# ACTIVITY HANDBOOK

How to get started with
software development
for the OLPC XO

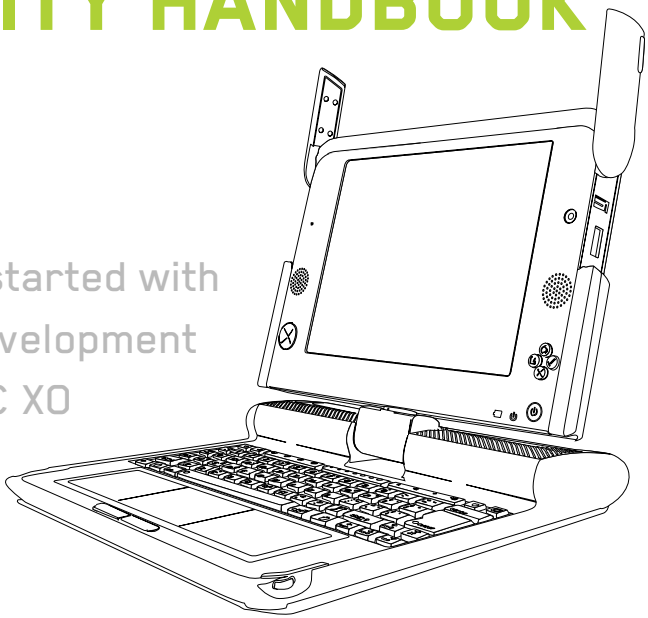


Table of Contents

# Welcome to the Activity Handbook!

The purpose of this handbook is to provide you with all the information you need in order to get started with software development for the OLPC XO. We assume that all of you have used books such as „Teach Yourself C / Java / Visual Basic in 21 Days" and we want to provide a similar source of information for interested readers. The main difference here is that this handbook is not going to teach you the basics of programming. Neither are we going to teach you Python, the programming language we are mostly going to rely on, in this handbook. There are many other sources for learning these things and it would go far beyond the scope of this handbook to serve as an introduction to programming and / or Python. What we however do want to give you is a head start when it comes to writing software for the XO. To that end we will give you a detailed overview of both the hardware and software which is used in this remarkable machine.

Our motivation to write this handbook arose from our own desire to write code for the XO. We just did not know how to do it and the documentation we found on the internet was not as helpful as we had hoped. So we basically consider ourselves to be members of the target group that we envision this document to be useful for. We are writing the document we had hoped to find when we started out with XO software development ourselves. When we started working on this handbook in early October 2007 the documentation, especially when it comes to the software, of the XO left much to be desired. Realizing this OLPC had started looking for a „documentation lead" and various discussions took place in order to gather feedback on what documentation was needed. The OLPC Wiki contained several entries related to software development but there was no single coherent document that contained all the information a developer needs in order to get started with writing code. So while it was certainly doable to find that information it was by no means comfortable. Basically the barrier to entry was higher than it should have been.

When it comes to software development and especially documentation the process behind it is as important as the final result. So allow me to take a step back and describe how we currently envision this whole thing to work. As previously mentioned we feel that with documentation it is important to have a coherent piece of writing that readers can rely on. This thought lead us to the decision that a Wiki is not the best way of organizing the information the way we want to present it. We have

therefore settled for the DocBook format (http://en.wikipedia.org/wiki/Docbook) as the basis for this handbook. This decision of course entails that only a small number of people can actually actively edit and enhance the handbook. While this might seem like a limitation for some people we feel it is a necessary step in order to preserve the coherence we consider to be such an important characteristic of this handbook. We of course encourage a controlled collaboration process as 200 eyes spot more mistakes than 4 eyes and 100 brains know more than 2 brains. Additionally we are all restrained by the amount of time and energy we can invest into this project so we will never be able to provide a handbook that is 100% complete and finished. Therefore the handbook and all the information it contains is very much provided „as is" and you are free to do with it as long as you follow the rules imposed by the Creative Commons' „Attribution alone (CC-BY)" license that we are using for this project.

On a sidenote: At the end of many chapters you will find a section called „to do" which contains a list of things we still consider to be missing from the respective chapter. On the one hand this will serve as sort of a roadmap for ourselves but on the other hand we also actively encourage you to contribute missing pieces. Additionally we have provided a sort of chapter-framework that will still be relatively empty at the beginning. Again this is meant to help ourselves stay on course when writing the handbook. But again it should also enable people to start filling up the gaps by contributing (or simply suggesting) relevant documentation.

The best way to communicate with us (especially when you're interested in contributing to the activity handbook) is by sending an e-mail to handbook@olpcaustria.org. We very much value your feedback and will do our best to reply to your comments, questions, suggestions and rants as soon as possible.

Last but not least you will always find the latest version of the Activity Handbook over on http://www.olpcaustria.org/mediawiki/index.php/Activity_handbook.

Happy coding!

Christoph Derndorfer
Daniel Jahre

OLPC Austria (www.olpcaustria.org/), Vienna, May 2008

# 1 Introduction to Sugar

Chances are that if you are reading this handbook you have already spent considerable time reading about the OLPC project in general and the XO laptop and its hard- and software in more detail. Apart from the predictable question about the missing hand crank one of the first things everyone who sees an XO notes is the user interface. A user interface that is very different from what most of us are used to. No start button. No clock on the lower right corner of the screen. No virtual desktops. No list of running applications in a taskbar. No widgets. Very different indeed!

So, let us take a minute to get familiar with some of the characteristics of the Sugar desktop. The first question here is: What exactly is Sugar? In a nutshell Sugar is a GUI (short for „graphical user interface"), which is written in the Python programming language and runs on top of the X Window System and the Matchbox Window Manager (http://projects.o-hand.com/matchbox). A less technical description might sound like this:

*„Sugar is a unique approach to making computers more accessible to a broader range of people."*

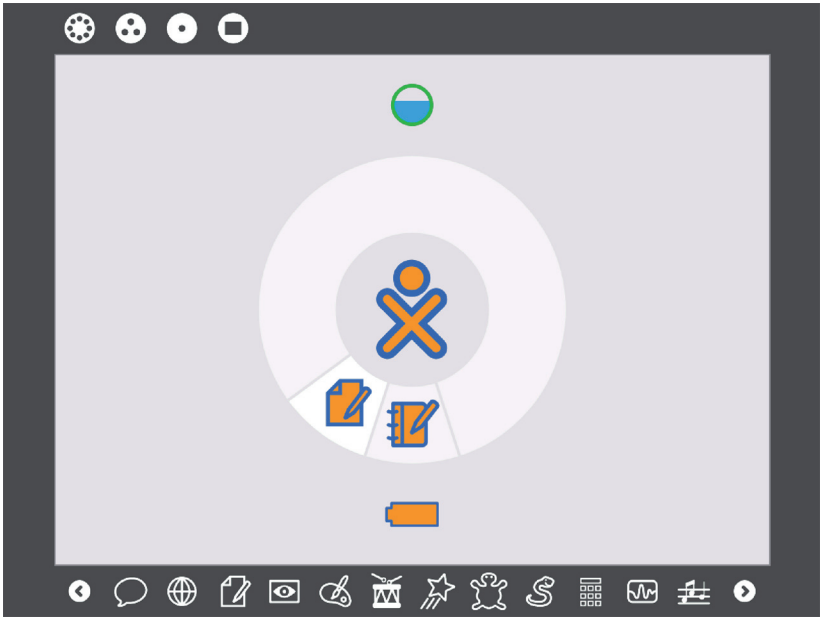
If you take a step back and think about the way we interact with computers then many words will come to mind. But few people will think of the term „intuitive". The simple fact is that our interaction with computers is highly artificial and things only really start to feel natural once you have spent a lot of time doing them a certain way. Also the names, symbols and abstractions we use are often less than self-explanatory. To give you a concrete example let us look at the way data is stored on computers. The reason why we are dealing with „files" and „folders" is quite historical in that it shows for what environment a lot of the early software was written: offices. It is not like this is the most complex thing to understand but at the same time it also is not that easy to crasp it. Especially not for a 6 year-old child living in the Peruvian Andes or Nigerian savanna. It must be possible to come up with an easier and more universal approach when it comes to how people interact with computers. This challenge sits at the very core of the OLPC human interface guidelines which is a definite must-read for anyone interested in developing software for the XO. In fact we would go as far as arguing that the article is a must-read for anyone who is interested in the challenges and requirements that lie at the heart of making computers more accessible, especially in a broader cultural context. Reading that text one realizes that in terms of innovation Sugar is one of the best examples of

the „thinking outside the box" approach that OLPC has taken throughout the whole project.

One vitally important concept that we have to explain at this point is that in the XO context there are not „applications" but rather „activities". The best explanation for why this is important can be found in the human interface guidelines document mentioned above:

„The laptop focuses children around „activities." This is more than a new naming convention; it represents an intrinsic quality of the learning experience we hope the children will have when using the laptop. Activities are distinct from applications in their foci—collaboration and expression—and their implementation—journaling and iteration."

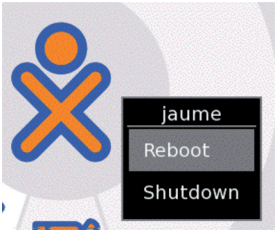
Now we are going to look at some of the Sugar elements in more detail. (Please note that these screenshots were taken with software builds 605 and 624 so things might look slightly different by the time you're reading this.)

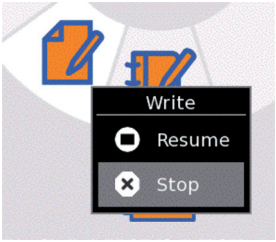


What you are looking at on the screenshot above is the so-called „home view" of

the Sugar desktop. The home view is best compared to the regular desktop that most of us look at when we start our computers. In the center of the screen you find the XO icon in the child's custom colours. The ring around it contains icons for all the activities which are currently running on the machine. Above that ring you can find the network status indicator while below it you have the battery status indicator. On the lower edge of screen you can see a scrollable list of all the activities which are installed on the laptop. The icons on the upper edge represent the 4 different views that are available in Sugar.

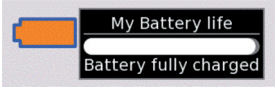
If you move across the XO icon in the middle the user's name is displayed. After a second of hovering above the icon you are presented with a menu where you can select to shutdown or reboot the machine.




Moving the cursor above an activity icon displays its name before more options pop up after another second. The default choice is of course resume so if you right-click on the icon then you will be taken directly into the activity.

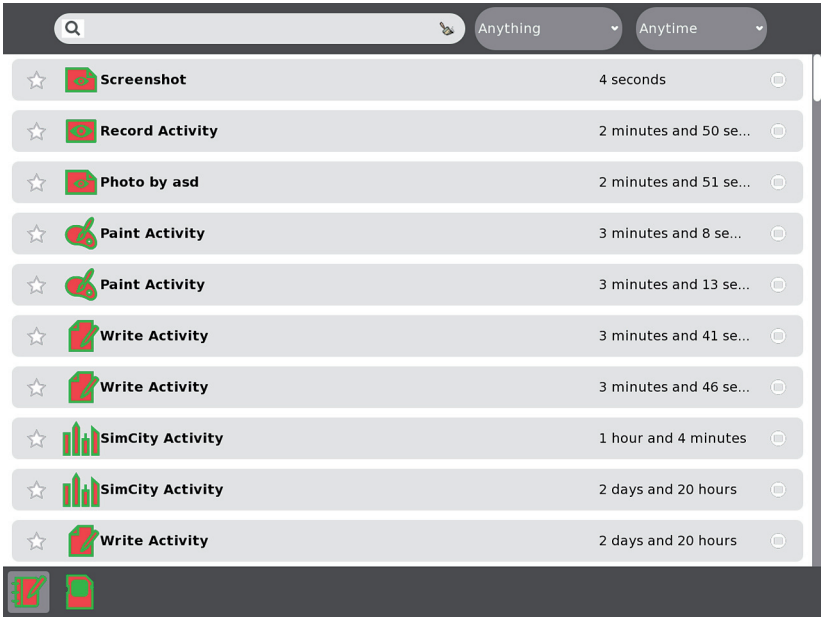

The same thing is true for the network status indicator. Moving across it reveals the SSID name of the network that you are currently connected to. After another second you get more information such as which channel is being used and of course also the option to disconnect from the network. The cool thing is that the icon also indicates the signal strength of the network connection, so if the circle is full then the signal is very good.




The battery icon also works in a very similar fashion as it roughly indicates the battery's current status and hovering over it reveals more details.

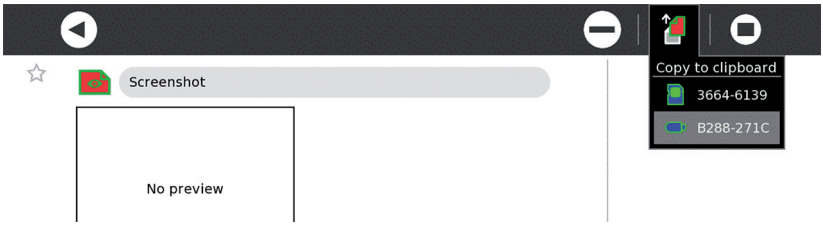
The toolbar on the bottom contains icons for all the currently installed activities plus there are scrollbuttons to gain access to more activities. The icons are pretty self-explanatory, from left to right the activities installed on this X0 are: chat, browse (internet browser), write (text processor), record (audio, video and still recording), paint, TamTamjam (one of three music / sound focused activities), Etoys (educational programming language), Turtle Art (drawing application similar to logo), Pippy (Python interpreter), calculate, measure and TamTamedit.

I am not going into details about what views are and how they work (we're going to talk more about that later). What is important to know at this point is that Sugar basically offers 4 different perspectives of the computer and you can always and easily switch between them.

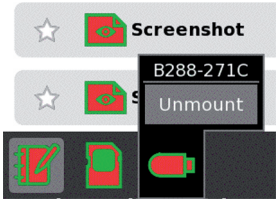



What you see on the screenshot above is one of Sugar's most important features, the so-called „Journal". The Journal is a unique and innovative way of accessing and storing data. Instead of ‚saving' and ‚opening' ‚files' the Journal allows you to

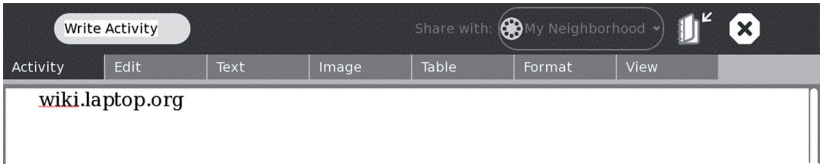
resume past activities. This again shows that the traditional seperation between „programs“ and „files“ doesn’t really exist in Sugar. I’m not going to go into more details at this point so please refer to the Journal entry on wiki.laptop.org (http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/The_Laptop_Experience/The_Journal) for further information.




A concept that has however made it into Sugar is copy-paste. Once you select a Journal entry you have the possibility to copy it to your clipboard (it then shows up on the left side of Sugar’s frame) or any removable storage device that you may have attached (in my case either an SD card or a USB thumbdrive).

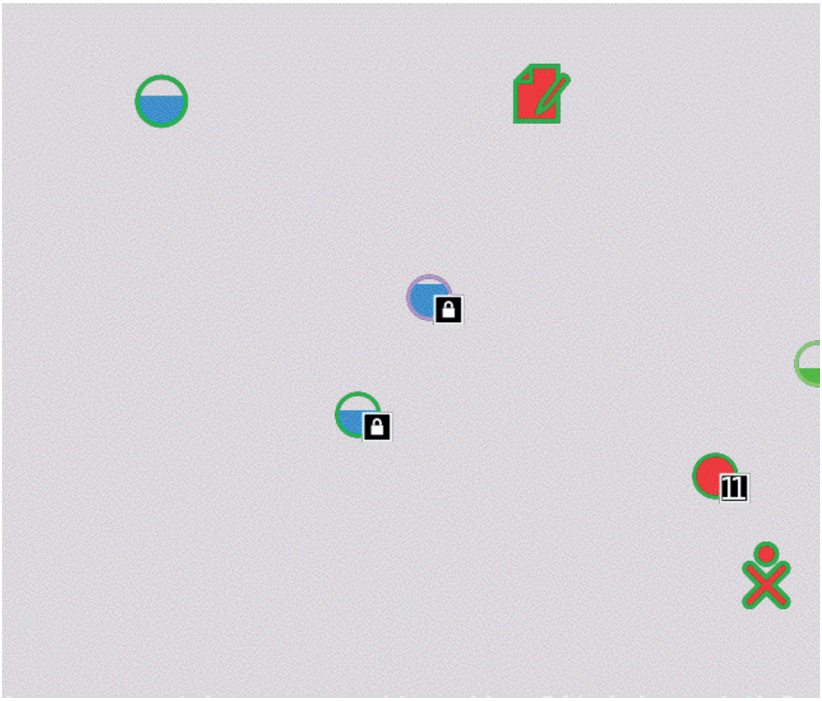



When it comes to removable media you can also easily access them within the Journal. Unmounting these devices is also as easy as hoovering over it and waiting for the „unmount“ option to appear.




As previously mentioned collaboration also sits at the heart of the OLPC project. Therefore every actitivity should allow for children to easily collaborate. Collaborating is as easy as going to the „activity“ tab within an activity and selecting „Share with: My Neighborhood“. This will lead to the activity that you’re sharing (in this

example it's write) to show up on everyone else's Neighborhood View:



Clicking on that activity-icon in their neighborhood will connect other users to you and within seconds you're able to collaborate on whatever activity you're using.

You can find some more information about how to use the OLPC XO laptop at www.laptop.org/en/laptop/start/index.shtml

Now that you have an idea about some of the core elements of Sugar let's take the next step and write our first activity, shall we?

# 2. Preparation

This chapter gives you an overview of what you need to do before you're able to dive into activity-development. We're going to look into the different options you have for development such as emulation, Live-CDs, sugar-jhbuild or on the actual XO laptops. Get, set, ready, go..

## Emulation

Currently emulating an XO is the recommended way for both demo'ing and developing software running on the XO laptops. As always there are some limitations related to emulation which can range from cosmetic details such a the wrong font-size, to the lack of audio-support to core-issues such as the obvious lack of a mesh-network and camera. Evaluating the speed of execution of an actual piece of code is also hardly possible when you're using an emulation. Regardless of what emulation you're using there are some common factors that you need to consider when setting up your environment. Probably one of the most important ones is deciding how much memory you're going to allow your emulation to use. The current OLPC XO-1 hardware comes with 256MB DDR-RAM so this is the recommended setting for a relatively close simulation of the performance you can expect on the OLPC XO. In general there are several different routes that you can take, below you can find a list of your options depending on the operating system that you're running:

### Linux

We assume that most developers running Linux will have some experience with emulation so we're not going to go into details here. The best way to emulate Sugar on a Linux machine is to use QEMU, alternatively you can also rely on solutions such as VirtualBox or Vmware. Please see the Windows XP emulation section below for more details.

### Mac OS X

Due to the fact that we don't have access to a machine running Mac OS X we can't comment on its capabilities when it comes to Sugar emulation. If you have any experience with this then please contact us at handbook@olpcaustria.org.

One of our readers reported back that he is succesfully using VMware Fusion 1.0 (build 51348) coupled with the „Joyride" OLPC vmx image. Based on his experience with build 1447 (joyride-OLPC-1447.zip at http://dev.laptop.org/pub/virtualbox/) he

said:
Everything work fine except some difficulties with the splash screen on boot and shutdown. It failed to show up. The sound don't work. I didn't have time to look at it why but must be a simple question of vmware sound drivers. Anything else works real fine. I was able to start an ssh server and connect to it without any problems. This is how I do my dev. by the way.
So it's pretty much it. The important issue here is that I recommended to use VMware for the XO Laptop only for dev and not for common use on vmware. It's still a bit „shaky" on different point but for dev. it works just fine.

### Windows XP

At the time of writing (January 2008) there are three different ways to emulate an XO environment on a Windows XP host-machine. In general emulation on Windows XP works rather painlessly however still more information and feedback is required about emulating Sugar under Windows Vista. While we have three sub-sections below there are actually two main approaches to emulation, QEMU and VirtualBox/VMware:

#### *QEMU on Windows*

| | |
|---|---|
| **Links** | http://www.h7.dion.ne.jp/~qemu-win/<br>http://wiki.laptop.org/go/QEMU |
| **Description** | We ran into some issues with QEMU on Windows which we haven't been able to solve yet. We will update this section once we have more information. |

#### *VirtualBox*

| | |
|---|---|
| **Links** | http://www.virtualbox.org/<br>http://dev.laptop.org/pub/virtualbox/<br>http://wiki.laptop.org/go/VirtualBox |

| | |
|---|---|
| **Description** | VirtualBox is a popular emulator that runs under both Linux and Windows. Unfortunately we experienced some issues when testing VirtualBox (under Windows XP) with some of the pre-converted images from dev.laptop.org/pub/virtualbox/. Build 649 or Build 655 (Ship 2) for example would go into a constant reboot cycle and never managed to succesfully load Sugar. Based on this experience we would currently recommend using VMware (see below). However we would appreciate any comments and experiences about using VirtualBox for Sugar emulation. So please e-mail us at handbook@olpcaustria.org and also post your findings on http://wiki.laptop.org/go/VirtualBox. |

### *VMware*

| | |
|---|---|
| **Links** | http://www.vmware.com/<br>http://dev.laptop.org/pub/virtualbox/<br>http://wiki.laptop.org/go/VMWare |
| **Description** | The easiest way to get started with an XO emulation is to download a pre-converted image from dev.laptop.org/pub/virtualbox/ and get the latest version of the free VMware Player (http://www.vmware.com/). After verifying that the download worked flawlessly by comparing the md5-checksum with the published md5 value you can unpack the .zip file and start the virtual-machine by double-clicking on the *.vmx file. |

With regards to the settings of the virtual-machine we're using the following configuration:

| | |
|---|---|
| **Memory** | 256MB |
| **Hard Disk** | olpc-6xx.vmdk |

| | |
|---|---|
| **Ethernet** | Device status: Connect at power on (enabled)<br>Network connection: custom (VMnet0 (default Bridged) |
| **USB Controller** | Automatically connect new USB devices to this virtual machine when it has focus. (enabled) |
| **Audio** | Device status: Connect at power on (enabled)<br>Connection: Use default host sound adapter (selected) |
| **Virtual Processors** | Processors: Number of processors (One) |

## Live-CD

At the moment there are two different Live-CDs (three if you count a do-it-yourself version) available for testing Sugar without having to install anything on your hard-drive:

### (1) LiveBackup Live-CD

| | |
|---|---|
| **Links** | http://dev.laptop.org/pub/livebackupcd/<br>ftp://rohrmoser-engineering.de/pub/XO-LiveCD/ (ftp-mirror)<br>http://skolelinux.de/XO-LiveCD/ |
| **Description** | This Live-CD was created with LiveBackup (http://livebackup.sourceforge.net/) and at publication time the latest version (XO-LiveCD_080130.iso) relies on Sugar build 689, therefore being pretty much up to date (FYI: the current stable build is 656). While we had previously suggested using the Xubuntu Live-CD (see below for details) we now recommend the LiveBackup Live-CDs due to the fact that they're a much better and up-to-date representation of the current development status. As a nice side effect they also include many more activities than previous Live-CDs. All in all it's an easy and convenient way to take a closer look at Sugar. After inserting the Live-CD and re-booting your computer the system comes up with an empty splash-screen where you have to press „Enter" to continue. The auto-configuration and boot-up takes some time but afterwards it takes you straight to the screens where you can enter your name and colour. This latest version of the Live-CD solves issues that people previously experienced when trying to use the sound in the TamTam activities and also the well-known font issues. Some activites do not scale well on screens that are smaller than the 1200 x 900 pixels that the XO uses but this is more of a minor annoyance than anything else. Please also see the XO-LiveCD_080130.pdf help document which contains a short overview of the LiveBackup Live-CD. |

### (2) Xubuntu Live-CD

| | |
|---|---|
| **Links** | http://startx.ro/sugar/ |
| **Description** | This Live-CD is built around the popular Xubuntu distribution (www.xubuntu.org) and while we had previously recommended using it, it's now a bit outdated which is the reason why we currently suggest using the LiveBackup Live-CD (see above for details). After inserting the Live-CD and re-booting your computer the system comes up with a simple splash-screen where you have to select „Start or install Xubuntu" to continue. Then you simply wait for the desktop to come up, click on the „Applications" button in the top-left corner and select „Other" -> „Sugar Emulator". Voil?Sugar starts and you can now type in your name, select a colour and explore it! Alternatively you can also logon to your Live-CD by clicking on the „Sessions" button on the login screen and selecting „2. Sugar" from the list. However this didn't work reliably at publication time. Other than that this approach has been proven to work quite well and we're currently not aware of any major issues regarding the Xubuntu Live-CD. |

### (3) Pilgrim Fedora Live-CD [DIY]

| | |
|---|---|
| **Links** | http://gregdek.livejournal.com/19843.html |
| **Description** | RedHat's Greg DeKonigsberg started a kickstart file to create a Live-CD image based on Fedora and built using the Fedora livecd-creator. We haven't tested this path up to now but in case you have please let us know how it went by e-mailing us at handbook@olpcaustria.org and adding the information to http://wiki.laptop.org/go/LiveCd. |

## sugar-jhbuild

The sugar-jhbuild skript is a python script that is using for building and running Sugar from source. It might have dependencies on your System that you have to install before you can use sugar-jhbuild. You should check the laptop.org Wiki if you have all the required packages on your Distribution or OS. The script is managed in a git repository. You need to have git installed on your system. You can download the script by typing

```
git-clone git://dev.laptop.org/sugar-jhbuild
```

Now you can change to the directory sugar-jhbuild by typing

```
cd sugar-jhbuild
```

Use git-pull to sync to the Repository. Now you are ready to use sugar-jhbuild. As a first step tell jh-build to download all the required source packages to download by using sugar-jhbuild's update command.

```
./sugar-jhbuild update
```

Use the build command to run the actual build.

```
./sugar-jhbuild build
```

sugar-jhbuild implicitly calls update on every run, so you can skip the update run in the first place but it might be better to have the huge download in one rush before start building. Sometimes some of sugar-jhbuild's builtin checks for required packages will tell you about missing requirements even if they are installed. If you are sure that you met all requirements you can switch that checks of by invoking the command as

```
./sugar-jhbuild build -x
```

The script runs interactively by default. This means if an error occurs the script will stop and asking you what to do next. If you don't like this behaviour because you don't have the time to attend the progress of the build all day then use the --no-interact switch to turn it off. Be aware that this might lead to a broken or incomplete build.

If everything built fine you can now start Sugar by typing

```
./sugar-jhbuild run
```

If you want to run multiple instance e.g. for testing collaboration you can set an environment variable like this

```
SUGAR_PROFILE=2 ./sugar-jhbuild run
```

This would run two instances of Sugar at the same time.

## Ubuntu packages

Given that the OLPX XO is built on a Fedora 7 base environment and uses many standard libraries (see a full list of software components at http://wiki.laptop.org/go/Software_components) it should theoretically be quite easy to run Sugar under a variety of different Linux distributions. Unfortunately this isn't the case at the moment even though we do expect to see a lot of progress in that area throughout the first half of 2008. For now your best bet is to use Ubuntu (see below for details) even though other distributions might work as well. Please let us know (handbook@olpcaustria.org) if you have any experience with running Sugar on other distributions so we can add it below!

### Ubuntu (7.10 Gutsy Gibbon / 8.04 Hardy Heron)

| | |
|---|---|
| **Links** | http://www.ubuntu.com<br>http://wiki.laptop.org/go/Sugar_on_Ubuntu_Linux<br>https://edge.launchpad.net/~jani/+archive |
| **Description** | If you're running Ubuntu (7.10 Gutsy Gibbon or 8.04 Hardy Heron) you can also use deb packages to get Sugar (plus several activities) directly on your system. Please follow the instructions available at http://wiki.laptop.org/go/Sugar_on_Ubuntu_Linux (please note that the wiki entry only mentions Gutsy support at the moment) and https://edge.launchpad.net/~jani/+archive which explain all the steps you need to take in order to get Sugar running on your Ubuntu system. Even with Christoph's Linux experience being somewhat limited he managed to get Sugar running on his laptop in less than 10 minutes, it's really that easy! |

## XO Laptop

At the time of writing these lines (January 2008) OLPC is working on a new way for people to request post-mass-production developer machines. Currently the number of laptops available to developers is still somewhat limited (this is subject to change though as more units are produced) but interested developers can directly apply at OLPC for an XO-1.

Please keep a close eye on
http://wiki.laptop.org/go/Developers_Program#How_to_apply_for_an_XO and
http://wiki.laptop.org/go/Contributors as more information becomes available.

> Chris from OLPC Austria has written up a nice page on how to customize and configure an XO:
> www.olpcaustria.org/mediawiki/index.php/Xo_setup_user_guide

## General notes

Regardless of which of these methods you're using to develop activities there are a couple of things which will make your life easier:

- **`rm -rf /home/olpc/Activities/YourActivity/`**: This command will remove the selected activity from the XO. After restarting Sugar (CTRL + ALT + Erase) the icon will also be gone from the frame. Very useful for situations where the activity you have just installed doesn't work.
- Pippy (http://wiki.laptop.org/go/Pippy) provides code samples and a fully interactive interpreter making it a good choice for your first coding steps in Sugar.
- Develop (http://wiki.laptop.org/go/Develop) is a more full featured Python IDE which comes with a tree view of files, a tabbed editor, a basic version control system and other useful features. However as of May 2008 the activity is still in a very early development stage and bugs can cause serious issues such as the total loss of Journal contents. Use at your own risk, you have been warned!
- When it comes to designing an icon for your activity we'd recommend

reading the following two entries on wiki.laptop.org as they will definitely save you some headaches: Making Sugar Icons (http://wiki.laptop.org/go/Making_Sugar_Icons) and Sugar-iconify (http://wiki.laptop.org/go/Sugar-iconify)

## Links

- OLPC Developer's Program
  http://wiki.laptop.org/go/Developers_Program#How_to_apply_for_an_XO
- OLPC Contributor's Program
  http://wiki.laptop.org/go/Contributors
- QEMU
  http://fabrice.bellard.free.fr/qemu/
- Virtual Box
  http://www.virtualbox.org/
- VMware
  http://www.vmware.com/
- XO Setup User Guide
  http://www.olpcaustria.org/mediawiki/index.php/Xo_setup_user_guide

# 3. Sugar Basics

This chapter covers the basics of creating a software bundle for the XO, including a short code sample („hello world", what else?) to show you all the steps you need to take in order to write an activity.

First of all it's important to explain the concept behind these activity bundles. Here's what the description from the Activity Bundles (http://wiki.laptop.org/go/Activity_Bundles) entry on wiki.laptop.org has to say:

Every activity in the Sugar environment is packaged into a self-contained „bundle". The bundle contains all the resources and executable code (other than system-provided base libraries) which the activity needs to execute. Any resources or executable code that is not provided by the base system must be packaged within the bundle.

One of the main reasons why these self-contained bundles play such a vital role within the XO ecosystem is that children should easily be able to transfer activities they don't have yet to their own laptops.

Activities are meant to be shared between children. If a child doesn't have the activity, it is automatically transfered to the child when he or she joins the shared activity. Packaging activities in self-contained bundles allows easy sharing, installation, removal, and backup.

## The package layout

Like most modern package formats the .xo package requires a specific directory layout and files that follow a certain naming scheme. The basic directory layout for our sample activity is shown below:

```
MyActivity.activity/
MyActivity.activity/MANIFEST
MyActivity.activity/MyActivity.py
MyActivity.activity/setup.py

MyActivity.activity/activity
MyActivity.activity/activity/activity.info
MyActivity.activity/activity/activity-myactivity.svg
```

```
MyActivity.activity/lib (additional libraries required
by the activity) [optional]
MyActivity.activity/locale (additional libraries
required for localization) [optional]
```

## The package files in more detail

### MANIFEST

This file contains a list of the files which are part of the Activity Bundle. (Make sure that you don't leave any blank lines at the end of the file!)
In our example the MANIFEST looks like this:

```
MyActivity.activity/HelloWorldActivity.py
MyActivity.activity/setup.py
MyActivity.activity/activity/activity.info
MyActivity.activity/activity/activity-helloworld.svg
```

### HelloWorldActivity.py

This file contains the code of our activity.

```
from sugar.activity import activity
import logging

import sys, os
import gtk

class HelloWorldActivity(activity.Activity):
    def hello(self, widget, data=None):
        logging.info('Hello World')

    def __init__(self, handle):
        print "running activity init", handle
        activity.Activity.__init__(self, handle)
        print "activity running"

        self.set_title('Hello World')
```

```
        # Creates the Toolbox. It contains the
        # Activity Toolbar, which is the bar that appears
        # on every Sugar window and contains essential
        # functionalities, such as the 'Collaborate'
        # and 'Close' buttons.
        toolbox = activity.ActivityToolbox(self)
        self.set_toolbox(toolbox)
        toolbox.show()

        # Creates a new button
        # with the label "Hello World".
        self.button = gtk.Button("Hello World")

        # When the button receives the "clicked"
        # signal, it will call the function hello()
        # passing it None as its argument.  The hello()
        # function is defined above.
        self.button.connect("clicked", self.hello, None)

        # Set the button to be our canvas. The canvas
        # is the main section of every Sugar Window.
        # It fills all the area below the toolbox.
        self.set_canvas(self.button)

        # The final step is to display this newly
        # created widget.
        self.button.show()

        print "AT END OF THE CLASS"
```

**setup.py**

```
from sugar.activity import bundlebuilder
if __name__ == "__main__": ##checks whether the class is
started as main; not absolutely necessary
       bundlebuilder.start('HelloWorld')
```

**activity.info**

The activity.info file contains all important metadata for the package similar to a .spec file in a rpm package. The fileformat is kept simple, it looks like the .ini files from dos and Windows and are also specified on freedesktop.org

```
[Activity]
name = HelloWorld
service_name = com.olpcaustria.HelloWorldActivity
activity_version = 1
host_version = 1
bundle_id = org.olpcaustria.HelloWorldActivity
icon = activity-helloworld
class = HelloWorldActivity.HelloWorldActivity
mime_types = application/pdf;image/tiff
show_launcher = yes
```

Here's a more detailed break-down of the file:

**`[Activity]:`** The activity.info file must always start with „[Activity]", no other tag is allowed in the first line of the file!
**`name:`** The name of the activity - unless defined with an additional language code in []-brackets is assumed to be „en_US".
**`activity_version:`** A single positive integer, larger values are considered to be „newer" versions.
**`host_version:`** Again a single positive integer, which specifies the version of the Sugar environment which the activity is compatible with. (While writing this book no higher versions than 1 for Sugar are available.)
**`bundle_id:`** Activity bundle identifier which is also used as the default service type when sharing the activity.
**`icon:`** Specifies the name of the activity's icon which is expected to be in the base directory and has an .svg extension.
**`class:`** Defines which class to start upon the activity initialization. (You can either use the „class" or the „exec" line below to define that class.)
**`mime_types:`** Specifies which mime types (separated by semi colons) are supported by the activity, is used when downloading files from the web or for offering the activity when resuming something from the Journal. Don't use it unless your

activity really needs and supports these file-types.
**`show_launcher:`** (optional) If this tag is missing or set to „yes" the activity's icon will be shown in Sugar's launcher panel.

### icon.svg

The activity icon has to be placed in the /activity sub-directory and should use the .SVG format (http://www.w3.org/Graphics/SVG/ - SVG is a a XML based language for two-dimensional vector graphics). Please take a look at the Sugar Icon Format (http://wiki.laptop.org/go/Sugar_Icon_Format) and the Icon section of the Human Interface Guidelines (http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/The_Sugar_Interface/Icons) for more information about icon design.

Here's a sample icon produced by Daniel Jahre's activity-bundle-generator

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

### xo bundle

All that's left to do at this point is to create the .xo bundle package. You can easily accomplish this by using a compression tool of your choice to make a zip-archive. Then you simply rename the .zip file to HelloWorld.xo and voila, you've just finished your first activity!

### Localization issues

In the locale directory are subdirectories for each locale that the activity is localized for. The localization is done in files called activity.linfo. How localization is handled will be described further down the road.

```
<!--
<section>
<title>TO-DO</title>
<para>
- add information about localization (maybe in another
chapter or section)
- add relevant links
- add Hello World .xo package
</para>
</section>
-->
```

# 4. Sugar User Interface

In this chapter we're going to explore some of the basics related to designing the interface of activities. As previously mentioned in Chapter 2 Sugar is quite a unique approach when it comes to user interfaces. The goal of project necessitates a radical rethinking in terms of how software is designed. So now we'll look at some of the more technical details involved in building toolbars for the activity interfaces. Please note that this chapter is not meant to be a GTK+ or PyGTK tutorial as this would go way beyond the scope of this handbook. Instead we're going to focus on the Sugar-specific toolbars. However you will find some pointers to more general GTK+ or PyGTK tutorials and resources in the Link section at the end of this chapter.

## Toolbox / Toolbars

The first thing you notice when you start an activity is the toolbox at the top of the screen. In its standard form it has at least an „Activity" toolbar which contains the following 4 elements:

- a label with the name of the activity
- a „Share with" combo-box that allows you to share the activity on the network
- a „keep" button (more on that later)
- a „stop" button that closes the activity

Only 3 short lines of code are necessary to create and display that toolbar:

```
toolbox = activity.ActivityToolbox(self)
self.set_toolbox(toolbox)
toolbox.show()
```


The standard toolbar (from „Write"-activity).

While collaboration sits at the very heart of the „activity"-concept there will be cases where an activity is just meant to be used by a single person. For the most part this is true with service-orientated software and looking at the current set of preinstalled activities there are some examples of this such as „Terminal" or „Log Viewer". There will also be cases where the"keep" functionality

(http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/The_Laptop_Experience/The_Journal#The_Notion_of_.22Keeping.22) (based on the Journal this basically replaces the traditional „save" and „load" concept) isn't needed.

We're going to take a closer look at the Journal in the next chapter.

As mentioned above the standard activity toolbar comes with both the „Share with" combo-box and the „keep"-button enabled. For situations where these elements aren't needed we can simply hide them by setting their „visible" property to „False". Below you can find a code-snippet that does exactly that:

```
# Loads the activity toolbar
activity_toolbar = toolbox.get_activity_toolbar()
# Hides the „Share with:" combo-box
activity_toolbar.share.props.visible = False
# Hides the keep button
activity_toolbar.keep.props.visible = False
```

The resulting toolbar will look like this:


The toolbar without the collaboration and keep buttons (from „Terminal"-activity).

The following sample is a code snippet (http://wiki.laptop.org/go/Sugar_Code_Snippets#Toolbar) that illustrates how to insert a custom toolbar (in this case a ‚help' toolbar) with a single button that is connected to the self.help_button_ pressed method.

```
from sugar.graphics.toolbutton import ToolButton
from sugar.activity import activity
...
class MyActivity(activity.Activity):
...
   mytoolbox = gtk.Toolbar()
   helpbut = ToolButton(‚help') #Stock help icon
   helpbut.set_tooltip(_(„Get help"))
```

```
helpbut.connect(‚clicked‘, self.help_button_pressed)
mytoolbox.insert(helpbut, -1)
helpbut.show()
mytoolbox.show()

toolbox = activity.ActivityToolbox(self)
toolbox.add_toolbar(„mytoolbar“,mytoolbox)
self.set_toolbox(toolbox)
toolbox.show()
```


Customized toolbar (from „Write“-activity).

## Links

- Human Interface Guidelines
  http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines
- Software Components
  http://wiki.laptop.org/go/Software_components
- GTK+ Homepage
  http://www.gtk.org/
- PyGTK Homepage
  http://www.pygtk.org/
- PyGTK 2.0 Tutorial
  http://www.moeraki.com/pygtktutorial/pygtk2tutorial/index.html

# 5. The Sugar Services for Datastore and Collaboration

This chapter covers the D-Bus based services datastore and collaboration. The datastore is used for managing actvities and its data by the journal and collaboration enables the concurrent use of Activities over the network. Example code in this chapter is in C but the python calls are quite similar.

## D-Bus - the heart of the Sugar services

D-Bus is a messaging API that is widely used in Linux based Systems e.g. for propagating hardware events like plugged in external drives. It is designed as a Server-Client architecture. It is not neccessary to know all the details of the D-Bus system because the Sugar API wraps around it. There are several wrappers called services.

The basic service is called the Sugar shell. The Sugar shell provides a clipboard, a registry of all installed activities and an object type registry to map mimetypes and activities.

The Journal keeps track of the user's actions on the laptop.

## The Activity Life Cycle

The life cycle of an Activity splits up into three parts: Startup, Operation and Shutdown. Each phases uses D-Bus services to work with the Sugar environment.

### Startup

On startup the Activity creates D-Bus services that Sugar can an activity with its window. If an object id is passed via command line the object is loaded from the datastore. If no object id is given, a new datastore object will be created. The Activity asks the Presence Service if the activity is shared and joins it in that case. On Startup some important Environment Variables are set. SUGAR_ACTIVITY_ROOT points to a directory where the activity is allowed to write into. Activities are encapsulated that way that they are not allowed to write anywhere else. SUGAR_BUNDLE_PATH points to the directory where the Activity bundle is installed too. So you can read images and other needed data that is bundles with that Activity from there.

### Operation

During operation the Activity uses the Presence Service to share itself and the data-

store to update its status so that the Activity can be continued in that state in case of a shutdow. Of course it can uses all other services as well during operation.

### Shutdown

On Shutdown the Activity saves its state to the datastore and quits.

## The datastore

The datastore is central storage service on each laptop. Its content will be managed by the Journal. It is neccessary for an Activity to write its state to the datastore so that the Activity can be managed by the Journal.

### Metadata

The Metadata is a set of properties that is stored along with the actual data into the datastore. The following list shows some of these properties.

```
,activity‘:          ,my.organization.MyActivity‘
# bundle id
,activity_id‘:       ,6f7f3acac-
ca87886332f50bdd522d805f0abbf1f‘
,title‘:             ,My new project‘
,title_set_by_user‘: ,0‘
      # ,1‘ if not default title
,keep‘:              ,0‘
      # ,1‘ if marked as „favorite“ (star)
,ctime‘:             ,1972-05-12T18:41:08‘
      # created
,mtime‘:             ,2007-06-16T03:42:33‘
      # modified
,timestamp‘:         1192715145
      # modified, in seconds since the UNIX epoch
,preview‘:           ByteArray(png file data, 300x225 px)
,icon-color‘:        ,#ff0000,#ffff00‘
      # owner buddy or shared activity color
,mime_type‘:         ,application/x-my-activity‘
,share-scope‘:
```

```
      # if shared
,buddies‘:           ,{}‘
      # buddies in a shared activity as JSON
,summary:text‘:     ,text I want to be indexed‘
# properties with key ending in „:text“ will be searched
in fulltext search
```

The API for maintaining datastore objects is quite simple. To create a datastore object call the create function. It creates a datastore object for a file. The parameter properties is a dictionary containing the metadata properties.

```
object_id = datastore.create(properties, filename)
```

If filename is not empty, the file will be copied to the datastore. The activity should delete the file once the call completes. The returned id will be a string like ‚4543af91-7be9-404e-b2f1-3e27cb15a15d‘. To update the information, use the update call like this.

```
datastore.update(object_id, properties, filename)
```

Like for the create call the file will be deleted after the call. The supply of a filename is optional for this call. It is possible to retrieve the properties and the file from the datastore. To retrieve the properties call get_properties.

```
properties = datastore.get_properties(object_id)
```

If you want to retrieve the file use the get_filename call. You should remove the file after you used it, a copy will be still there in the datastore.

```
filename = datastore.get_filename(object_id)
```

It is also possible to query the datastore with the find call. You can do an unstructured query by passing a string to find which means that the string is used to search in all properties. A second possibility is to do a structured query by passing a dictionary for the properties you are looking for to find. In both cases the call looks like this.

```
(results,count) = datastore.find(query)
```

Here are examples for some structured queries.

```
,title‘ = ,First Project‘
,mime_type‘ = [,image/png‘, ,image/jpeg‘]
,mtime‘ = {,start‘ = ,2007-07-01T00:00:00‘, ,end‘ =
,2007-08-01T00:00:00‘}
```

There are additional parameters to adjust your query to your needs.

```
,query‘: fulltext search term
,order_by‘: key (or array of keys) to order results by,
to reverse order use ,-key‘
,limit‘, ,offset‘: return only limit results starting at
offset
,mountpoints‘: array of mountpoint ids to search (or all
if not specified)
,include_files‘: if true, generate files as if get_filena-
me() had been called for each item. In results, a pro-
perty ,filename‘ will be added.
```

### Progressbar

Now we want to use the datastore to show a little progressbar for jobs that take a little bit longer. The progressbar shows up in the Journal. The Browse Activity uses this feature e.g. The datastore does not support to save files incrementally so wen need to store it in filesystem first and then adding it to the datastore. The trick is to create a metadata for the progressbar and update it accordingly. The user can cancel the operation so the key gets deleted. We have to take care to that.

```
meta = ...                                          # regular
metadata
meta[„progress“] = 0
id = datastore.create(meta, „“)                     # create
with progress bar
while (done() &gt; 100) {
    if (got_signal(datastore, „Deleted“, id))
        return user_cancelled();
    write_to(tmpfile)
    meta[,progress‘] = done()
    datastore.update(id, meta, „“)                  # update
```

```
progress bar
}
meta.deleteKey(„progress“)
datastore.update(id, meta, tmpfile.name)        # check-in
file, remove progress bar
```

### Mountpoints

Devices are represented as mountpoints in the datastore. If no mount point is specified the datastore is used. You can get the mountpoints by using the following call.

```
mounts = datastore.mounts()
```

That call returns at least three paramters in a dictionary.

- ‚id‘: the id used to refer explicitly to the mount point
- ‚title‘: Human readable identifier for the mountpoint
- ‚uri‘: The uri which triggered the mount

Mount points can be specified when creating an object (using a ‚mountpoint‘ key and id value in the properties), and when querying the datastore (by adding a ‚mountpoints‘ query option). Large files to be stored on an external device should be placed at the uri of the mount point. External Devices are mounted by the Journal and appear in /media.

### File access

All writing to the file system is restricted to subdirectories of the path given in the SUGAR_ACTIVITY_ROOT environment variable. This directory has three subdirectories with different policies.

$SUGAR_ACTIVITY_ROOT/data/ is used similar to a $HOME directory, for persistent activity data such as configuration files. Make sure files in there are group readable and writable. The directory itself is group-writable. Files stored here will survive reboots and OS upgrades.

$SUGAR_ACTIVITY_ROOT/tmp/ is used similar to a /tmp directory, being backed by RAM. It may be as small as 1 MB. This directory is deleted when the activity exits (specifically, as soon as all children of the activity‘s first process die). This directory

is only accessible to the activity and its children; not even to sugar.

$SUGAR_ACTIVITY_ROOT/instance/ is used similar to a /var/tmp directory, being backed by flash rather than by RAM. It is unique per instance. It is used for transfer to and from the datastore (see keeping and resuming). This directory is deleted when the activity exits (specifically, as soon as all children of the activity's first process die)

## Collaboration

The Collaboration features are provided by the Presence Service which is also a D-Bus service. There are are several ways of initiating a collaboration. You can start it by sharing your Avtivity over the network or by inviting a buddy or by joining an Activity that is shared by somebody else. After initiating you can manage your buddies and using so called tubes to communicate with other instances of your Activity over the network.

### Sharing

To share an Activity use the following call.

```
activity = PS.ShareActivity(activity_id, bundle_id,
name, properties)
```

The bundle id is used for the icon and to launch the same activity when someone joins it. The name will be shown in the mesh view and should generally be the same as the title of the datastore object (see above). The properties argument is not used currently and should be an empty dictionary.

Note that sharing will be private (invitation-only) by default, that is, the icon will not be visible in the mesh. To share publicly, set the ‚private' property to False:

activity.SetProperties({‚private': False})

### Inviting

If you don't want to share your Activity publicly you can invite your buddies to join you. This is a more private way of collaboration. To invite a buddy call:

```
buddy = PS.GetBuddyByPublicKey(buddy_key)
activity.Invite(buddy, message)
```

### Joining

When launching, the Presence Service must be consulted to see if this instance was shared by someone else, meaning it was launched by the user is trying to join it:

```
activity = PS.GetActivityById(activity_id)
```

This yields an error if this instance (identified by its activity id) was not shared, in which case a regular non-shared startup should be performed. Otherwise, the activity object held by the PS is returned, and this activity instance needs to join:

```
activity.Join()
```

It should continue by establishing a communication channel with the originating instance (see below).

### Leaving

To leave a shared activity (e.g. because it is closing) you need to inform the Presence Service by using this call.

```
activity.Leave()
```

### Buddies

The activity object created by either sharing the current activity or joining an existing activity is used to establish means of communication between these instances. The joined XOs can be accessed to start communicating with them.

```
buddies = activity.GetJoinedBuddies()
```

To get notified of buddies joining or leaving, listen to these signals.

```
buddies = activity.GetJoinedBuddies()
```

### Tubes

„Tubes“ are the transport medium of choice on the XO, provided by the Telepathy framework. There are „D-Bus Tubes“ allowing remote D-Bus calls, and „Stream Tubes“ which forward sockets (similar to ssh forwarding). Tubes are collected in a „Channel“, and channels are associated with a „room“, one per shared activity instance.

First, get the Telepathy connection from the shared activity object

```
(tp_service, tp_connection, channels) = activity.Get-
Channels()
```

Where tp_service and tp_connection is the D-Bus service name and object path for the Telepathy connection. An array of channels pre-created in the activity room is returned, too. There is at least a text chat and a tubes channel at

```
Service:     (tp_service)
Object path: (channels[i])
Interface:   ‚org.freedesktop.Telepathy.Channel‘
```

Use GetChannelType() to tell the channels apart

```
if (channel.GetChannelType() == ‚org.freedesktop.Telepa-
thy.Channel.Type.Tubes‘)
  ...
elseif (channel.GetChannelType() == ‚org.freedesktop.
Telepathy.Channel.Type.Text‘)
  ...
```

### Stream Tubes

A stream tube forwards a socket to a remote host (similar to ssh forwarding, but not encrypted). The activity can set up a listening socket by whatever means and then create a tube to forward it.

```
tube_id = channel.OfferStreamTube(     # sa{sv}uvuv -> u
              ‚my-activity-xttp‘,       # unique service
name
              {},                      # dict of params
              2,                       # socket type
(0=Unix, 2=IPv4, 3=IPv6)
              (‚127.0.0.1‘, 12345),     # socket address
(depends on type)
              0, 0)                    # access control
and params
```

You can forward Unix, IPv4, or IPv6 sockets. Access control is restricted to localhost

by default. New tubes are announced by a signal.

```
NewTube ( u: tube_id, u: initiator, u: type, s: service,
a{sv}: parameters, u: state )
```

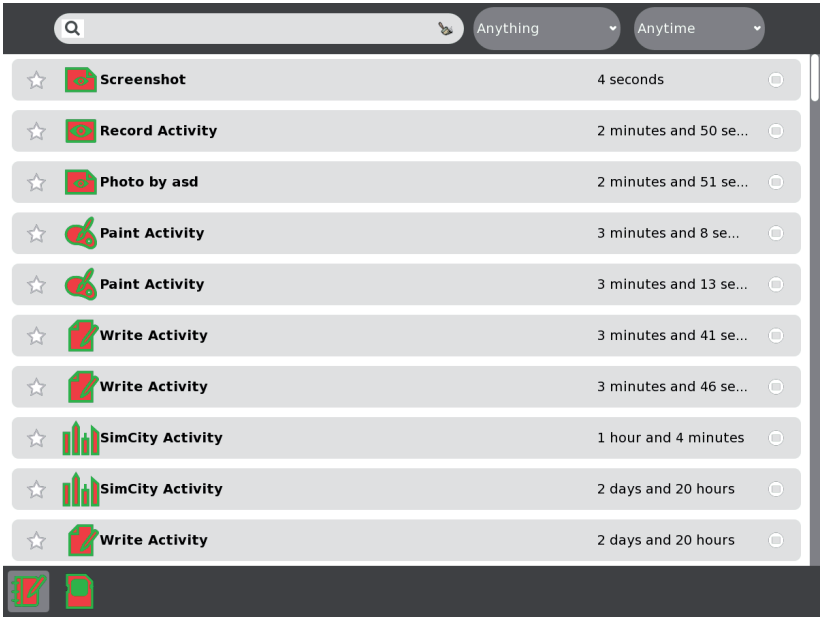
On the remote host you can connect to that tube.

```
address = channel.AcceptStreamTube(    # uuuv -> v
              tube_id,
              2,                        # socket type to
return (0=Unix, 2=IPv4, 3=IPv6)
              0, 0)                     # access control
and params
```

Which returns an address struct of the specified type, e.g., (‚127.0.0.1', 45679). Then just connect a socket to that address and you are ready to share data.

# 6 The Journal

This chapter covers one of the most important features of Sugar: the Journal. The Journal is best described in the accompanying entry (wiki.laptop.org/go/Journal) on wiki.laptop.org: „The Journal activity is an automated diary of everything a child does with his or her laptop." The Journal is very closely tied to the activity-concept that we briefly discussed in Chapter #1. Basically what it boils down to is that instead of working with files - saving and loading documents - the Journal is used to „resume" previously started activities.




As an example project we are going to write a two very simple programs that allow you to store and retrieve dummy entries in the Journal. In order to focus on the Journal functionality here we decided to skip writing a complete activity and so we developed a simple python script that can be run in either the Terminal activity, the shell or within Pippy.

## Storing Journal entries

You only need 3 lines of code in order to create a dummy Journal entry:

```
from sugar.datastore import datastore
journal_object = datastore.create()
datastore.write( journal_object )
```

Running this code will result in two things: A warning saying that „DSObject was deleted without cleaning up first. Please call DSObject.destroy() before disposing it." (more on this in a minute) and an entry in your Journal that should look like this one.


Empty Journal entry

In order to really use the Journal you will want to use a couple of additional lines of code:

- **`journal_object.metadata[,title'] = ,Journal dummy'`** Journal items support a variety of meta data such as ‚title', ‚mime_type' (so the Journal offers appropiate activities for resuming the entry), ‚tags', ‚keep' (marking it as a favorite) and ‚preview' (please refer to <ulink url="http://wiki.laptop.org/go/Low-level_Activity_API#Meta_Data">this list</ulink> for a complete overview). This meta data can be accessed with DSObject.metadata[‚meta data type']. Note: As of May 13 there still seems to be a bug (<ulink url="http://dev.laptop.org/ticket/4662">#4662</ulink>) which results in the loss of some meta data between reboots.
- **`journal_object.file_path = ,/yourDirectory/yourFile.extension'`** When you want to store a file (e.g. an image or a text that you're importing) in the Journal then you use DSObject.file_path to access the file.
- **`journal_object.destroy()`** Executing DSObject.destroy() after you have written an object to the datastore will safely disposes of the datastore object and get rid of that error message mentioned above.

Based on these buildings blocks we come up with the following little Python script:

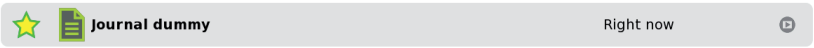
```
from sugar.datastore import datastore

title = raw_input(„Enter the name of your dummy item: „)
mime = raw_input(„Enter the item‘s mime_type (e.g.
image/png, text/plain, text/html): „)
favorite = raw_input(„Do you want to make this item a
favorite (‚0‘ for no, ‚1‘ for yes): „)
tags = raw_input(„Enter any tags you want this item to
be associated with: „)

journal_object = datastore.create()
journal_object.metadata[‚title‘] = title
journal_object.metadata[‚mime_type‘] = mime
journal_object.metadata[‚keep‘] = favorite
journal_object.metadata[‚tags‘] = ‚Dummy , + tags
journal_object.metadata[‚icon-color‘] =
‚#AFD600,#5B615C‘ #We‘re using the colors from www.olpc.
at here.
datastore.write( journal_object )

journal_object.destroy()

print „Well done. You‘ve just created your first dummy
item in the Journal.“
```




Dummy Journal entry

Another simple example that one could write is to take a vanilla .jpg or .png photo and add it to the Journal.

## Retrieving Journal entries

Obviously the next step is finding out how to retrieve data from Journal entries. When writing this handbook we often use the screenshot function on the XO (alt + 1) and then manually have to copy each of them to an attached USB thumbdrive. For this chapter we decided to write a simple script that looks for all the screenshots in the Journal and copies them into the /tmp/ directory so we can easily copy them over to a thumbdrive.


Screenshot search in the Journal

The key method for retrieving data from the Journal looks like this:

```
(results,count) = datastore.find(query)
```

Per the documentation (http://wiki.laptop.org/go/Low-level_Activity_API#Querying) a query ‚returns the results as array of properties and a count of matching items'. There are several possibilities how this query can be constructed, in its simplest form a full-text search is used:

```
datastore.find(dict(query="YourSearchString"))
```

Other possible queries are based on the datastore object's meta data such as ‚title' or ‚mime_type' (such as in the example below). A query can also be adjusted by ordering results or limiting the query to certain mountpoints. When it comes to the mountpoints the main datastore is used as a default which means that both the internal Flash memory on the XO and all connected storage devices (such as USB thumbdrives and SD cards) will be included in a search. Please refer to http://wiki.

laptop.org/go/Low-level_Activity_API#Querying for more details.

For our example we're building the query to look for png files with the titel „Screenshot". The matching Journal items will then be exported to the /tmp/ directory as .png files.

```
from sugar.datastore import datastore

import sys
import shutil

(results,count) = datastore.find(dict(title='Screenshot',
mime_type='image/png'))
i = 0
for f in results:
       i = i + 1
       src = f.get_file_path()
       dst = ,/tmp/screenshot%i' % (i, ) + ,.png'
       dict = f.get_metadata().get_dictionary()
       if dict[„mime_type"] == „image/png":
              object_id = f.object_id
              shutil.copyfile(src, dst)
              f.destroy()
```

```
[olpc@xo-05-23-5A tmp]$ ls
olpc-session-bus  screenshot2.png  screenshot4.png  screenshot6.png
screenshot1.png   screenshot3.png  screenshot5.png
[olpc@xo-05-23-5A tmp]$
```

Extracted screenshots

## Journal UI

Currently not too many activities integrate the Journal into their UI with the notable exception of browse (http://wiki.laptop.org/go/Browse). Browse brings up an overlayed Journal view whenever a user clicks on a file-selector button on a Web site. http://wiki.laptop.org/go/Low-level_Activity_API#Journal_UI contains more information about how this feature can be utilized.

## Links

- http:// wiki.laptop.org/go/Journal
- http:// wiki.laptop.org/go/Journal_Activity
- http:// wiki.laptop.org/go/Low-level_Activity_API#Datastore

# 7 Using the XO Input Devices

This chapter focuses on using the various input devices that the XO provides such as the camera, the microphone and the game buttons. We've decided to develop two sample activities which make use of these input devices: (a) a simple StopMotion activity to illustrate the use of the camera and (b) a GameButton activity that allows you to quickly test the game buttons. With regards to the microphone we didn't come up with any reasonable use-cases which aren't covered by existing activities. We will therefore update the respective section in the future once we do come up with a useful activity.

In case anyone is looking for an idea for an activity that could make use of both the camera and the game buttons then we would suggest a photo album that offers several features based on these inputs: photos can be taken with the camera (and in case you're bored you can also include those standard filters such as the ones found on all Apple MacBooks), and the game buttons can be used in the presentation-mode (aka e-book mode) to provide a simple picture-viewer.

## Camera

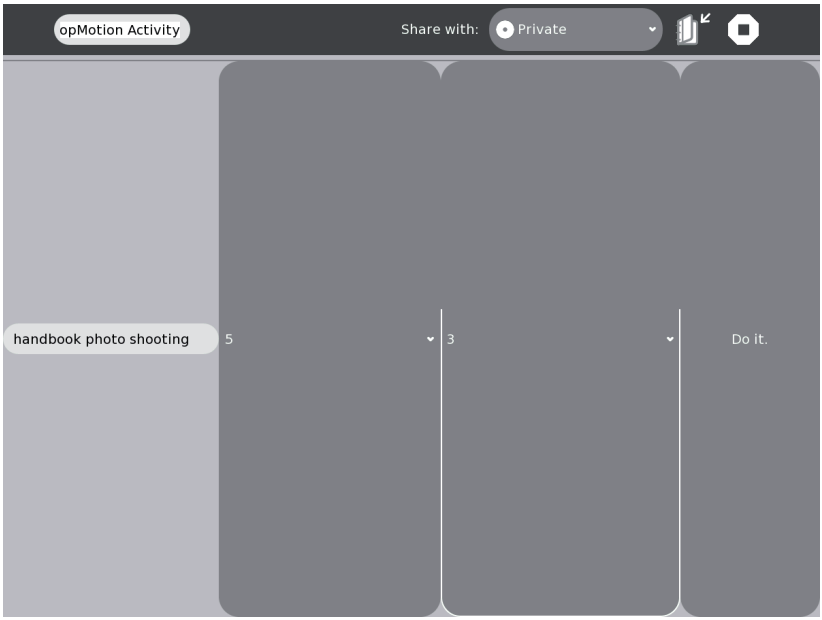
Especially when it comes to the camera there are many interesting efforts under way to use it in a variety of scenarios. One example is the headtracker (http://www.olpcaustria.org/mediawiki/index.php/Headtracker) which is an accessibility project that will allow for complete control over a laptop without using a mouse and keyboard. Another interesting approach is the Vision Processing (http://code.google.com/soc/2008/olpc/appinfo.html?csaid=9FEE89D188E4CE8A) project (part of Google Summer of Code 2008) which Nirav Patel is working on with the support of Chris Hager. Stay tuned for updates on these efforts on the various mailing-lists and OLPC related Web sites.

So while the record (http://wiki.laptop.org/go/Record) activity covers the more obvious use-cases when it comes to using the camera (capturing photos, videos and sharing them with other users) there are many other activities and scenarios which will be developed around the XO's camera functionality.

## StopMotion Activity

To demonstrate how the camera can be used we developed a simple StopMotion ac-

tivity that allows you to specify two parameters: (a) the delay between subsequent photos and (b) the number of photos you want to take. This is a relatively primitive activity but it can be relatively easily extended and adapted for other purposes.




StopMotion activity

We're going to start off with a closer look at the key command for using the camera which you can try out by entering it in the Terminal activity or the shell:

```
gst-launch-0.10 v4l2src ! ffmpegcolorspace ! jpegenc !
filesink location=/tmp/photo.jpg
```

In general the way it works is the following: The Omnivision OV7670 (http://www.ovt.com/products/part_detail.asp?id=53) used in the OLPC XO is a video4linux (http://linuxtv.org/v4lwiki/index.php/Main_Page) device which dumps its output into GStreamer (http://www.gstreamer.net/). GStreamer is basically a library which consists of a pipeline that takes the original multimedia input and converts it into files or direct screen-output. There are several keywords here which we'll be looking at in more detail:

- v4l2src: As mentioned above the XO's camera is a regular video4linux device which can be accessed via GStreamer's v4l2src source.
- ffmpegcolorspace: Converts the raw data from the camera into a color space (http://en.wikipedia.org/wiki/Color_space) that can be processed by the rest of the pipeline.
- jpegenc: This plugin takes the input from the previous step in the pipeline and converts the data into the JPEG file format. As an alternative you could also use ‚pngenc' to receive PNG files.
- filesink location=/tmp/photo.jpg: ‚filesink' writes the stream from the pipeline to a file which is defined in the location argument

When you use this functionality in Python the code will look like this:

```
pipeline = gst.parse_launch(‚v4l2src ! ffmpegcolorspace
! jpegenc ! filesink location=/tmp/photo.jpg')
```

Additionally you need to need to set the pipeline to gst.STATE_PLAYING so it processes the input. We give it one second to finish its work before we set it back to gst.STATE_NULL.

```
pipeline.set_state(gst.STATE_PLAYING)
time.sleep(1)
pipeline.set_state(gst.STATE_NULL)
```

The next step is to actually save the temporary file in the datastore so the photo is accessible via the Journal (which we covered in the previous chapter). We want to give the user the ability to choose his own title for the project which will be used as the title for all related Journal entries. In order for the Journal to offer the correct choice of activities to view the photos it is important to include the ‚mime_type' which in our case is ‚image/jpeg' but could also be ‚image/png' when using GStreamer's ‚pngenc' option. Additionally the entries will be tagged with ‚Stop motion' so it's easy to find all photos created with the activity.

```
journal_object = datastore.create()
journal_object.metadata[‚title'] = title + ‚: ‚+ str(i)
+ ‚/' + str(number)
journal_object.metadata[‚mime_type'] = ‚image/jpeg'
journal_object.metadata[‚tags'] = ‚Stop motion'
```

```
#journal_object.metadata[,icon-color‘] = ,icon-color‘
journal_object.file_path = ,/tmp/photo%i‘ % (i, ) +
,.jpg‘
datastore.write( journal_object )
journal_object.destroy()
```

The complete Python scripts then looks something like this:

```
import gst, sys, time

from random import *
from sugar.datastore import datastore

title = raw_input(„Enter the title of your project: „)
delay = input(„Enter the delay between each photo: „)
number = input(„Enter the number of photos you want to
take: „)

i=1

while (i!=number):
      photocmd = ,v4l2src ! ffmpegcolorspace ! jpegenc !
filesink location=/tmp/photo%i‘ % (i, ) + ,.jpg‘

      print i

      # take photo
      pipeline = gst.parse_launch (photocmd)
      pipeline.set_state(gst.STATE_PLAYING)
      time.sleep(1)
      pipeline.set_state(gst.STATE_NULL)

      journal_object = datastore.create()
      journal_object.metadata[,title‘] = title + ,: ,+
str(i) + ,/‘ + str(number)
      journal_object.metadata[,mime_type‘] = ,image/
jpeg‘
```

```
        journal_object.metadata[,tags‘] = ,Stop motion‘
        journal_object.file_path = ,/tmp/photo%i‘ % (i, ) +
,.jpg‘
        datastore.write( journal_object )

        journal_object.destroy()

        # delay until next shot
        time.sleep(delay)

        i = i + 1

print „The end.“
```

The complete .xo activity with the simplistic GUI shown above is available here: http://www.olpcaustria.org/mediawiki/index.php/Activity_handbook#Code_examples

## Game Buttons

The game buttons have probably been one of the least utilized features that the OLPC XO laptop has to offer. A couple of activities such as record and browse offer rudimentary support for the buttons. However most people only really use them when either running the read activity in handheld-mode or when updating their firmware.

Over the coming months this situation will most likely improve, mainly through efforts such as the Season of Usability project „Handheld-Mode Interface for the OLPC XO Laptops“ (http://season.openusability.org/index.php/projects/2008/olpc). We also believe that as activity developers in general become more more experienced with the XO and its capabilities there will be a bigger focus on utilizing its special hardware features in the future.

## Microphone

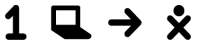
This section will be updated once we come up with a good idea for an activity.

For further information on how the microphone can be used please refer to the record (wiki (http://wiki.laptop.org/go/Record), git (http://dev.laptop.org/git?p=activities/record;a=summary)) and measure (wiki (http://wiki.laptop.org/go/Measure), git (http://dev.laptop.org/git?p=projects/acoustic-measure-activity;a=summary)) activities.

## Links

- Programming the camera
  http://wiki.laptop.org/go/Programming_the_camera
- GStreamer
  http://wiki.laptop.org/go/GStreamer
- GStreamer website
  http://www.gstreamer.net/


www.olpcaustria.org



www.laptop.org

Das Activity Handbook der OLPC (Austria) wird größtenteils
durch die Unterstutzung der ‚net culture labs' realisiert.