGame developer, but using the tutorial *Rapid Game Development with Python* by Richard Jones at this URL:

http://richard.cgpublisher.com/product/pub.84/prod.11

I was able to put together a modest game in about a day. It would have been sooner but the tutorial examples had bugs in them and I had to spend a fair amount of time using **The GIMP** to create image files for the sprites in the game.

Sprites are small images, often animated, that represent objects in a game. They generally have a transparent background so they can be drawn on top of a background image. I used the **PNG** format for my sprite files because it supports having an **alpha channel** (another term that indicates that part of the image is transparent).

PyGame has code to display background images, to create sprites and move them around on the background, and to detect when sprites collide with one another and do something when that happens. This is the basis for making a lot of 2D games. There are lots of games written with PyGame that could be easily adapted to be Sugar Activities.

My game is similar to the car game in the tutorial, but instead of a car I have an airplane. The airplane is the *Demoiselle* created by Alberto Santos-Dumont in 1909. Instead of having "pads" to collide with I have four students of Otto Lilienthal hovering motionless in their hang gliders. The hang gliders pitch downwards when Santos-Dumont collides with them. The controls used for the game have been modified too. I use the Plus and Minus keys on both the main keyboard and the keypad, plus the keypad 9 and 3 keys, to open and close the throttle and the Up and Down arrows on both the main keyboard and the keypad to move the joystick forward and back. Using the keypad keys is useful because the arrow keys on the keypad map to the game controller on the XO laptop, and the non-arrow keys on the keypad map to the other buttons on the XO laptop screen. These buttons can be used to play the game when the XO is in tablet mode.

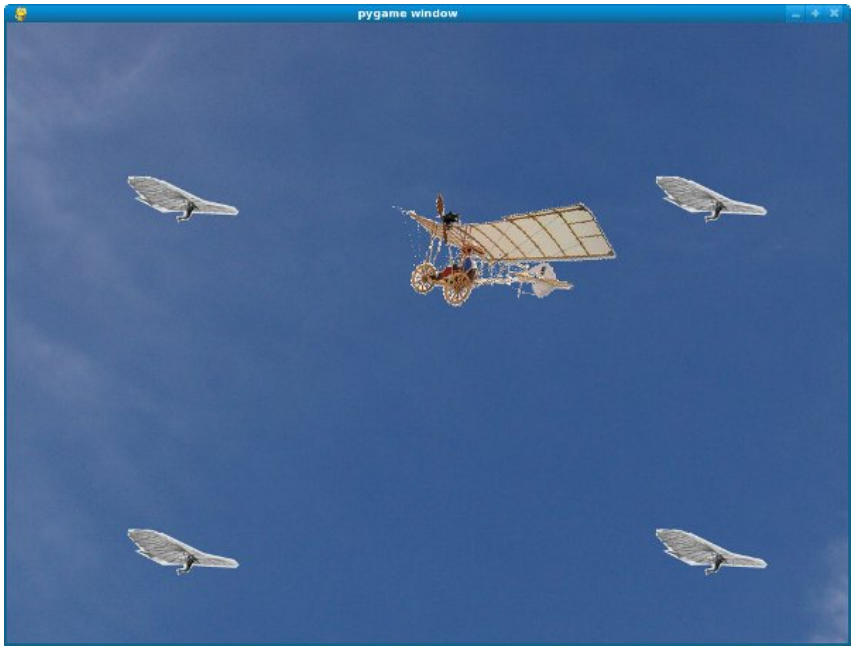
As a flight simulator it isn't much, but it does demonstrate at least some of the things PyGame can do. Here is the code for the game, which I'm calling **Demoiselle**:

```
#! /usr/bin/env python
import pygame
import math
import sys

class Demoiselle:
    "This is a simple demonstration of using PyGame \
    sprites and collision detection."
    def __init__(self):
        self.background = pygame.image.load('sky.jpg')
        self.screen = pygame.display.get_surface()
        self.screen.blit(self.background, (0, 0))
        self.clock = pygame.time.Clock()
        self.running = True

        gliders = [
            GliderSprite((200, 200)),
            GliderSprite((800, 200)),
            GliderSprite((200, 600)),
            GliderSprite((800, 600)),
        ]
        self. glider_group = pygame.sprite.RenderPlain(
            gliders)

    def run(self):
        "This method processes PyGame messages"
        rect = self.screen.get_rect()
        airplane = AirplaneSprite('demoiselle.png',
            rect.center)
        airplane_sprite = pygame.sprite.RenderPlain(
            airplane)

        while self.running:
            self.clock.tick(30)

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    return
                elif event.type == pygame.VIDEORESIZE:
                    pygame.display.set_mode(event.size,
                        pygame.RESIZABLE)
                    self.screen.blit(self.background,
                        (0, 0))

                if not hasattr(event, 'key'):
                    continue
                down = event.type == pygame.KEYDOWN
                if event.key == pygame.K_DOWN or \
                    event.key == pygame.K_KP2:
                    airplane.joystick_back = down * 5
                elif event.key == pygame.K_UP or \
                    event.key == pygame.K_KP8:
                    airplane.joystick_forward = down * -5
                elif event.key == pygame.K_EQUALS or \
                    event.key == pygame.K_KP_PLUS or \
                    event.key == pygame.K_KP9:
                    airplane.throttle_up = down * 2
                elif event.key == pygame.K_MINUS or \
                    event.key == pygame.K_KP_MINUS or \
```

```
                    event.key == pygame.K_KP3:
                    airplane.throttle_down = down * -2

            self.glider_group.clear(self.screen,
                self.background)
            airplane_sprite.clear(self.screen,
                self.background)
            collisions = pygame.sprite.spritecollide(
                airplane,
                self.glider_group,  False)
            self.glider_group.update(collisions)
            self.glider_group.draw(self.screen)
            airplane_sprite.update()
            airplane_sprite.draw(self.screen)
            pygame.display.flip()

class AirplaneSprite(pygame.sprite.Sprite):
    "This class represents an airplane, the Demoiselle \
    created by Alberto Santos-Dumont"
    MAX_FORWARD_SPEED = 10
    MIN_FORWARD_SPEED = 1
    ACCELERATION = 2
    TURN_SPEED = 5
    def __init__(self, image, position):
        pygame.sprite.Sprite.__init__(self)
        self.src_image = pygame.image.load(image)
        self.rect = pygame.Rect(
            self.src_image.get_rect())
        self.position = position
        self.rect.center = self.position
        self.speed = 1
        self.direction = 0
        self.joystick_back = self.joystick_forward = \
            self.throttle_down = self.throttle_up = 0

    def update(self):
        "This method redraws the airplane in response\
        to events."
        self.speed += (self.throttle_up +
            self.throttle_down)
        if self.speed > self.MAX_FORWARD_SPEED:
            self.speed = self.MAX_FORWARD_SPEED
        if self.speed < self.MIN_FORWARD_SPEED:
            self.speed = self.MIN_FORWARD_SPEED
        self.direction += (self.joystick_forward + \
            self.joystick_back)
        x_coord, y_coord = self.position
        rad = self.direction * math.pi / 180
        x_coord += -self.speed * math.cos(rad)
        y_coord += -self.speed * math.sin(rad)
        screen = pygame.display.get_surface()
        if y_coord < 0:
            y_coord = screen.get_height()

        if x_coord < 0:
            x_coord = screen.get_width()

        if x_coord > screen.get_width():
            x_coord = 0

        if y_coord > screen.get_height():
            y_coord = 0
        self.position = (x_coord, y_coord)
        self.image = pygame.transform.rotate(
            self.src_image, -self.direction)
        self.rect = self.image.get_rect()
        self.rect.center = self.position

class GliderSprite(pygame.sprite.Sprite):
    "This class represents an individual hang \
    glider as developed by Otto Lilienthal."
    def __init__(self, position):
        pygame.sprite.Sprite.__init__(self)
        self.normal = pygame.image.load(
             'glider_normal.png')
        self.rect = pygame.Rect(self.normal.get_rect())
        self.rect.center = position
        self.image = self.normal
        self.hit = pygame.image.load('glider_hit.png')
    def update(self, hit_list):
        "This method redraws the glider when it collides\
        with the airplane and when it is no longer \
        colliding with the airplane."
        if self in hit_list:
            self.image = self.hit
        else:
            self.image = self.normal
```

```
def main():
    "This function is called when the game is run \
    from the command line"
    pygame.init()
    pygame.display.set_mode((0, 0), pygame.RESIZABLE)
    game = Demoiselle()
    game.run()
    sys.exit(0)

if __name__ == '__main__':
    main()
```

And here is the game in action:



You'll find the code for this game in the file **demoiselle.py** in the book examples project in Git.

INTRODUCING SUGARGAME

SugarGame is not part of Sugar proper and probably never will be. If you want to use it you'll need to include the Python code for SugarGame inside your Activity bundle. I've included the version of SugarGame I'm using in the book examples project in the **sugargame** directory, but when you make your own games you'll want to be sure and get the latest code to include. You can do that by downloading the project from Gitorious using these commands:

```
mkdir sugargame
cd sugargame
git clone git://git.sugarlabs.org/sugargame/mainline.git
```

You'll see two subdirectories in this project: **sugargame** and **test**, plus a **README.txt** file that contains information on using sugargame in your own Activities. The test directory contains a simple PyGame program that can be run either standalone or as an Activity. The standalone program is in the file named **TestGame.py**. The Activity, which is a sort of wrapper around the standalone version, is in file **TestActivity.py**.

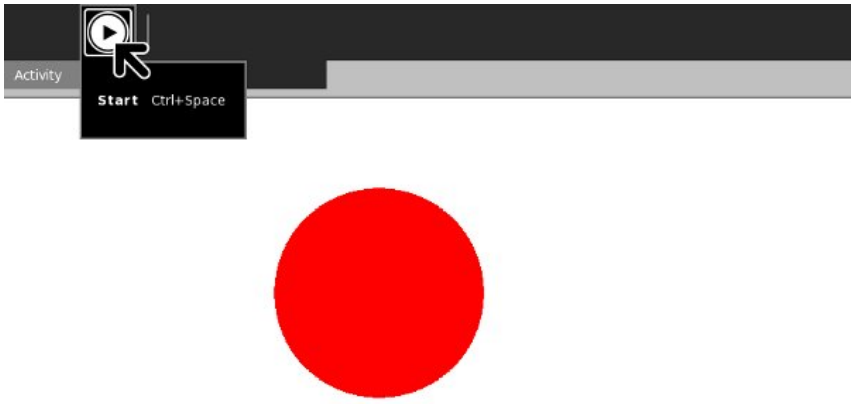
If you run **TestGame.py** from the command line you'll see it displays a bouncing ball on a white background. To try running the Activity version you'll need to run

```
./setup.py dev
```

from the command line first. I was not able to get the Activity to work under sugar-emulator until I made two changes to it:

- I made a copy of the **sugargame** directory within the **test** directory.
- I removed the line reading "**sys.path.append('..') # Import sugargame package from top directory.**" from **TestActivity.py**. Obviously this line is supposed to help the program find the **sugargame** directory in the project but it didn't work for me. Your own experience may be different.

The Activity looks like this:



The **PyGame** toolbar has a single button that lets you make the bouncing ball pause and resume bouncing.

MAKING A SUGAR ACTIVITY OUT OF A PYGAME PROGRAM

Now it's time to put our ship in that bottle. The first thing we need to do is make a copy of the **sugargame** directory of the SugarGame project into the mainline directory of our own project.

The **README.txt** file in the SugarGame project is worth reading. It tells us to make an Activity based on the **TestActivity.py** example in the SugarGame project. This will be our bottle. Here is the code for mine, which is named **DemoiselleActivity.py**:

```
# DemoiselleActivity.py

from gettext import gettext as _

from gi.repository import Gtk
import pygame
from sugar3.activity import activity
from sugar3.graphics.toolbutton import ToolButton
from sugar3.graphics.toolbarbox import ToolbarButton
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3.activity.widgets import StopButton
from sugar3.activity.widgets import ActivityToolbar
from gi.repository import GObject
import sugargame.canvas
```

```
import demoiselle2

class DemoiselleActivity(activity.Activity):
    def __init__(self, handle):
        super(DemoiselleActivity, self).__init__(handle)

        # Build the activity toolbar.
        self.build_toolbar()

        # Create the game instance.
        self.game = demoiselle2.Demoiselle()

        # Build the Pygame canvas.
        self._pygamecanvas = sugargame.canvas.PygameCanvas(self)
        # Note that set_canvas implicitly calls read_file when
        # resuming from the Journal.
        self.set_canvas(self._pygamecanvas)
        self._pygamecanvas.grab_focus()
        self.score = '0'

        # Start the game running.
        self._pygamecanvas.run_pygame(self.game.run)

    def build_toolbar(self):
        toolbar_box = ToolbarBox()

        view_toolbar = ViewToolbar()
        view_toolbar.connect('go-fullscreen',
                self.view_toolbar_go_fullscreen_cb)
        view_toolbar.show()
        view_toolbar_button = ToolbarButton(
            page=view_toolbar,
            icon_name='toolbar-view')
        toolbar_box.toolbar.insert(view_toolbar_button, -1)
        view_toolbar_button.show()

        separator = Gtk.SeparatorToolItem()
        separator.props.draw = False
        separator.set_expand(True)
        toolbar_box.toolbar.insert(separator, -1)
        separator.show()

        stop_button = StopButton(self)
        stop_button.props.accelerator = '<Ctrl><Shift>Q'
        toolbar_box.toolbar.insert(stop_button, -1)
        stop_button.show()

        self.set_toolbar_box(toolbar_box)
        toolbar_box.show()

    def view_toolbar_go_fullscreen_cb(self, view_toolbar):
        self.fullscreen()

    def read_file(self, file_path):
        score_file = open(file_path, "r")
        while score_file:
            self.score = score_file.readline()
            self.game.set_score(int(self.score))
        score_file.close()

    def write_file(self, file_path):
        score = self.game.get_score()
        f = open(file_path, 'wb')
        try:
            f.write(str(score))
        finally:
            f.close

class ViewToolbar(Gtk.Toolbar):
    __gtype_name__ = 'ViewToolbar'

    __gsignals__ = {
        'needs-update-size': (GObject.SIGNAL_RUN_FIRST,
                              GObject.TYPE_NONE,
                              ([])),
        'go-fullscreen': (GObject.SIGNAL_RUN_FIRST,
                          GObject.TYPE_NONE,
                          ([]))
    }

    def __init__(self):
        Gtk.Toolbar.__init__(self)
        self.fullscreen = ToolButton('view-fullscreen')
        self.fullscreen.set_tooltip(_('Fullscreen'))
        self.fullscreen.connect('clicked', self.fullscreen_cb)
        self.insert(self.fullscreen, -1)
        self.fullscreen.show()
```

```
    def fullscreen_cb(self, button):
        self.emit('go-fullscreen')
```

This is a bit fancier than **TestActivity.py**. I decided that my game didn't really need to be paused and resumed, so I replaced the **PyGame** toolbar with a **View** toolbar that lets the user hide the toolbar when it is not needed. I use the *read_file()* and *write_file()* methods to save and restore the game score. (Actually this is faked, because I never put in any scoring logic in the game). I also hide the **Share** control in the main toolbar.

As you would expect, getting a ship in a bottle does require the ship to be modified. Here is **demoiselle2.py**, which has the modifications:

```
#! /usr/bin/env python
#
# demoiselle2.py

import pygame
from gi.repository import Gtk
import math
import sys

class Demoiselle:
    "This is a simple demonstration of using PyGame \
    sprites and collision detection."
    def __init__(self):
        self.clock = pygame.time.Clock()
        self.running = True
        self.background = pygame.image.load('sky.jpg')
        self.score = 99

    def get_score(self):
        return self.score

    def set_score(self,  score):
        self.score = score

    def run(self):
        "This method processes PyGame messages"

        screen = pygame.display.get_surface()
        screen.blit(self.background, (0, 0))

        gliders = [
            GliderSprite((200, 200)),
            GliderSprite((800, 200)),
            GliderSprite((200, 600)),
            GliderSprite((800, 600)),
        ]
        glider_group = pygame.sprite.RenderPlain(gliders)

        rect = screen.get_rect()
        airplane = AirplaneSprite('demoiselle.png', rect.center)
        airplane_sprite = pygame.sprite.RenderPlain(airplane)

        while self.running:
            self.clock.tick(30)

            # Pump GTK messages.
            while Gtk.events_pending():
                Gtk.main_iteration()

            # Pump PyGame messages.
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    return
                elif event.type == pygame.VIDEORESIZE:
                    pygame.display.set_mode(event.size,
pygame.RESIZABLE)
                    screen.blit(self.background, (0, 0))

                if not hasattr(event, 'key'):
                    continue
                down = event.type == pygame.KEYDOWN
                if event.key == pygame.K_DOWN or \
                    event.key == pygame.K_KP2:
                    airplane.joystick_back = down * 5
                elif event.key == pygame.K_UP or \
                    event.key == pygame.K_KP8:
                    airplane.joystick_forward = down * -5
                elif event.key == pygame.K_EQUALS or \
                    event.key == pygame.K_KP_PLUS or \
```

```
                    event.key == pygame.K_KP9:
                    airplane.throttle_up = down * 2
                elif event.key == pygame.K_MINUS or \
                    event.key == pygame.K_KP_MINUS or \
                    event.key == pygame.K_KP3:
                    airplane.throttle_down = down * -2

            glider_group.clear(screen, self.background)
            airplane_sprite.clear(screen, self.background)
            collisions = pygame.sprite.spritecollide(airplane, \
                                                     glider_group,
False)
            glider_group.update(collisions)
            glider_group.draw(screen)
            airplane_sprite.update()
            airplane_sprite.draw(screen)
            pygame.display.flip()

class AirplaneSprite(pygame.sprite.Sprite):
    "This class represents an airplane, the Demoiselle \
    created by Alberto Santos-Dumont"
    MAX_FORWARD_SPEED = 10
    MIN_FORWARD_SPEED = 1
    ACCELERATION = 2
    TURN_SPEED = 5
    def __init__(self, image, position):
        pygame.sprite.Sprite.__init__(self)
        self.src_image = pygame.image.load(image)
        self.rect = pygame.Rect(self.src_image.get_rect())
        self.position = position
        self.rect.center = self.position
        self.speed = 1
        self.direction = 0
        self.joystick_back = self.joystick_forward = \
            self.throttle_down = self.throttle_up = 0

    def update(self):
        "This method redraws the airplane in response\
        to events."
        self.speed += (self.throttle_up + self.throttle_down)
        if self.speed > self.MAX_FORWARD_SPEED:
            self.speed = self.MAX_FORWARD_SPEED
        if self.speed < self.MIN_FORWARD_SPEED:
            self.speed = self.MIN_FORWARD_SPEED
        self.direction += (self.joystick_forward + self.joystick_back)
        x_coord, y_coord = self.position
        rad = self.direction * math.pi / 180
        x_coord += -self.speed * math.cos(rad)
        y_coord += -self.speed * math.sin(rad)
        screen = pygame.display.get_surface()
        if y_coord < 0:
            y_coord = screen.get_height()

        if x_coord < 0:
            x_coord = screen.get_width()

        if x_coord > screen.get_width():
            x_coord = 0

        if y_coord > screen.get_height():
            y_coord = 0
        self.position = (x_coord, y_coord)
        self.image = pygame.transform.rotate(self.src_image, -
self.direction)
        self.rect = self.image.get_rect()
        self.rect.center = self.position

class GliderSprite(pygame.sprite.Sprite):
    "This class represents an individual hang glider as developed\
    by Otto Lilienthal."
    def __init__(self, position):
        pygame.sprite.Sprite.__init__(self)
        self.normal = pygame.image.load('glider_normal.png')
        self.rect = pygame.Rect(self.normal.get_rect())
        self.rect.center = position
        self.image = self.normal
        self.hit = pygame.image.load('glider_hit.png')
    def update(self, hit_list):
        "This method redraws the glider when it collides\
        with the airplane and when it is no longer \
        colliding with the airplane."
        if self in hit_list:
            self.image = self.hit
        else:
            self.image = self.normal

def main():
    "This function is called when the game is run from the command
```

```
line"
    pygame.init()
    pygame.display.set_mode((0, 0), pygame.RESIZABLE)
    game = Demoiselle()
    game.run()
    sys.exit(0)

if __name__ == '__main__':
    main()
```

Why not load both **demoiselle.py** and **demoiselle2.py** in Eric and take a few minutes to see if you can figure out what changed between the two versions?

Surprisingly little is different. I added some code to the PyGame main loop to check for PyGTK events and deal with them:

```
        while self.running:
            self.clock.tick(30)

              # Pump GTK messages.
              while Gtk.events_pending():
                  Gtk.main_iteration()

            # Pump PyGame messages.
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    return
                elif event.type == pygame.VIDEORESIZE:
                    pygame.display.set_mode(event.size,
pygame.RESIZABLE)
                    screen.blit(self.background, (0, 0))

                if not hasattr(event, 'key'):
                    continue
                down = event.type == pygame.KEYDOWN
                if event.key == pygame.K_DOWN or \

... continue dealing with PyGame events ...
```

This has the effect of making PyGame and PyGTK take turns handling events. If this code was not present GTK events would be ignored and you'd have no way to close the Activity, hide the toolbar, etc. You need to add

```
from gi.repository import Gtk
```

at the top of the file so these methods can be found.

Of course I also added the methods to set and return scores:

```
     def get_score(self):
        return self.score

     def set_score(self, score):
        self.score = score
```

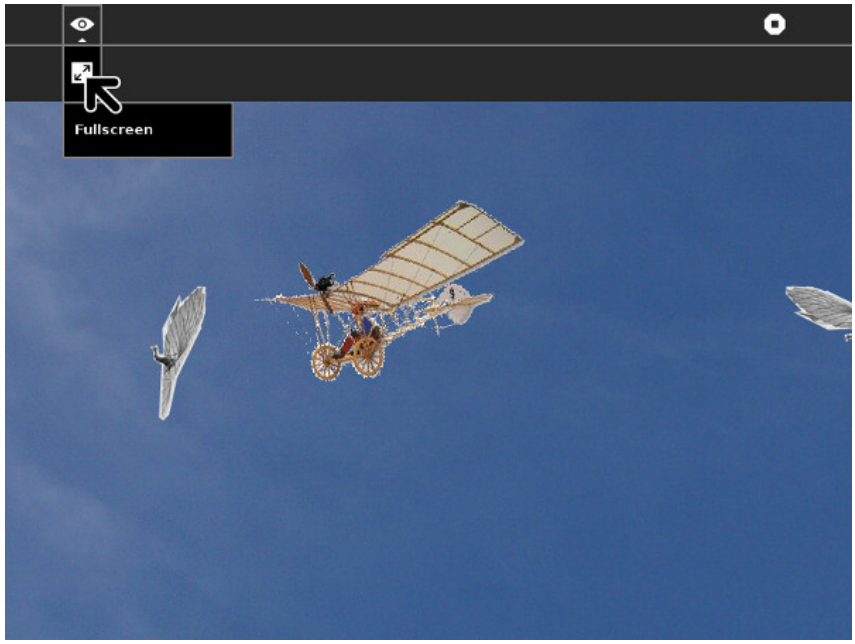
The biggest change is in the *__init__()* method of the **Demoiselle** class. Originally I had code to display the background image on the screen:

```
    def __init__(self):
        self.background = pygame.image.load('sky.jpg')
        self.screen = pygame.display.get_surface()
        self.screen.blit(self.background, (0, 0))
```

The problem with this is that sugargame is going to create a special PyGTK Canvas object to replace the PyGame display and the DemoiselleActivity code hasn't done that yet, so **self.screen** will have a value of None. The only way to get around that is to move any code that refers to the **display** out of the *__init__()* method of the class and into the beginning of the method that contains the event loop. This may leave you with an *__init__()* method that does little or nothing. About the only thing you'll want there is code to create instance variables.

Nothing we have done to **demoiselle2.py** will prevent it from being run as a standalone Python program.

To try out the game run **./setup.py dev** from within the **Making_Activities_Using_PyGame_gtk3** directory. When you try out the Activity it should look like this:



PORTING ACTIVITIES FROM OLPCGAMES TO SUGARGAME

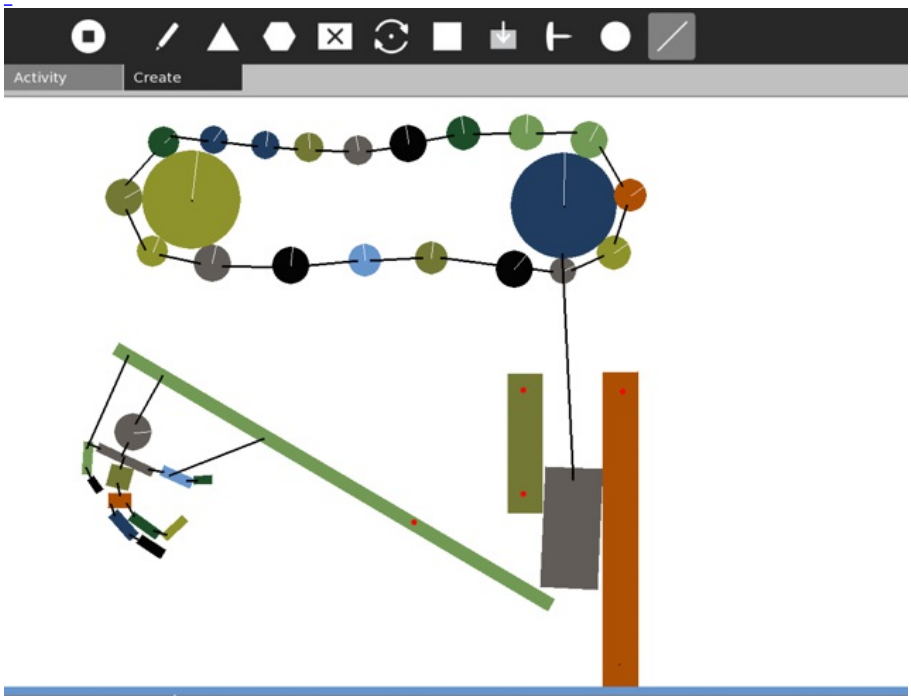
Olpcgames is a deprecated tool to make a Sugar activity using Pygame. It has been replaced by Sugargame. The main difference between Olpcgames and Sugargame is that Olpcgames provides a framework to develop Activities with toolbars in Gtk and the canvas in Pygame. Sugargame gives the possibility to embed Pygame into your Gtk window. With Sugargames you can use the Pygame canvas as another Gtk widget and combine it with other gtkwidgets in the same canvas.

Olpcgames was not ported to Gtk3, so activities using Olpcgames will need to be ported to Sugargame.

Porting the Physics Activity

To see the entire example, visit:

http://git.sugarlabs.org/~danielf/physics/sugargame



First steps

To move an activity from Olpcgames to Sugargame, the first is replace the olpcgames directory with the sugargame directory.

Modifying activity.py

Imports to remove:

```
import olpcgames
```

Imports to add

```
import sugargame
import sugargame.canvas
```

We also need to import the game code:

```
import physics
```

The main change in this file in regard to the activity class. Olpcgames has its own activities class, but with Sugargame, we must inherit the activity class from activity.Activity. We also remove some class attributes that are no longer needed.

This is the activity declaration using Olpcgames:

```
class PhysicsActivity(olpcgames.PyGameActivity):
    game_name = 'physics'
    game_title = _('Physics')
    game_size = None
 # Olpcgame will choose size
```

Using Sugargame, the code is simplier.

```
 class PhysicsActivity(activity.Activity):
```

The __init__ method will need to add the following new lines:

```
        # Build the Pygame canvas.
        self._canvas = sugargame.canvas.PygameCanvas(self)
        self.game = physics.main(self)
        self.build_toolbar()
        self.set_canvas(self._canvas)
```

```
        # Start the game running.
        self._canvas.run_pygame(self.game.run)
```

In Olpcgames it is normal to communicate between the main activity and pygame by sending Pygame events. In Sugargame it is not necessary, but if you don't want to rewrite your activity, you can still do it by sending pygame events instead of the custom olpcgames events. The following expression:

```
    pygame.event.post(olpcgames.eventwrap.Event(pygame.USEREVENT,
                                    action="stop_start_toggle"))
```

Can be replaced with this other:

```
    pygame.event.post(pygame.event.Event(pygame.USEREVENT,
                                    action="stop_start_toggle"))
```

The game code in physics.py

Imports: Replace:

```
import olpcgames
```

With:

```
import sugargame
```

Game class

Olpcgames gives the screen to the gameclass and sugargame not. The screen must be obtained in the run method. A typical game class in Olpcgames, starts with:

```
 class PhysicsGame:
     def __init__(self, screen):
         self.screen = screen
         self.canvas = olpcgames.ACTIVITY.canvas
         self.clock = pygame.time.Clock()
...
```

In Sugargame, it doesn't but it can need to access the main activity:

```
    def __init__(self, activity):
         # Get everything set up
         self.canvas = activity.canvas
         self.clock = pygame.time.Clock()
...
```

If the code in __init__ needs to use the screen, you must move it to the run method and of course, get the screen by yourself.

```
    def run(self):
        self.screen = pygame.display.get_surface()
```

Another change must be in the constructor:

```
def main():
    toolbarheight = 75
    tabheight = 45
    pygame.display.init()
    video_info = pygame.display.Info()
    x = video_info.current_w
    y = video_info.current_h
    screen = pygame.display.set_mode((x, y - toolbarheight -
tabheight))
    game = PhysicsGame(screen)
    game.run()
```

A constructor for a Sugargame, can be like this one if you want to. Anyway, you can make the class directly without using a function.

```
def main(activity):
    game = PhysicsGame(activity)
    return game
```

Journal files:

The way to open and save data in Journal is different in olpcgames

and sugargame. The first step to port is add the write_file and read_file methods to the activity.

```
    def read_file(self, file_path):
        self.game.read_file(file_path)

    def write_file(self, file_path):
        self.game.write_file(file_path)
```

Then you must change the game class, in Olpcgames there is an event checker in the run loop that you must move to methods called write_file and read_file.

```
    def write_file(self, path):
        #Saving to journal
        self.world.add.remove_mouseJoint()
        self.world.json_save(path)

    def read_file(self, path):
        #Loading from journal
        self.opening_queue = path
```

The old code, called from the run loop was:

```
            elif hasattr(event, "code"):
                if event.code == olpcgames.FILE_WRITE_REQUEST:
                    #Saving to journal
                    self.game.world.add.remove_mouseJoint()
                    self.game.world.json_save(event.filename)
                elif event.code == olpcgames.FILE_READ_REQUEST:
                    #Loading from journal
                    self.game.world.json_load(event.filename)
```

Note that the new read_file method is called before Pygame is started, so it only sets the file path to a opening queue. But I had to modify other things to get it working.

```
class PhysicsGame:
     def __init__(self, activity):
...
         self.opening_queue = None

     def run(self):
...
            if self.opening_queue:
                self.world.json_load(self.opening_queue)
```

GUEST CHAPTERS

19. DEVELOPING SUGAR ACTIVITIES USING HTML5 AND WEBKIT

by Lionel Laské

Sugar Activities, like Sugar itself, are usually developed in Python. Python is a very nice dynamic language, with a clear and readable syntax that is both powerful and simple to learn. In addition to that, Python is an interpreted language that allows Sugar Activities to run unmodified on multiple platforms (from XO-1/XO 1.5 or a standard PC based on x86 architecture to XO-1.75/XO-4 based on ARM architecture).

But Python is not the only language that has these advantages. Recently HTML5 and JavaScript have become worthy alternatives to Python. In this chapter we'll explore how to develop a Sugar activity using mostly HTML5/JavaScript, with just a little Python wrapped around it.

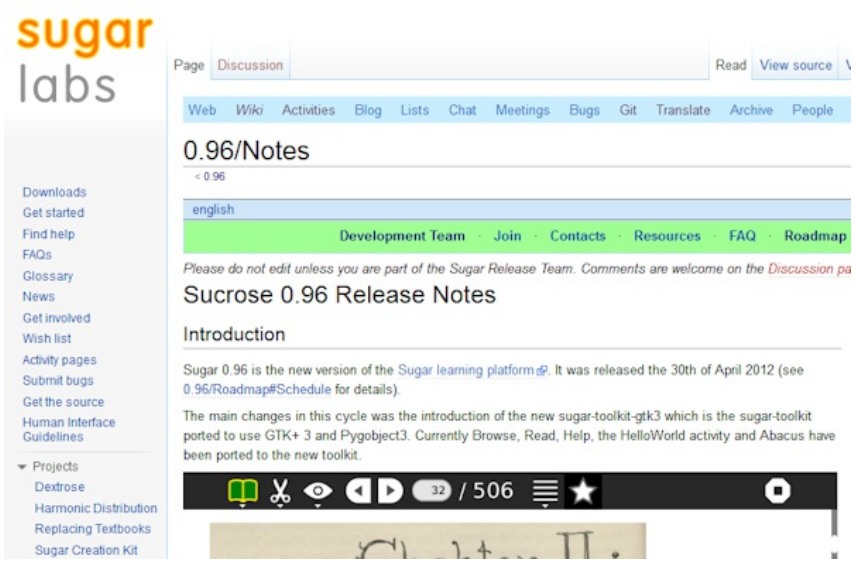
WHAT YOU NEED

Using HTML in a Sugar Activity is not new. The Wikipedia Activity has been embedding HTML contents for years. It does this by integrating into the Activity two things: the Sugar browser to render HTML and an HTTP server to react to user clicks by calling Python code.

However this architecture is not fully satisfying. First because it's relatively complex: three paradigms are in the same application (client code, embedded HTML and HTTP server code). Second because the initial version of Sugar browser was based on an old version of Gecko (the HTML engine from Firefox), so the HTML rendering had very limited capacity.

The solution described below uses an alternate way that allow to write Sugar Activities in HTML5, JavaScript, and Python. You'll get excellent HTML rendering and you'll be able to call Python code from JavaScript so your Activity can do anything an all-Python Activity can do, and you won't need to put an HTTP server inside your Activity to make that happen. You'll need:

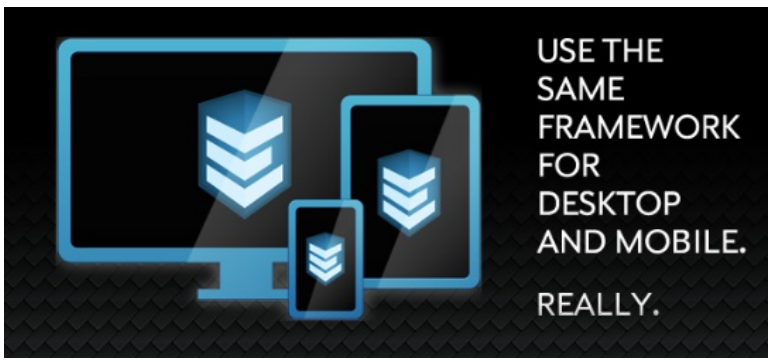
- Sugar 0.96 or more,
- The Enyo JavaScript Framework,
- Some nice contents.



The first thing to do is to use a recent version of Sugar: Sugar 0.96 or later. Sugar 0.96 is now officially available as a signed release for XO 1, XO-1.5 and XO 1.75. Sugar 0.96 came with two very important features:

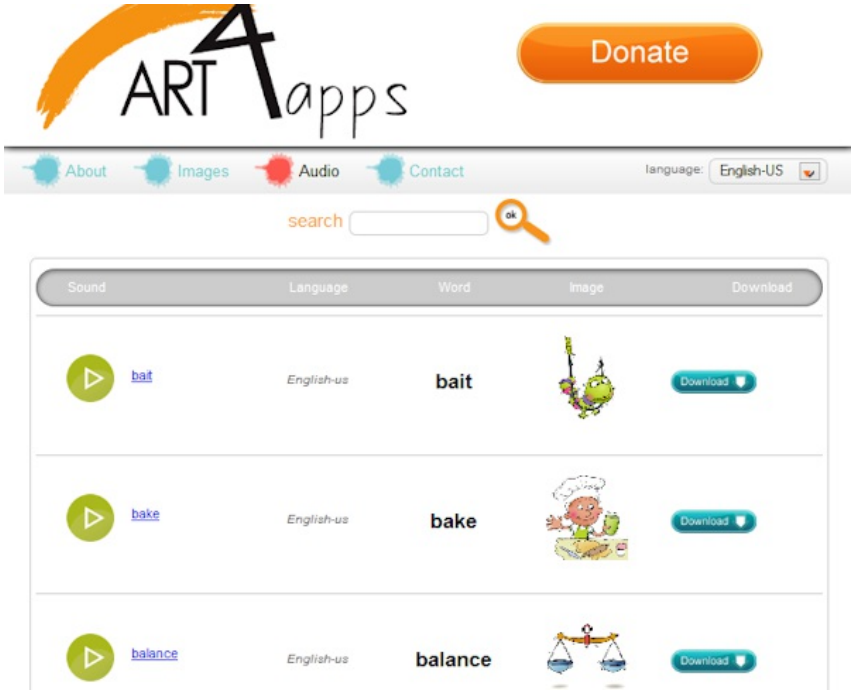
- Porting of Sugar to Gtk3 that allow to Sugar to have a powerful and up-to-date graphic framework. You will learn more on Gtk3 porting it in the next chapter.
- Using of WebKit as HTML rendering engine that allow a high-level support of HTML5 features (WebKit is the rendering engine for Chrome and Safari browsers).

The second thing to do is to use the Enyo JavaScript Framework (http://enyojs.com/). This is one of the many Open Source JavaScript frameworks on the market today. Enyo is very simple, elegant, component-oriented and, portable. "Portable" means that an application developed with Enyo could work easily on lot of different devices (smartphones, tablets, ...). So a developer could write an application not only for Sugar but at the same time for other systems. This might be a consideration for a developer who would otherwise view Sugar as a "limited market".



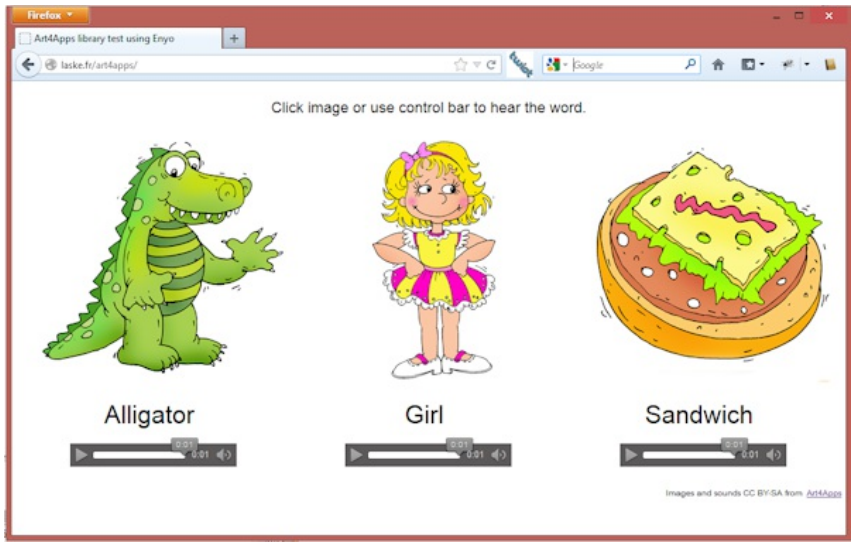
Finally, the last thing to think about when writing an HTML activity is content. When writing an Activity, coding is not all, content is important too, especially when you've got opportunity to write HTML contents that play sounds or display nice images. Because developers are not always artistic, it will be useful to have an existing library of images and sounds that you can use in your Activity.

ILearn4Free is an editor of iStory, which are nice interactive stories for the iPad. ILearn4Free provides a large database of professionally recorded audio and images on a dedicated website called Art4apps (http://www.art4apps.org/). All the contents can be distributed freely (under CC BY-SA). It's a good start to conceive new HTML Activities.



YOUR FIRST HTML5 ACTIVITY

We've got now all the pieces of the puzzle, so let's start writing our first HTML Activity. We'll start with a very simple HTML page with images and sounds coming from the Art4Apps library. The capture below show you the page in the browser. You could play with the page here: http://laske.fr/art4apps/



When you click on an image, you could hear the pronunciation of the word.

Here is the HTML5 code for this page: "index.html".

```
<!doctype html>
<html>
<head>
  <title>Art4Apps library test using Enyo</title>
  <link href="enyo/enyo.css" rel="stylesheet"
        type="text/css" />
  <script src="enyo/enyo.js" type="text/javascript"/>
  <link href="styles.css" rel="stylesheet"
        type="text/css" />
  <script src="package.js" type="text/javascript"/>
</head>
<body>
<script type="text/javascript">
    new TestArt4Apps().renderInto(document.body);
</script>
</body>
</html>
```

As you could see, it's almost empty because with Enyo, all the design of your page is built into JavaScript classes. In the HTML file you've got only the link to the Enyo framework ("enyo.css" and "enyo.js"), the style sheet for your content ("styles.css") and a "package.js" file that you use to reference all JavaScript files in your project.

Here is the "package.js" file content:

```
enyo.depends(
  "audio.js",
  "app.js"
);
```

"enyo.depends" is an Enyo framework method that allow you to list all JavaScript files that you need. We're going to ignore "audio.js" that contain only HTML5 audio stuff. Let's see the source code for the "app.js" file, our main file:

```
enyo.kind({
  name: "TestArt4Apps",
  kind: enyo.Control,
  components: [
    { components: [
      { content:
         "Click image or use control bar to hear the word.",
         classes: "title" },
        { kind: "Item.Element", text: "Alligator",
          image: "images/alligator.png",
          sound: ["audio/alligator.ogg", "audio/alligator.mp3"],
          classes: "item" },
        { kind: "Item.Element", text: "Girl",
          image: "images/girl.png",
          sound: ["audio/girl.ogg", "audio/girl.mp3"],
          classes: "item" },
        { kind: "Item.Element", text: "Sandwich",
          image: "images/sandwich.png",
          sound: ["audio/sandwich.ogg", "audio/sandwich.mp3"],
          classes: "item" },
          { classes: "footer", components: [
            { content: "Images and sounds CC BY-SA from",
              classes: "licence" },
            { tag: "a",
              attributes: {"href":"http://art4apps.org/"},
              content: "Art4Apps", classes: "licence" }
          ]}
        ]}
      ],

      // Constructor
      create: function() {
        this.inherited(arguments);
      }
});
```

This source code creates a new JavaScript Enyo class named "TestArt4Apps" with three "Item.Element" and some simple text contents. I'm sure you could appreciate here the simplicity of the Enyo framework: you just need to compound components of the page in JavaScript objects and arrays. The "TestArt4app" object is created by the JavaScript contents in the HTML page (see before) and rendered as the body of the HTML document using the "renderInto" Enyo method.

The "Item.Element" class mentioned in the "TestArt4apps" class is another Enyo component that consolidates image, sound and text. Here is the source code:

```
enyo.kind({
  name: "Item.Element",
  kind: enyo.Control,
  published: { image: "", sound: "", text: "" },
  ontap: "taped",
  components: [
    { name: "itemImage", classes: "itemImage", kind: "Image",
      ontap: "taped" },
    { name: "itemText", classes: "itemText", ontap: "taped" },
    { name: "itemSound", classes: "itemSound",
      kind: "HTML5.Audio", preload: "auto", autobuffer: true,
      controlsbar: true }
  ],

  // Constructor
  create: function() {
    this.inherited(arguments);
    this.imageChanged();
    this.soundChanged();
    this.textChanged();
  },

  // Image setup
  imageChanged: function() {
    if (this.image.length != 0) {
      this.$.itemImage.setAttribute("src", this.image);
      this.$.itemImage.show();
    } else {
      this.$.itemImage.hide();
    }
  },

  // Sound setup
  soundChanged: function() {
    this.$.itemSound.setSrc(this.sound);
  },

  // Text setup
  textChanged: function() {
    if (this.text.length != 0) {
      this.$.itemText.setContent(this.text);
      this.$.itemText.show();
    } else {
      this.$.itemText.show();
    }
  },

  // Play sound when image taped
  taped: function() {
    if (this.$.itemSound.paused())
      this.$.itemSound.play();
    else
      this.$.itemSound.pause();
  }
});
```

Here you could see that Enyo allows you not only to declare new class but let you also declare properties (here "image", "sound" and "text" for the "Item.Element") and let you react to event (properties value changed or taping on a component).

You could be surprised to not seen any styles and formatting stuff in the HTML and JavaScript files. It's because all formatting is done in the CSS file. Here is the content of the CSS file:

```
.title {
 margin-bottom: 20px;
 margin-top: 20px;
```

```
  text-align: center;
  font-size: large;
}

.item {
  display: inline-block;
}

.itemImage {
  margin-left: 30px;
}

.itemText {
  font-size:30px;
  text-align: center;
  padding: 16px;
}

.itemSound {
  width: 200px;
  margin-left: 70px;
}

.footer {
  width: 100%;
  text-align: right;
}

.licence {
  display: inline-block;
  font-size: x-small;
  margin-right: 5px;
  margin-top: 30px;
}
```

The CSS file define classes that you associate to each JavaScript component using the "classes" attribute. For example, the line:

```
{ content:
  "Click image or use control bar to hear the word.",
  classes: "title" },
```

Tell to Enyo to display the text centered, using a large font with a margin top and bottom of 20 pixels.

Okay, we've got now a cool HTML5 contents but let's convert this into a Sugar Activity.

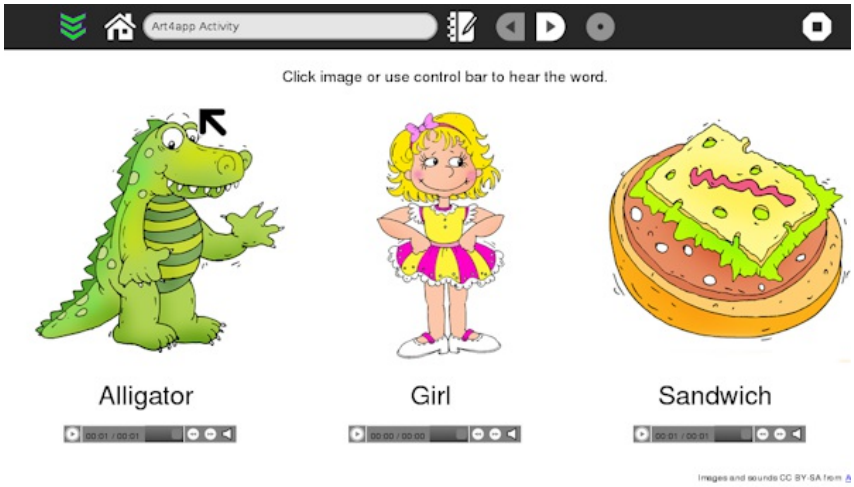
As you learn at Chapter 3, a Sugar Activity is a ".XO" file. A "XO file" is just a zipped file with Python code and initialization (setup and a manifest). To avoid complexity related to this file, I've prepared a template here http://git.sugarlabs.org/art4apps-Activity/master/trees/master. This template is a standard Activity but with a "WebView Gtk3 widget" that fills the main part of the screen. A WebView Gtk3 Widget is a widget that encapsulate the WebKit browser into a control that you could use like any other Gtk control. Here is the Python code to create it:

```
vbox = Gtk.VBox(True)
self.webview = webview = WebKit.WebView()
webview.show()
vbox.pack_start(webview, True, True, 0)
vbox.show()
```

At startup the Python code will open an HTML page into the WebView using the "load_uri" method. This HTML page could be an external page (pointing anywhere on the web) or a local one. My template uses an "html" local subdirectory inside the activity to store all HTML contents and the latest version of the Enyo Framework. So here how the initialization of the WebView is done for the activity :

```
web_app_page = os.path.join(activity.get_bundle_path(), \
                            "html/index.html")
self.webview.load_uri('file://' + web_app_page)
```

That's all ! You've got now a standard Sugar activity like any other but mainly in HTML5. The resulting XO activity is downloadable on http://laske.fr/art4apps/art4app-1.xo. Here is the result.



The complete source code can be found on http://git.sugarlabs.org/art4apps-activity/master/trees/master.

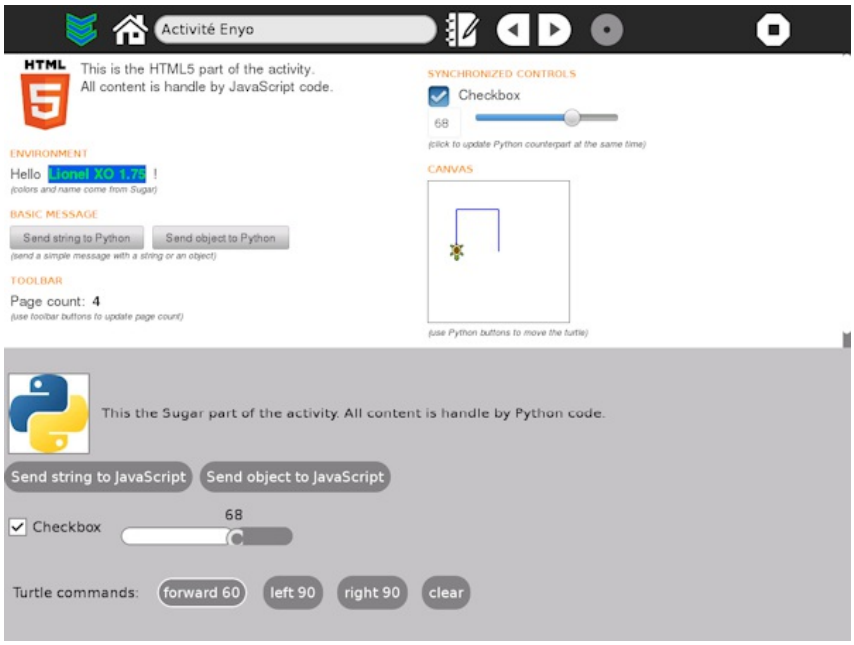
GOING FURTHER: INTEGRATE HTML5 CONTENT WITH SUGAR

Writing an Activity using HTML5 and JavaScript is nice but there are times when you will need to call the Sugar API. It's the case for example when you would interact with the Activity toolbar or when you need to store contents in the Sugar Datastore (i.e. Journal).

To avoid integration of an HTTP Server in the Activity, I choose to develop a small framework that allows bi-directional exchange between the Activity and the embedded HTML content. The framework is available both in Python and in JavaScript. So you could call Python code from your HTML5 content and conversely.

Let's see it in another example to understand how this framework works. You can download the activity example on http://olpc-france.org/download/enyo-1.xo and see the complete source code on http://git.sugarlabs.org/enyo-activity/.

This time the main screen of the Activity is split in three parts: the standard toolbar, the WebView with the HTML5/JavaScript content and a set of Python Gtk controls.



The sample show how JavaScript and Python code could be mixed together. Specifically features that we're going to demonstrate are:

- Toolbar buttons that call JavaScript code in the HTML5 page,
- JavaScript code that call Sugar API to get XO buddy colors and name,
- Synchronized Python and HTML controls (checkbox and progress bar),
- Sending of basic and complex data between Python and JavaScript and conversely.

All these stuff use the same feature, the capacity to send custom message between JavaScript and Python code. To do that, on the Python side, you first need to import the Enyo class from then enyo.py file:

```
from enyo import Enyo
```

Then you need to create a new instance of a class named "Enyo" with the WebView object that you created before, as parameter :

```
self.enyo = Enyo(webview)
```

This new object will give you access to the framework. Two methods of the Enyo object should be learn: "connect" and "send_message".

You could subscribe to JavaScript messages using the "connect" method. For example, we subscribe to the JavaScript "ready" message so the "init_context" Python method will be called when the HTML5 page will send this message.

```
self.enyo.connect("ready", self.init_context)
```

The source of then "init_context" method will give us opportunity, to discover the second method of the Enyo object, the "send_message" method:

```
def init_context(self, args):
  """Init Javascript context sending buddy information"""
  # Get XO colors
  buddy = {}
  client = gconf.client_get_default()
  colors = client.get_string("/desktop/sugar/user/color")
```

```
    buddy["colors"] = colors.split(",")

    # Get XO name
    presenceService = presenceservice.get_instance()
    buddy["name"] = presenceService.get_owner().props.nick

    self.enyo.send_message("buddy", buddy)
```

As you can see, the "init_context" method call the Sugar API (presence and gconf) to build a Python object with colors and name value of the buddy XO (the small icon in the center the Sugar home view). Using the "send_message" method, we'll send these values to the JavaScript code.

Another interesting "send_message" call is used in our sample to handle toolbar events. Here is a part of the Python code to create and handle events for Toolbar button Back and Forward (go to chapter 19 to learn more about Toolbar handling).

```
def make_toolbar(self):
    # toolbar with the new toolbar redesign
    toolbar_box = ToolbarBox()

    # ...

    back_button = ToolButton('go-previous-paired')
    back_button.set_tooltip('Page count -1')
    back_button.connect('clicked', self.go_back)
    toolbar_box.toolbar.insert(back_button, -1)
    back_button.show()

    forward_button = ToolButton('go-next-paired')
    forward_button.set_tooltip('Page count +1')
    forward_button.connect('clicked', self.go_forward)
    toolbar_box.toolbar.insert(forward_button, -1)
    forward_button.show()

    # ...

  def go_back(self, button):
    """Back clicked, signal to JavaScript to update page count"""
    self.enyo.send_message("back_clicked", -1)

  def go_forward(self, button):
    """Forward clicked, signal to JavaScript to update page count"""
    self.enyo.send_message("forward_clicked", 1)
```

The "back_clicked" and "forward_clicked" events are sent to JavaScript when the toolbar forward buttons previous/next are clicked in Python. These events will receive a parameter (number 1 or -1) that tell to JavaScript to update page count in the HTML page.

Let's see now the JavaScript of the activity. It's no more complex.

Like the Python side, you'll start by embedding the framework in your code. Here you just had to include the "Sugar.js" file in the list of dependencies in the "package.js" file:

```
enyo.depends(
  "sugar.js",
  "app.js",

  // ...
);
```

Then you need to instantiate a "Sugar" class object in your HTML5 content.

```
this.sugar = new Sugar();
```

The Sugar object has the same "connect" and "sendMessage" methods that its Python counterpart. You could subscribe to Python messages using the "connect" method that you bind to a JavaScript method. Here the code to react to the Python Toolbar message "forward_clicked" explained before:

```
this.sugar.connect("forward_clicked",
                   enyo.bind(this, "upgradePageCount"));
```

Note than I'm using it the Enyo bind function to reference a JavaScript method but you could use the standard JavaScript syntax as well:

```
this.sugar.connect("forward_clicked",
                   function(args) { /* ... */ } );
```

The source code of the "upgradePageCount" method will give us opportunity to learn about the "sendMessage" method.

```
// Handle Python message coming from toolbar
upgradePageCount: function(args) {
  // Process toolbar button click
  var currentValue = parseInt(this.$.pageCount.getContent());
  switch(args)
  {
  case 1:
    currentValue++;
    break;
  case -1:
    currentValue--;
    break;
  case 0:
    currentValue = 1;
    break;
  }
  this.$.pageCount.setContent(currentValue);

  // Change toolbar button sensitivity depending of current page
  var back = "False";
  var forward = "False";
  if (currentValue == 1)
    back = "True"
  else if (currentValue == 10)
    forward = "True"
  this.sugar.sendMessage("disableBack", back);
  this.sugar.sendMessage("disableForward", forward);
},
```

As you probably understand, this method just upgrade the counter in the HTML5 page depending of the message parameter. Finally, you could see at the end of the method, two calls of the "sendMessage" method. These calls just ask to Python to change Toolbar button sensitivity depend of the current page. 1 is the minimum page number and I choose arbitrarily to set 10 as the maximum page number.

Here is a comeback to the Python source code to understand how it works:

```
    self.enyo.connect("disableBack", self.disable_back)
    self.enyo.connect("disableForward", self.disable_forward)

    # ...

  def disable_back(self, args):
    """Change sensitive status of the toolbar back button"""
    self.back_button.set_sensitive(args != "True")

  def disable_forward(self, args):
    """Change sensitive status of the toolbar forward button"""
```

This same process: "User interaction / Python message / JavaScript processing / JavaScript message / Python user interface changing" is used in the sample to synchronize HTML5 and Python controls. I will let you to discover how it works as an exercise.

Note three important things about message sending between Python and JavaScript:

- The parameter to the "sendMessage/send_message" method is optional. You could forget it. It will be replaced by a "null/None" value in the receiving method.
- The "sendMessage/send_message" method is call synchronously. There is no delay between the send messasge call and the connected method processing.
- Thanks to JSON format, the framework automatically handle value conversion between Python and JavaScript for basic data types (number, string, boolean, ...), arrays and object composed of basic data types and arrays. Let's see it in the source code of the sample in Python first:

```
class DummyObject:
  """Dummy class used for object transfer to JavaScript"""
  name = "Lionel"
  version = 2.0
  modified = date.today()
  language = ['Python', 'JavaScript']
  def foo(self):
    pass

def send_string(self, button):
  """Send a simple string to JavaScript"""
  self.enyo.send_message("helloFromPY", "Hello JavaScript !")

def send_object(self, button):
  """Send a simple string to JavaScript"""
  self.enyo.send_message("helloFromPY", self.DummyObject())
```

Then in JavaScript:

```
  // Send a simple string message to Python
  buttonStringClicked: function() {
    this.sugar.sendMessage("helloFromJS", "Hello Python !");
  },

  // Send a dummy JavaScript object to Python
  buttonObjectClicked: function() {
    var person = { name: "Lionel", version: 2.0,
                   language: ["Python", "JavaScript"] };
    person.modified = new Date();
    this.sugar.sendMessage("helloFromJS", person);
  },
```

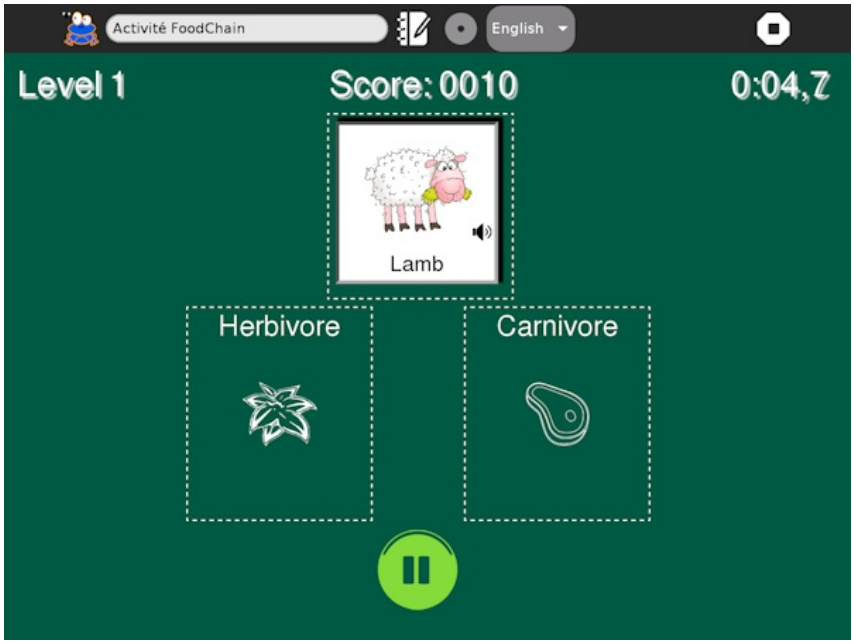
In both case you will receive an object like if it comes from the target language of the message.

THE NEXT STEP: PUBLISH YOUR HTML5 ACTIVITY FOR SUGAR

These first samples demonstrate the capacity to easily write new pedagogical Activities for Sugar using HTML5/JavaScript.

I've published, FoodChain, my first activity using the content of this chapter in the Sugar App Store. It's downloadable on http://activities.sugarlabs.org/en-US/sugar/addon/4612/.

FoodChain is a pedagogical game to learn the name of animals (word and pronunciation currently in French and English) and concept of food chains: Who eats what? Who eats who?



You can see the complete source code of the FoodChain activity on http://git.sugarlabs.org/foodchain-activity. It will show you some other advanced features that you could use for your own HTML5 Activity:

- Saving activity context into the Journal,
- Using Sugar localization for both Python and HTML5 contents,
- Embedding a debug console to log JavaScript message,
- Using of HTML5 extended features: media tag, drag&drop, SVG, Canvas, Touch support, ...,
- Multi-device handling (XO, PC, tablet).

20. PORTING YOUR SUGAR ACTIVITIES TO GTK3

by Aneesh Dogra

This is a guide to port existing sugar applications from gtk2 to gtk3, and from sugar to sugar3.

Important Changes in sugar3 :-

- The keep button has been removed completely
- The old-style toolbar has been removed
- Do not use set_toolbox anymore use set_toolbar_box instead.
- Remove import of deprecated ActivityToolbox. We have an ActivityToolbar in sugar3.activity.widgets now.
- Support for 'service_name' and 'class' has been removed from the activity.info make sure you are using: 'bundle_id' instead of 'service_name' and 'exec' instead of 'class'

This guide is organized into the following sections :-

1. **Imports**
2. **Gtk API changes**
3. **New Toolbar**
4. **Boxes**
5. **Pango**

IMPORTS

The first step in the gtk2->gtk3 porting is to replace/remove all the imports of gtk2 or any library using gtk2.

```
import gtk
```

to :

```
from gi.repository import Gtk
```

Any references to gtk.gdk should be replaced by Gdk and Gdk should be imported.

```
from gi.repository import Gdk
```

Any imports to sugar libraries should be replaced with sugar3 libraries (you need sugar-toolkit-gtk3)

```
from sugar.activity import activity
```

to:

```
from sugar3.activity import activity
```

do the same for other sugar.* imports.

Changes to the setup.py

You have to change the bundlebuilder import in your activity's setup.py file.

```
from sugar.activity import bundlebuilder
```

to:

```
from sugar3.activity import bundlebuilder
```

These are the header changes you need to do for ReadEText 1 Activity :-

```
import os
import zipfile
import gtk
import pango
from sugar.activity import activity
from sugar.graphics import style
```

to:

```
import os
import zipfile
from gi.repository import Gtk
from gi.repository import Gdk
from gi.repository import Pango
from sugar3.activity import widgets
from sugar3.activity.widgets import StopButton
from sugar3.activity import activity
from sugar3.graphics import style
```

Notice: That we add 2 more imports for StopButton and widgets. This is because the ActivityToolbox in sugar.activity.activity is replaced by ActivityToolbar in sugar.activity.widgets.

The ActivityToolbar doesn't have the stop button as in ActivityToolbox so we need to add it.

Please check the Gtk3 version of ReadETexts 1 activity to see how it is done.

GTK API CHANGES

There are a lot of API changes in Gtk3. I usually start the porting by replacing all gtk.* with Gtk.* and the fixing the errors one by one.

A simple script pygi-enumerate.py can be as a reference in the porting.

How to use?

1) Download pygi-enumerate.py

2) In the main function add the library you need to recurse.

```
do_recurse(LIB, "NAME")
```

like:

```
do_recurse(Gtk, "Gtk")
```

Add an import for the LIB in the header and let it rip.

When you run it, you'll see a long list of class methods, constants running through your terminal.

How do I use it to find my desired gtk2->gtk3 replacement?

A: grep is your friend!

Example: How to find the replacement of gtk.WRAP_WORD_CHAR

```
[aneesh-sugardev@localhost ~]$ python pygi-enumerate.py | grep
constant | grep -i word_char
Gtk.WrapMode.WORD_CHAR (integer constant)
```

Dissecting the command:

We use pipes to transfer stdout of one command to stdin of other.

1) python pygi-enumerate.py

This would print all the Class methods and functions to stdout.

2) grep constant

This would filter out all the constants.

3) grep -i word_char

The '-i' tells grep to ignore case

ReadEText I Activity Replacements:

First of all replace all the gtk.* with Gtk.*. Then look for errors and fix them one by one.

Hint: You might need to change some constants. Check the Gtk3 version in the code directory to see how its done.

Gtk.Adjustment: It turns out that Gtk.Adjustment class no longer contains class attributes like lower, upper, value or page size. To get these you need to use get_lower(), get_upper(), get_value() and get_page_size() functions respectively. To set the value you need to use set_value().

NEW TOOLBAR

sugar3 no longer has the ActivityToolbox, therefore you'll need to use the new ToolbarBox.

To use it we need to import it:

```
from sugar3.graphics.toolbarbox import ToolbarBox
```

To add ActivityToolbox widgets in your ToolbarBox you can use ActivityToolbarButton.

Import ActivityToolbarButton:

```
from sugar3.graphics.widgets import ActivityToolbarButton
```

This button can be added to the ToolbarBox:

```
        toolbar_box = ToolbarBox()
        activity_button = ActivityToolbarButton(self)
        toolbar_box.toolbar.insert(activity_button, 0)
        activity_button.show()
```

Similarly you can add other toolbars in your ToolbarBox:

```
        self.view_toolbar = ViewToolbar()
        self.view_toolbar.connect('go-fullscreen', \
            self.view_toolbar_go_fullscreen_cb)
        self.view_toolbar.zoom_in.connect('clicked', self.zoom_in_cb)
        self.view_toolbar.zoom_out.connect('clicked', self.zoom_out_cb)
        self.view_toolbar.show()
        view_toolbar_button = ToolbarButton(
            page=self.view_toolbar,
            icon_name='toolbar-view')
        toolbar_box.toolbar.insert(view_toolbar_button, -1)
        view_toolbar_button.show()
```

What we are doing here is, first we create our desired toolbar and then we create a button which if pressed collapses our toolbar.

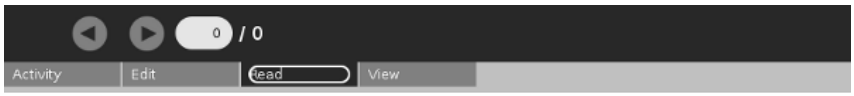
Don't have icons? Well, you can also add buttons with labels.

```
        view_toolbar_button = ToolbarButton(
            page=self.view_toolbar,
            label=_('View'))
```

Check GTK3 versions of ReadETexts Activity 3 and 4 in the code example directory for working examples.

Screenshots

Old Toolbar:



New Toolbar:

BOXES

Gtk Boxes have 2 common functions pack_start and pack_end. In Gtk2 these 2 functions could be called using 1-4 arguments (excluding self) that's because these functions were defined with default values. We no longer have default values is Gtk3, thus you are required to call pack_start and pack_end with 4 arguments (excluding self) only.

pack_start(widget, expand, fill, padding)

widget: The Gtk3 widget you wish to pack inside the box.

expand: Whether the child should get extra space when the container grows.

fill: True if space given to child by the expand option is actually allocated to child, rather than just padding it. This parameter has no effect if expand is set to False. A child is always allocated the full height of a Gtk.HBox and the full width of a Gtk.VBox. This option affects the other dimension.

padding: extra space in pixels to put between child and its neighbor.

Here's how it was defined in gtk2 :-

pack_start(widget, expand=True, fill=True, padding=0)

thus, you could use, in gtk2, pack_start(widget). **But in Gtk3 you have to provide all the arguments.**

pack_start(widget, True, True, 0)

even keyword arguments work :-

pack_start(widget, expand=True, fill=True, padding=0)

same is with pack_end.

In Gtk3, GtkHBox and GtkVBox have been depreciated, which suggests they might get removed in Gtk4. You can use the following for replacing your GtkVBox and GtkHBox respectively.

```
vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)

hbox = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL,
homogeneous=True, spacing=8)
```

Note: When GtkBox is used instead of GtkHBox or GtkVBox. The default value of expand is set to False.

PANGO

Pango adds a couple of useful tools in Gtk3. One of them is Pango Text Markup, you can use it to style your texts your way. Yes, no need for those attribute lists, its as simplified as HTML.

Why wouldn't I use attribute lists instead?

The problem with attribute list is that you'd need to apply attributes to some numeric range of characters, for example "characters 12-17." This is broken from an internationalization standpoint; once the text is translated, the word you wanted to italicize could be in a different position.

How to fix this without markup?

Okay, so you don't want to use markup and still want to stick to attribute list. One way to fix the above issue would be to set your page as a GtkToolItem and by inserting GtkLabels with custom attributes. So, if you wanna add "A **bold**" you'll have to use:-

```
page = Gtk.ToolItem()
l1 = Gtk.Label()
l1.set_text('A ')
page.insert(l1)

l2 = Gtk.Label()
l2.set_text('bold')
attr = pango.AttrList()
attr.change(pango.AttrWeight(pango.WEIGHT_BOLD, 0, -1))
l2.set_attributes(attr)

page.insert(l2)
```

Painstaking? Pango Text Markup to the rescue.

How to fix this with markup?

Markup is the most elegant way to solve the above problem. So, if you wanna add "A **bold**" and style it using markup, it'll take just 2 lines.

```
l1 = Gtk.Label()

l1.set_markup("A <b>bold</b>")
```

Note: While porting from Pango gtk2 to Pango gtk3 you also need to change some constants. The same tool discussed in Gtk Api Changes sections can be used to do the needful.

CONVERTING HIPPO CANVAS TO GTK3

by Aneesh Dogra

When the Sugar environment was first created it made a lot of use of Hippo Canvas and so did some Activities. Lately Sugar Labs has reconsidered that and Sugar 3 will use no Hippo Canvas at all

Sugar used Hippo because it had some significant improvements over Gtk Widgets, In the past time it was the only way to do some things, like layout modernization (see [1]), but now most of its features can be done in gtk using Cairo, Pango and Gtk3.

Sugar 3 has been completely ported to Gtk3 and doesn't use Hippo canvases. To make your application Gtk3 based you will have to remove Hippo dependencies and use Gtk3 widgets to do the job. Gtk3 makes that a little easy for us, as we are provided with CSS, Pango Markup and Boxes which can replace Hippo to create an identical UI.

PORTING

CanvasBox Object

A hippo.CanvaxBox() could include the following arguments.

orientation: Whether it should be a Vertical box or a Horizontal one.

background_color: The box's background color.

box_width: The box's width

box_height: The box's height

padding: The padding between the Box and its surrounding widgets. Analogous to padding in Gtk.Box.pack_start() and Gtk.Box.pack_end()

spacing: The spacing between the Box and its surrounding widgets.

A CanvasBox can be easily ported to Gtk3 based code using a Gtk.Box

Example:

```
        conversation = hippo.CanvasBox(
                       spacing=0,
                       background_color=COLOR_WHITE.get_int())
        self.conversation = conversation
```

The above CanvasBox call can be replaced by :-

```
        self.conversation = Gtk.VBox()

self.conversation.override_background_color(Gtk.StateType.NORMAL, \
                           Gdk.RGBA(*COLOR_WHITE.get_rgba()))
```

CanvasText Object

A hippo.CanvasText could include the following options

text: The text you wish to display
size_mode: Wrap mode.
color: Text color.

font_desc: A Pango.FontDescription object, used to style the text.

xalign: Start or End. Start for Left to Right languages and End for Right to Left languages.

A CanvasText object can be converted to a Gtk3 widget using textview.

Example:

```
message = hippo.CanvasText(
    text=text,
    size_mode=hippo.CANVAS_SIZE_WRAP_WORD,
    color=text_color,
    font_desc=FONT_NORMAL.get_pango_desc(),
    xalign=hippo.ALIGNMENT_START)
```

The above CanvasText call can be replaced by :-

```
msg = Gtk.TextView()
text_buffer = msg.get_buffer()
text_buffer.set_text(text)
msg.set_editable(False)
msg.set_justification(Gtk.Justification.LEFT)
msg.set_font(FONT_NORMAL.get_pango_desc())
msg.override_color(Gtk.StateType.Normal,  text_color)
# You need to convert text_color to a Gdk.RGBA object.
msg.set_wrap_mode(Gtk.WrapMode.WORD_CHAR)
```

CanvasScrollbars Object

Its almost similar to the way Gtk.ScrolledWindow works.

Example:

```
    sw = hippo.CanvasScrollbars()
    sw.set_policy(hippo.ORIENTATION_HORIZONTAL,
hippo.SCROLLBAR_NEVER)
    sw.set_root(conversation)
```

The above hippo implementation can be replaced by:

```
    sw = Gtk.ScrolledWindow()
    sw.set_policy(Gtk.PolicyType.NEVER,
                      Gtk.PolicyType.AUTOMATIC)
    sw.add(conversation)
```

CSS STYLING

At some points you might want to use CSS to style your widgets, because Its more manageable secondly CSS is more documented than Gtk.

Here's how you can do it:

Add the following in your application's init (you need to import Gtk and Gdk) :-

```
        screen = Gdk.Screen.get_default()
        css_provider = Gtk.CssProvider()
        css_provider.load_from_path(FILE)
        context = Gtk.StyleContext()
        context.add_provider_for_screen(screen,
                                        css_provider,

Gtk.STYLE_PROVIDER_PRIORITY_USER)
```

You need to change FILE to your desired CSS file's path.

A basic CSS file

```
GtkLabel {
```

```
    background-color: red;
}
```

Note: In the style sheet we define styles using their gtype names.

It means you can also set gtype name to a specific widget and style it specifically.

PORTING MINICHAT FROM HIPPO TO GTK3

MiniChat can be found in the book code examples repository, by the name /MiniChat (Hippo Version) and /MiniChat_gtk (GTK3 version)

Screen Shots

Hippo Version

GTK3 Version:

As you can see there isn't much difference in the 2 layouts, except that the GTK3 version's chatbox has sharp corners instead of rounded ones.

Porting

Understanding the layout

The Hippo version uses the following layout.

```
 .------------- rb ---------------.
| +name_vbox+ +----msg_vbox----+ |
| |         | |                | |
| | nick:   | | +--msg_hbox--+ | |
| |         | | | text       | | |
| +---------+ | +------------+ | |
|             |                | |
|             | +--msg_hbox--+ | |
|             | | text       | | |
|             | +------------+ | |
|             +----------------+ |
`--------------------------------'
```

We can replicate it easily in Gtk3. Using a Horizontal box for rb, A TextView for name_vbox and another TextView for the messages.

Thus, our Gtk3 layout would be :-

```
.------------- rb ---------------.
| +name_vbox+ +----msg_vbox----+ |
| |         | |                | |
| | nick:   | | +------------+ | |
| |         | | |    Text    | | |
| +---------+ | +------------+ | |
|             +----------------+ |
`--------------------------------'
```

Its similar to the Hippo version just a lot simpler. Note that we don't need msg_hbox in the Gtk3 version because TextView are enough for the functionality we are looking for.

What do we need to port?

Let's checkout the function calls we need to port/replace. Grepping for "hippo." in the MiniChat/minichat.py would give you a list.

1. hippo.CanvasBox (used with name_vbox, msg_vbox, msg_hbox and for the conversation box)
2. hippo.CanvasScrollbars (used for adding Scrollbar to the conversation box)
3. hippo.Canvas (the container of conversation box) [We'll not require it in out Gtk3 version.]
4. hippo.CanvasText (used for displaying nick and messages)
5. CanvasRoundBox (used with rb)

What can they be replaced with?

Now, we need to think about possible Gtk3 widgets with which we can replace these hippo elements.

1. hippo.CanvasBox can be replaced with Gtk.Box
2. hippo.CanvasScrollbars can be replaced with Gtk.ScrolledWindow
3. hippo.CanvasText can be replaced with Gtk.TextView
4. hippo.Canvas is not required.
5. CanvasRoundBox [part of sugar, not available in sugar3] can be replaced with a Gtk.Box.

Note: Gtk.Box doesn't support rounded corners, there is a way to add rounded corners to Gtk.Box but that requires Cairo which is out of the scope of this chapter. You can check how it is implemented in the Chat activity.

Replacing

The Conversation Box

The conversation box is a container of rb's [rb is a box in our layout, as discussed above]

The conversation box is defined in the make_root function as :

```
        conversation = hippo.CanvasBox(
            spacing=0,
            background_color=COLOR_WHITE.get_int())
        self.conversation = conversation
```

Now, it needs to be a VBox because you expect your chat messages to be displayed linearly from top->bottom not from left->right. This can be replicated in GTK3 as:

```
self.conversation = Gtk.VBox()
self.conversation.override_background_color(Gtk.StateType.NORMAL, \
                                Gdk.RGBA(*COLOR_WHITE.get_rgba()))
```

The Scrollbar

The scrollbar is defined in make_root function as :

```
        sw = hippo.CanvasScrollbars()
        sw.set_policy(hippo.ORIENTATION_HORIZONTAL,
hippo.SCROLLBAR_NEVER)
        sw.set_root(conversation)
        self.scrolled_window = sw
```

This can be replicated in Gtk3 using Gtk.ScrolledWindow :

```
        self.scroller = Gtk.ScrolledWindow()
        self.scroller.set_vexpand(True)
        self.scroller.add_with_viewport(self.conversation)
        self.scroller.override_background_color(Gtk.StateType.NORMAL,

Gdk.RGBA(*COLOR_WHITE.get_rgba()))
```

The Round Box (RB)

The rb contains our message boxes and our name boxes.

Here's how it is defined in the Hippo Version:

```
            rb = CanvasRoundBox(background_color=color_fill,
                                border_color=color_stroke,
                                padding=4)
```

GTK boxes doesn't have any functionality to add border_colour but we can add borders to boxes by adding them in an EventBox. Here's how :

```
            eb = Gtk.EventBox()
            eb.override_background_color(Gtk.StateType.NORMAL,
color_stroke)

            rb = Gtk.HBox()
            rb.override_background_color(Gtk.StateType.NORMAL,
color_fill)
            rb.set_border_width(1)
            eb.add(rb)
```

The Boxes

msg_vbox:

```
 msg_vbox = hippo.CanvasBox(
                orientation=hippo.ORIENTATION_VERTICAL)
```

In Gtk3:

```
            msg_vbox = Gtk.VBox()
```

name_vbox:

```
                name_vbox = hippo.CanvasBox(
                    orientation=hippo.ORIENTATION_VERTICAL)
```

in Gtk3:

```
                 name_vbox = Gtk.VBox()
```

msg_hbox

msg_hbox is required in the Hippo version because hippo.CanvasText doesn't have a box so we need a container where we can append the text, but Gtk.TextView provides us with a box as well as a TextBuffer. Thus, we don't require msg_hbox in Gtk3.

Adding Messages

In the hippo version, every time a message is posted, the message is appended in the msg_hbox which is eventually appended to the msg_vbox.

```
             message = hippo.CanvasText(
                text=text,
                size_mode=hippo.CANVAS_SIZE_WRAP_WORD,
                color=text_color,
                font_desc=FONT_NORMAL.get_pango_desc(),
                xalign=hippo.ALIGNMENT_START)
             msg_hbox.append(message)
```

In Gtk3 we can make this simpler, Firstly we don't need the msg_hbox (as discussed in the preceding sections), secondly instead of creating new message boxes everytime a new message is posted we can append the text to the textview.
Here's how:

```
            if not new_msg:
                msg = msg_vbox.get_children()[0]
                text_buffer = msg.get_buffer()
                text_buffer.set_text(text_buffer.get_text(
                    text_buffer.get_start_iter(),
                    text_buffer.get_end_iter(), True) + "\n" + text)
            else:
                msg = Gtk.TextView()
```

```
text_buffer = msg.get_buffer()
text_buffer.set_text(text)
msg.show()
msg.set_editable(False)
msg.set_border_width(5)
msg.set_justification(Gtk.Justification.LEFT)
msg.override_font(FONT_NORMAL.get_pango_desc())
msg.override_color(Gtk.StateType.NORMAL, text_color)
msg.override_background_color(Gtk.StateType.NORMAL, \
                              color_fill)
msg.set_wrap_mode(Gtk.WrapMode.WORD_CHAR)
msg_vbox.pack_start(msg, True, True, 0)
```

Colors

Hippo elements generally input integer form of colors but in Gtk you need the Gtk.RGBA type object.

example:

```
text_color = COLOR_WHITE.get_int()
```

should be replaced by:

```
text_color = Gdk.RGBA(*COLOR_WHITE.get_rgba())
```

These were the major changes required to port the MiniChat activity from Hippo to Gtk3, you can checkout the fully ported activity in /MiniChat_gtk3

[1]: https://bugzilla.gnome.org/show_bug.cgi?id=310809

22. MAKING CONTRIBUTIONS TO SUGAR

by Daniel Francis

WHY TO CONTRIBUTE TO SUGAR?

I started contributing to Sugar itself when I was 14 years old after meeting some of the Sugar Labs developers. I saw that the same people maintain activities and fix the necessary bugs in Sugar to make those activities work. I noted that I can improve the Sugar activity development experience because the Sugar code and the activities code aren't very different and I think the free and open source software needs to be improved or fixed and the community can help on it.

Sometimes, as a user you may only report an issue to the developers, but in Sugar it is easier to know how to fix it and improve the Sugar components by yourself.

This can be the reason because at the moment I also have reported bugs in the GNOME bugzilla due some issues in Gtk and other libraries maintained by GNOME, but I only submitted code contributions to Sugar and its activities.

IDENTIFYING A BUG IN SUGAR TO WORK ON

The way I found to exemplify a contribution to Sugar, is telling when I wanted to add accessibility to one of my activities through implementing key accelerators.

Key accelerators are easy to implement in Sugar.

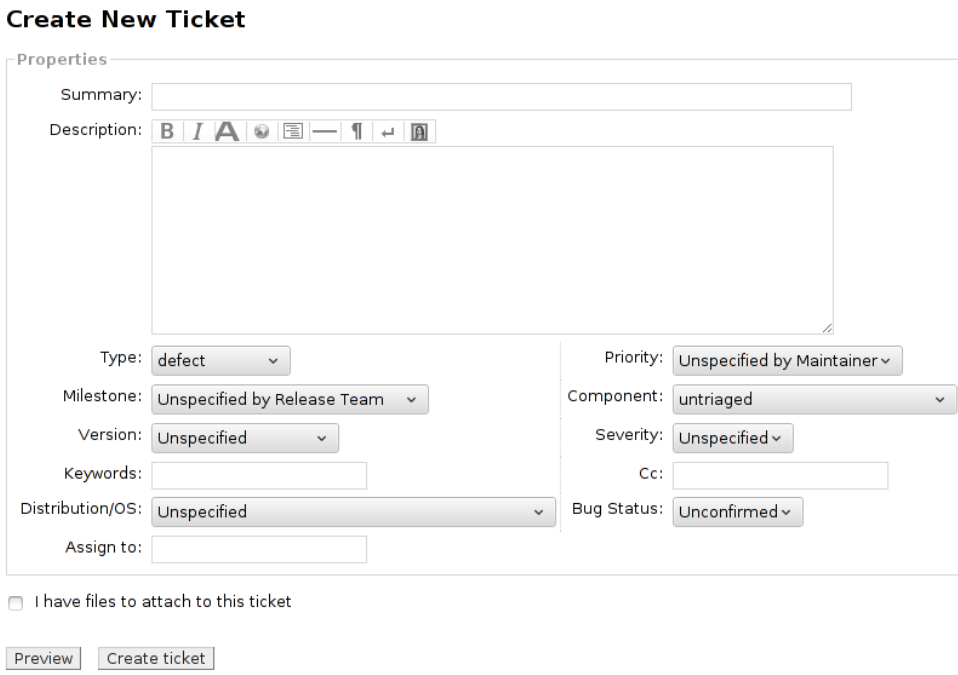
```
# First you create a button
toolbutton = ToolButton('gtk-quit')
toolbutton.props.tooltip = 'Bold font'
# And here you set the key accelerator
toolbutton.props.accelerator = '<Ctrl>Q'
```

When doing the same with a ToggleToolButton, I got an error because ToggleToolButtons didn't have the accelerator property. Of course a lot of activities weren't using key accelerators.

REPORTING THE BUG

To discuss if it's really an issue and if must be solved, the usual procedure consists in open a ticket in bugs.sugarlabs.org

A bug tracker is a place where people report bugs and some developers add tasks. The result is a list to tell maintainers or contributors how can they fix the project source code. So, if you want to contribute to Sugar but you didn't find a bug or you don't have a great idea, you can search for tickets in the Sugar Labs bug tracker.



The ticket I created is in: http://bugs.sugarlabs.org/ticket/3774

Some important ticket fields

- Summary

  This is like the title of the ticket. The issue in few words.

  In this case the summary was: **Missing key accelerators at ToggleToolButton**

- Description

  All the details of the bug including steps to reproduce it (if any). If it's graphic, you can link to a screenshot. If you have a log, you can attach it or paste a trace of the log in the ticket description.

  In this case it was: **The accelerator property, typical in the Sugar ToolButtons isn't available for the Sugar ToggleToolButtons.**

- Type

  Defect, enhancement or task. A defect is when doesn't work as expected, an enhancement is when you want to add a new feature but it must be discussed and a task is when maintainers have a task which can be assigned and solved without a discussion or confirmation like adding activities to Pootle or adding components to this bugtracker.

- Component

  This is the part of Sugar which must be modified. In the list there are activities, web services such as Pootle or bugs.sugarlabs.org which allows you to add your activity to the bugtracker and of course the Sugar environment which is divided in components. The components for the Sugar API are called sugar-toolkit(-gtk2) and **sugar-toolkit-gtk3**.

- Version

  Here you can select the Sugar Version you are using or **"Git as of bug date"** if you are using jhbuild or sugar-build.

WAITING LONG TIME FOR A REPLY? DO IT BY YOURSELF!

Using the Sugar development version

The latest version of the Sugar source code is in git.sugarlabs.org. A convenient tool for developers is 'sugar-build', which will build Sugar from git as a well as pull in any dependencies required by Sugar. (Sugar-build replaces sugar-jhbuild, which is no longer maintained.)

Sugar-build clones and installs sugar in its own directory, so you can change and test the sugar sources without any root administrative password.

Since both Sugar and sugar-build change very often, a description of the steps to build sugar would likely be obsolete before this book is published. However, you can read up-to-date documentation at:

http://sugarlabs.org/~dnarvaez/sugar-docs/

After the patches work in a sugar-build environment and they are applied in the git repository, you can test them on an OLPC development build, which are released at least once a month.

The OLPC development builds are available at http://build.laptop.org/ and they are announced in the Olpc-devel Mailing List, http://lists.laptop.org/listinfo/devel

Modifying the Sugar source code

The first part of the work is find the code files related to the functionality you want to edit. In this case the sugar toolkit in gtk3 is cloned in *sources/sugar* and the module I edited is in src/sugar3/graphics/toggletoolbutton.py

You can see all the changes with a diff format in:

http://patchwork.sugarlabs.org/patch/1605/

I only had to copy code from toolbutton.py, here some parts I copied to the toggletoolbutton:

```
def _add_accelerator(tool_button):
  if not tool_button.props.accelerator or \
        not tool_button.get_toplevel() or \
            not tool_button.get_child():
    return

  # TODO: should we remove the accelerator from the prev top level?
  if not hasattr(tool_button.get_toplevel(), 'sugar_accel_group'):
    logging.warning('No Gtk.AccelGroup in the top level window.')
    return

  accel_group = tool_button.get_toplevel().sugar_accel_group
  keyval, mask = Gtk.accelerator_parse(tool_button.props.accelerator)
  # the accelerator needs to be set at the child, so the
  # Gtk.AccelLabel in the palette can pick it up.
tool_button.get_child().add_accelerator('clicked', accel_group,
        keyval, mask, Gtk.AccelFlags.LOCKED | Gtk.AccelFlags.VISIBLE)
def _hierarchy_changed_cb(tool_button, previous_toplevel):
  _add_accelerator(tool_button)


def setup_accelerator(tool_button):
  _add_accelerator(tool_button)
  tool_button.connect('hierarchy-changed', _hierarchy_changed_cb)


class ToggleToolButton(Gtk.ToggleToolButton):
...
 def set_accelerator(self, accelerator):
    self._accelerator = accelerator
    setup_accelerator(self)

 def get_accelerator(self):
    return self._accelerator

accelerator = GObject.property(type=str, setter=set_accelerator,
           getter=get_accelerator)
```

MAKE A PATCH

Sugar-build clones from git all the Sugar components inside a directory called sources. After editing the code, you must build sugar again to test your changes following the instructions of the sugar-build documentation.

After editing and testing, you must commit your change and make a diff file the maintainers can review and push to the repository.

```
$ git add src/sugar3/graphics/toggletoolbutton.py
$ #Note that SL#xxxx is the bugs.sl.org ticket number
$ git commit -s -m "Add accelerator for toggletoolbutton SL#3774"
$ git format-patch -1
```

After that, you will have generated a file like this and you only have to replace the word PATCH with PATCH sugar-toolkit or the component you are patching.

```
From: Daniel Francis
Date: Sat, 29 Dec 2012 14:10:34 -0200
Subject: [PATCH sugar-toolkit] Add accelerator for toggletoolbutton
SL#3774

Signed-off-by: Daniel Francis
---
diff --git a/src/sugar3/graphics/toggletoolbutton.py
b/src/sugar3/graphics/toggletoolbutton.py
index 94cc6ae..f3a9c57 100644
--- a/src/sugar3/graphics/toggletoolbutton.py
+++ b/src/sugar3/graphics/toggletoolbutton.py
@@ -19,6 +19,8 @@
 STABLE.
 """

+import logging
+
 from gi.repository import GObject
 from gi.repository import Gtk

@@ -26,6 +28,33 @@  from sugar3.graphics.icon import Icon
 from sugar3.graphics.palette import Palette, ToolInvoker


+def _add_accelerator(tool_button):
+    if not tool_button.props.accelerator or\+          not
tool_button.get_toplevel() or \
+                    not tool_button.get_child():
+        return
....
```

You can send it as an email if you have installed and configured git-send-email or else you can attach the patch in the bugs.sugarlabs.org ticket. If you choice the first one, the correct procedure is comment the ticket telling that you patched it and giving a link to the mailing list archive.

If all is OK, the patch will be applied in the git repository, the commit title will appear in the release notes with your name as the author and will be part of an OLPC build after some time because OLPC releases builds about once or twice an year.

BENEFIT TO OTHER PEOPLE

In this case, there was a bug ticket to add key accelerators in the Write activity and they already noted the issue. I fixed it first so they tested the patches and requested the sugar-toolkit maintainers to reply my bug ticket and review my patch faster.

23. MAKING ACTIVITIES THAT USE THE ACCELEROMETER

by Aneesh Dogra

WHAT IS AN ACCELEROMETER?

As the name suggests an accelerometer is a device which measures acceleration forces in all directions. The XO 1.75+ provides us with a 3-axes accelerometer. A 3-axes accelerometer has 3 axes: X, Y, and Z. X & Y axes are parallel to the plane of the ground and the Z axis is perpendicular to the plane of the ground.

HOW TO GET THE DATA?

Getting accelerometer data is as simple as reading from a file. The accelerometer writes its position data in the following file:

```
/sys/devices/platform/lis3lv02d/position
```

This file contains a tupple of (x, y, z) values. Where x, y and z are the acceleration forces in X, Y and Z axes receptively. The x, y, z values range from -1152 to +1152.

The X axis can be used to measure Right and Left inclinations. If you dip your XO to the right you'll get positive X-axis values; if you dip your XO to the left you'll get negative X-axis values.

The Y and Z axes can be used to measure up and down inclinations. Y & Z axes values are positive when you dip your XO towards up; they are negative when you dip your XO towards the bottom.

When the XO is leveled i.e its kept parallel to the plane of the ground X and Y axis values are approximately 0 and Z axis value is -1152.

Here's how the values are get in the Level Tool [1] :

```
def read_accelerometer(canvas):
        fh = open(ACCELEROMETER_DEVICE)
        string = fh.read()
        xyz = string[1:-2].split(',')
        x = float(xyz[0]) / (64 * 18)
        y = float(xyz[1]) / (64 * 18)
        fh.close()
        canvas.motion_cb(x, y)
        return True
```

POLLING FOR ACCELEROMETER VALUES

If you are using accelerometer values in your code, most probably you will need to poll them constantly checking for changes. This can be easily done using GObject's add_timeout method.

```
GObject.timeout_add(interval, callback, ...)
```

interval is the timeout between calls to the function in milliseconds.

callback is the function you wish to call.

The `GObject.timeout_add()` function sets a function (specified by *`callback`*) to be called at regular intervals. Additional arguments to pass to *`callback`* can be specified after *`callback`*. The function is called repeatedly until it returns `FALSE`, at which point the timeout is automatically destroyed and the function will not be called again. The first call to the function will be at the end of the first interval. Note that timeout functions may be delayed, due to the processing of other event sources.

To Poll the function we read about in the previous section, just add the following line in the main or the init function:

```
GObject.timeout_add(100, read_accelerometer, canvas)
```

This would call the function - "read_accelerometer" - every 0.1 seconds.

STOP POLLING

Some times you might need to stop polling a particular function. Here's how it is done.

You can use the GObject's source_remove method to stop polling a particular function.

```
GObject.source_remove(tag)
```

where tag is the integer returned by the timeout_add function.

TESTING

Your activity can be tested without an XO too. You can use the Accelerometer Emulator [2].

It has a simple interface with 5 buttons, Left, Right, Up, Down and Reset. These buttons can be used to simulate acceleration forces is that particular direction.

Using the accelerometer emulator

The emulator writes to a particular file pointed by the PATH variable and updates it in real-time, as in the case of an accelerometer. To integrate this emulator in your activity you need to change the ACCELEROMETER_DEVICE_PATH in your activity to PATH. Read more [2]

ACTIVITIES USING THE ACCELEROMETER

Activities using the accelerometer are pretty scarce but we do have some activities to take inspiration from :

1. Level Tool [1]
2. Turtle Art [3]

[1] : git.sugarlabs.org/level-tool

[3] :
http://git.sugarlabs.org/turtleart/mainline/blobs/master/plugins/accelerometer/accelerometer.py

APPENDIX

24. ABOUT THE AUTHORS

JAMES SIMMONS

James Simmons has programmed professionally since 1978. Back then computer programs were made using a special machine that punched holes into cards, reels of tape were the most common data storage medium, and hard disks were so expensive and exotic that the hard disk inventory of a Fortune 500 company would today be considered barely large enough to hold a nice picture of Jessica Alba.

The industry has come a long way since then, and to a lesser extent so has James.

James learned to program at Oakton Community College in Morton Grove, Illinois and Western Illinois University in Macomb, Illinois. Times were hard back then and a young man's best chance of being employed after graduation was to become an Accountant or a Computer Programmer. It was while he attended OCC that James saw a Monty Python sketch about an Accountant who wished to become a Lion Tamer. This convinced James that he should become a Computer Programmer.

James' studies at WIU got off to a rough start when he signed up for Basic Assembly Language as his first real computer class, erroneously thinking that the word "Basic" meant "for beginners". From the computer's point of view it was basic, but for students not so much. He barely passed the course with a "D" but in the process learned that he enjoyed programming computers. He decided to continue his computer studies and graduated with a Bachelor's Degree in Information Science.

James was born in 1956, the year before Sputnik went up. He was a nerdy kid. At various times he fooled around with Erector sets, chemistry sets, microscopes, dissecting kits, model cars, model planes, model rockets, amateur radio, film making, and writing science fiction stories. He achieved no real success with any of these activities.

James participated in the first Give One Get One promotion of the One Laptop Per Child project and started developing Activities for the Sugar platform soon after. He has written the Activities **Read Etexts**, **View Slides**, **Sugar Commander** and **Get Internet Archive Books**.

OCEANA RAIN FIELDS

Oceana Rain Fields – Oceana is a visual artist and creative spirit with a flair for the unexpected and the desire to support worthy causes with her art. She graduated in 2010 from Pacific High School, earning several notable scholarships. In 2010, her painting “Malaria” won first in show in the Vision 2010 high school art competition at the Coos Art Museum in Coos Bay, Oregon. Oceana plans to continue her art education at Southwestern Oregon Community College in Fall 2010.

Oceana is responsible for the cover art of the bound and printed version of this book. As a "mentee" of the Rural Design Collective, she also did cover and interior illustrations for another FLOSS Manual: *E-Book Enlightenment: Reading and Leading with One Laptop Per Child*.

ANEESH DOGRA

Aneesh Dogra is a student from India with a passion for computing. He has been programming since 2009 and likes to code in Python, C and PHP. He got involved with Sugar Labs during Google Code-In 2012 and has contributed to Sugar activities like **Calculate**, **Write**, **IRC**, **Get Internet Archive Books**, **View Slides**, **Read Etexts** and many more.

Aneesh is responsible for porting the book code samples to Gtk3 and has written the chapters on Gtk2->Gtk3 Porting, Hippo Canvas->Gtk3 porting and Developing accelerometer based activities for this book.

You can get in touch with Aneesh at anee.me or at gplus.to/lionaneesh

LIONEL LASKÉ

Lionel Laské is the director of a business unit in a software company in France. He has an extensive background in software development including both Windows technologies and multi-platform environments. He's also the author of several applications for different mobile devices (webOS, Android, Windows Phone, iOS). He created "Liogo", one of the first existing Logo compilers. He has contributed articles to several website and magazines, including *Dr.Dobb's Journal*, *Code Project*, *Programmez*, *GreenIT.fr*, *DotNetGuru*, and others.

Lionel has worked as a volunteer on the One Laptop Per Child project since 2007. He has contributed to French localization, has written workshops on Mono development in Sugar and has tutored a project to create a nutrition Activity. Lionel is president and co-founder of OLPC France, a very active team of volunteers. OLPC France has lead a deployment of 200 XO laptops in Madagascar, another experimental deployment of 50 XO laptops in Saint-Denis, and has organized two SugarCamp events in Paris in 2009 and 2011.

Lionel is responsible for the chapter *Developing Sugar Activities using HTML5*. You can contact him at lionel AT olpc-france.org.

S. DANIEL FRANCIS

Santiago Daniel Francis (known as Daniel) is a student from Uruguay. He has used Sugar at school and got interested in programming in 2010. He likes coding in Python, C, Bash and JavaScript.

In 2011 he got involved in CeibalJAM, a volunteer managed community which intends to provide educational resources adapted to the needs of schools in Uruguay. As part of that community he developed Sugar Activities like **Graph Plotter**, which is used by high school students for maths lessons and to prepare for exams. This Activity eliminates the need to buy expensive graphing calculators.

In 2012 he met some Sugar Labs developers and got involved in Sugar Labs, where he fixed bugs in the Sugar platform. These bug fixes have been included in the official Sugar distribution. He has also helped to update some Sugar activities to use GTK 3.

Daniel coded new features to Sugar when he participated in the Google Code-in 2012. For this book he wrote the chapter *Making Contributions to Sugar* and the section *Porting from Olpcgames to Sugargame* of the chapter *Making Activities Using Pygame*.

Also in 2012 Daniel was elected to be part of the **Sugar Labs Oversight Board** and as a result he needed to disqualify himself as a contestant in the Google Code-in. At age fourteen he is currently the youngest member of that board.

You can contact Daniel at francis AT sugarlabs.org or at gplus.to/sdanielf.

25. WHERE TO GO FROM HERE?

This book attempts to give a beginning programmer the information she needs to develop and publish her own Sugar Activities. It already contains many URL's of websites containing information not covered in the book. This chapter will contain URL's and pointers to still more resources that will be useful to any Sugar developer.

LEARNING PYTHON

List compiled by Mike Rehner

- Finch Robot Python: http://www.finchrobot.com/software/python
- Google's Python Class: http://code.google.com/edu/languages/google-python-class/
- Guido van Robot home page: http://gvr.sourceforge.net/
- Invent with Python (Computer Games for students age 10+): http://inventwithpython.com/
- Interactive Python Tutorial: http://www.trypython.org/
- Khan Academy Python: http://www.khanacademy.org/#computer-science
- Learn Python (a teacher writes about teaching Python to 8th and 9th graders): http://learnpython.wordpress.com/
- Lego Robotics NXT-Python: http://code.google.com/p/nxt-python/
- Official Python Documentation: http://docs.python.org/
- On-Line Python Learning: http://cscircles.cemc.uwaterloo.ca/
- On-Line Python Tutor: http://people.csail.mit.edu/pgbovine/python/
- Panda 3D (3D Game Framework): http://www.panda3d.org/
- Python Cloud IDE (web based interpretor): http://pythonfiddle.com/
- Python Games Resources: http://wiki.python.org/moin/PythonGames
- Python Tutorials for Kids 8+: http://python4kids.wordpress.com/
- Snake Wrangling for Kids (downloadable book): http://www.briggs.net.nz/snake-wrangling-for-kids.html
- Udacity- on line learning where most classes are in Python: http://www.udacity.com/

PYGTK BOOK BY PETER GILL

Much of the work you will do writing Activities involves PyGTK. Peter Gill is working on a PyGTK book that covers the subject in great detail. You can download the book here:

- http://www.majorsilence.com/PyGTK_Book

OLPC AUSTRIA ACTIVITY HANDBOOK

This is the first attempt to write a manual on creating Sugar Activities. It is aimed at experienced programmers and covers topics that this book does not, like how to write Activities using languages other than Python. The book was written in 2008 and as a result some of the advice is a bit dated. It's still an excellent source of information. The authors are Christoph Derndorfer and Daniel Jahre.

- http://wiki.sugarlabs.org/images/5/51/Activity_Handbook_200805_online.pdf
- http://www.olpcaustria.org

THE SUGAR ALMANAC

This is a series of Wiki articles covering the Sugar **API** (**Application Programming Interface**). It's a good source of information that I have referred to many times.

- http://wiki.sugarlabs.org/go/Development_Team/Almanac

SUGAR LABS MAILING LISTS

Sugar Labs has several email mailing lists that might be worth subscribing to. The ones I follow most are the **IAEP** (**It's An Education Proiject**) list and **Sugar-Devel**. Sugar-Devel is a good place to ask questions about developing Sugar Activities and learn about the latest work being done on Sugar itself. IAEP is a good place to get ideas on what kinds of Activities teachers and students want and to get feedback on your own Activities. Anyone can sign up to these mailing lists here:

- http://lists.sugarlabs.org/

PYDOC

PyDoc is a utility for viewing documentation generated from the Python libraries on your computer, including the Sugar libraries. To run it use this command from a terminal:

```
pydoc -p 1234
```

This command will not finish. It runs a kind of web server on your system where *1234* is a port number. You can access the website it serves at **http://localhost:1234**. There is nothing magic about the number 1234. You can use any number you like.

The website lets you follow links to documentation on all the Python libraries you have installed. When you are done browsing the documentation you can stop the pydoc command bt returning to the terminal and hitting Ctrl-C (hold down the Ctrl key and hit the "c" key).

FLAVIO DANESSE

Flavio is the founder of *Python Joven*, a club for young Python developers who have written many Sugar Activities. His websites are a terrific source of information for anyone who wants to create Sugar Activities. The websites are in Spanish but support translation through Google Translate. I found the English translation to be quite readable.

- https://sites.google.com/site/flaviodanesse/
- https://sites.google.com/site/sugaractivities/

26. LICENSE

ACKNOWLEDGEMENTS

Many people contributed to this book besides the authors listed. They offered advice, technical support, corrections, and much code. If I tried to list all of their names I might leave someone out, so let me just thank all the members of the Sugar-Devel mailing list.

Cover art of the printed version Copyright (C) 2010 by Oceana Rain Fields.

Free manuals for free software

GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS