Maemo Diablo What is Maemo
Training Material

February 9, 2009

Contents

Chapter 1

What is Maemo

1.1 What is this thing called maemo™?

Maemo is an open source development platform for Internet Tablets. It means the collection of software that is used to develop and test software for the Internet Tablet-class devices, the first of which was the Nokia 770. It was later followed by the Nokia N800 and the Nokia N810. In this material we will be referring to all of these devices as **Internet Tablets**. Maemo is a registered trademark of Nokia Corporation.

This version of the material covers maemo SDK version 4.1.x as well as Nokia N800 and Nokia N810 Internet Tablets running OS2008.

Figure 1.1: Nokia N800

For a programmer, the Internet Tablets are really interesting as so much in them is based on free software and thus it's possible to use the same tools that are used in normal software development on other free and open source environments.

If you're coming from the Windows-world, or even the Symbian-world, this might be a new kind of encounter for you. All the tools, libraries and development processes that are used in maemo are equally used and applied in the desktop application arena, as well as for building server software. This is in part due to the GNU project (gnu.org), which has implemented a lot of the tools infrastructure in a highly portable way. The main graphical interface libraries come from the GNOME project (gnome.org), which is one of the most popular graphical environments used in Linux distributions.

By reusing existing portable and tested tools, we gain in an accelerated application development time. This also means that we can take the tool-set and apply it for writing software for embedded systems.

1.2 Internet Tablet overview

The devices are smaller than a laptop, larger than a PDA, and quite light-weight. Some of them (Nokia N810) have a small keyboard, and all of them have a stylus and a touch-sensitive screen. The stylus-driven GUI will cause some design challenges later on, since your software will need to be designed this in mind. There is also a possibility of using an on-screen keyboard with the stylus and this includes a hand-writing recognition and a predictive input system to aid the user. In all devices, there is a limited set of hardware buttons available for applications.

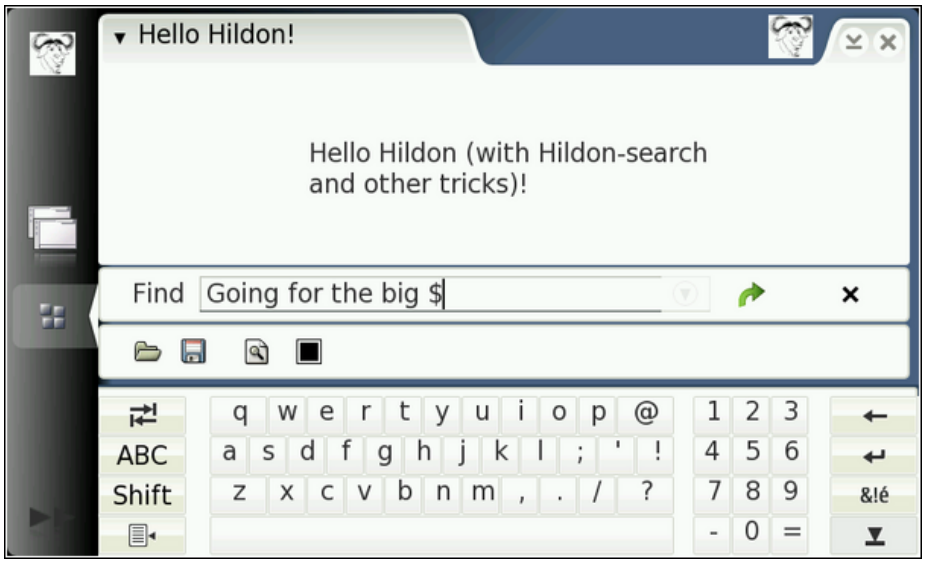


Figure 1.2: The Virtual Keyboard (VKB)

As programmers appreciate knowing a bit about the fundamentals of the

devices for which they program, table 1.1 presents a short list of the most important components.

<table>
<tr><td>N800</td><td>N810</td><td>N810 WiMax Edition</td></tr>
<tr><td colspan="3">an 800x480 pixel, 225 pixels-per-inch (PPI) wide-screen touch screen display with 16-bits per pixel color depth</td></tr>
<tr><td colspan="3">Hardware buttons with a layout optimised for Web surfing</td></tr>
<tr><td>Virtual Keyboard</td><td colspan="2">Small slide-out keyboard (& VKB)</td></tr>
<tr><td colspan="3">Wi-Fi (802.11b/g)</td></tr>
<tr><td colspan="2"></td><td>WiMAX 802.16e / 2.5GHz</td></tr>
<tr><td>External GPS via Bluetooth</td><td colspan="2">Integrated GPS (external also supported)</td></tr>
<tr><td colspan="3">1500 mAh battery</td></tr>
<tr><td colspan="3">3.5 mm stereo audio out socket (works also as mic input on N800/N810)</td></tr>
<tr><td colspan="3">Built-in VGA resolution webcam</td></tr>
<tr><td colspan="3">USB 2.0 port (in target mode by default)</td></tr>
<tr><td colspan="3">128 MiB of RAM</td></tr>
<tr><td colspan="3">256 MiB flash memory with JFFS2 filesystem</td></tr>
<tr><td>Two memory card slots, SD, MicroSD, MiniSD, MMC, and RS-MMC (some types with extender).</td><td colspan="2">One memory card slot, compatible with MiniSD and MicroSD (with extender).</td></tr>
<tr><td colspan="3">Bluetooth 2.0</td></tr>
<tr><td colspan="3">TI OMAP 2420 multi-core processor with maximum clock frequency of 400 MHz, with:
TMS320C55x DSP logic (Backward compatibility with the 54x-series)
ARM1136 core ("ARMv6") with an MMU (Backward compatibility with ARM926)</td></tr>
</table>

Table 1.1: Internet Tablet components

The USB port normally acts as a USB target, although the direction can be reversed, and the device can be the USB host (i.e. initiator). The port is not capable of providing USB power, so an external power feed is necessary. This allows various usage scenarios, when the R&D mode is enabled on a device. The default version of Internet Tablet software runs in target mode only.

Some noteworthy points about the hardware and software:

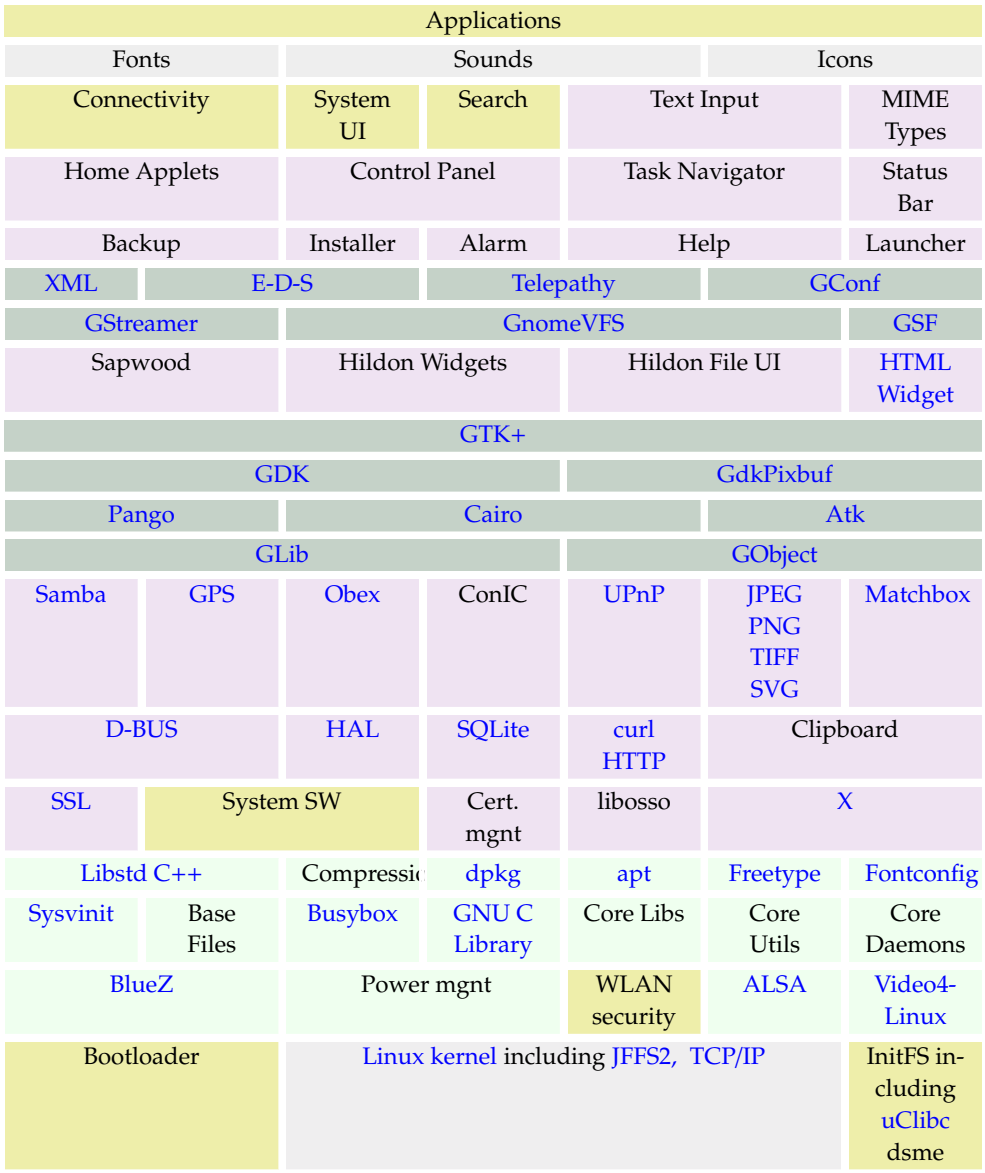
- There is not a lot of RAM (compared to a "PC"), and the memory is shared between all the applications that are executing at any given time.
- The system runs a modified Linux kernel 2.6 (omap-port).
- The system library is GNU libc 2, meaning that most software can be ported without too much effort (even networking software).

- To conserve battery power, one needs to be careful with application core logic (loops, delays, timeouts, threads etc.)
- There is no hardware acceleration for graphics operations (2D or 3D).
- The built-in flash contains approximately 64 MiB of shipped software. This means that about 192 MiB is available to be shared between applications.
- The built-in flash uses a filesystem specifically designed for flash memory, and contains transparent compression and decompression. This means that sometimes optimising for space requirements is not sensible. Compressing an image as a **.gif** is not very good idea, as it would have been compressed anyhow. However, the RS-MMC card uses FAT/VFAT filesystem. The compression rates may vary, and if space conservation is important for an application, it is advisable to test the specific use scenario properly.
- There is some support for Java acceleration in the ARM core, but this is not utilised, since there is no supported JVM to execute Java code.

N.B. The above feature list holds for the "end user" version of the software that is shipped with Internet Tablets.

1.3 Maemo runtime environment

Below is a table of the software "stack" for the maemo platform:



We'll start from the bottom layer and go upwards by covering the services:

Linux 2.6 kernel Processes hardware events, system-wide memory allocation, process creation and everything that you would expect from a modern multi-tasking UNIX-like kernel. Not covered in this material.

X Server A program that implements access to the graphics hardware and converts HID (human interface device) events from the kernel into events for the X server's clients. Explained shortly.

D-Bus A service that allows related processes to pass events to each other. The service runs as a daemon, which is a process that runs in the background.

The D-Bus daemon also passes important events from the core system to applications (e.g., "battery low"). Interfacing with D-Bus is an important part of integrating your application with the runtime environment. D-Bus was developed to provide a message bus for Linux desktop applications, the D comes from "Desktop". In fact, normally one would have at least two daemons, one that processes and sends system level events and one to allow related processes to communicate with each other inside one user's graphical session. D-Bus is more thoroughly covered in the "maemo Platform Development" material.

X window manager (customised Matchbox) Controls where the graphical applications' windows will be placed.

Task navigator Graphical program that is used to switch between applications. Always running, even if your application will be full-screen (Task navigator will be invisible in this case). Appears on left side of the screen when applications are not running in full-screen mode.

Home/Desktop A graphical program that implements a user-selectable background picture. Also provides space for applets which are small programs that draw on the background some useful (or not) information to the user. Applets are not covered in this material.

Status bar Implements the top-right area of the screen that holds the various plug-ins that indicate status and allow the user to easily change settings. Together with Task navigator and Home/Desktop, implements the screen that the user will see when the device has started.

Sapwood A daemon that caches images used to implement the overall graphical look and feel for applications designed for maemo. Used in the background by the GTK+ library.

Control panel A simple application for most system configuration tasks. It is possible to write your own Control panel plug-ins by making them dynamically loadable objects which the Control panel will load on demand. Not covered in this material.

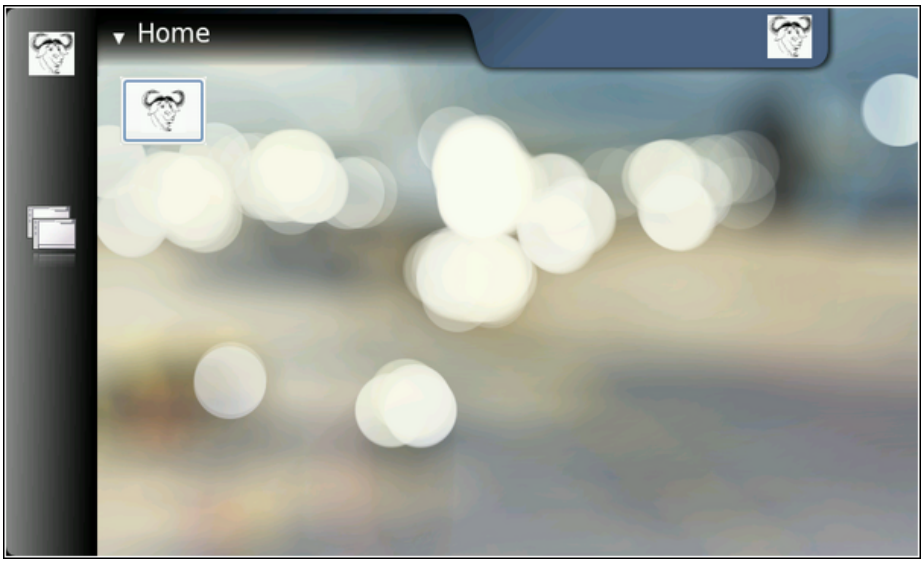


Figure 1.3: Task navigator, Home and Status bar. Also shown are the icons (the GNU heads) of the hello-world-app package.

1.4 X Window System

This is a short and simplified introduction to the X Window System. It is covered here because it is the underlying system by which graphics and user interaction is implemented in both the maemo platform and the Internet Tablets.

The X Window System is an architecture independent client/server system that allows multiple programs to interact with a user via a graphical (pixel-based) screen, keyboard and a pointing device (traditionally a mouse).

The program that wants to display something to the user, and read input from the user is called the **X client**. Each X client connects to one X server which will perform the requested graphics operations and will relay keyboard and pointer events back to the client.

When speaking about clients and servers, it's easy to make the mistake of reversing the meaning of client and server. It helps to think about the roles from the standpoint of the application, not the user. When the client starts, it will connect to some X server to create a window. A window is a rectangular area into which the client can draw. Note that the client can ask the server to position the window at some screen location, but normally doesn't. There is a special kind of client that will handle the placement of the windows of all other clients. This client is called the **window manager**. The window manager usually draws some graphical elements around each client's window, so that the user can more easily tell the boundaries between the windows. It also handles all HID-events in the window decoration areas, implements window minimising, closing, etc. The HID-events that occur within the client area of the window are passed to the client.

There are a lot of different window managers, but most work in similar ways. The "Desktop" (whatever the word means inside a computer) is normally

implemented by yet another client. And yes, the taskbar that you might see is yet another client. Even the screen saver is a separate client. In the real world there are some exceptions to the above arrangement, but having separate clients for all the elements is the most common case.

The protocol that clients use with the server is called X11. It is stream-based and bi-directional (for obvious reasons).

Clients can commonly connect to the server in two ways:

- By connecting to an IP address / TCP port on which the server is listening.
- By connecting locally using a UNIX domain socket. A UNIX domain socket is similar to TCP, but without the network in between, and the client will find the server using a name in the filesystem (note that this name does not correspond to a "regular" file).

How does the client know where to connect? By using an environmental variable called `DISPLAY`. There are only a handful of applications that know how to implement the X11-protocol, because it's quite complicated to encode and decode. Normally clients will use a library called `Xlib`, which was developed for this purpose. Xlib also contains the logic to read the `DISPLAY`-variable and will get the address to connect to from the contents of the variable. It is also possible to tell the client to use a specific display via a command line parameter (`--display=`). The parameter will be processed internally by Xlib and override the environmental variable (if any).

The content of the `DISPLAY`-variable consists of two parts:

Hostname a text field that contains a name that will go through a `gethostbyname` library call. In practise this is most often a DNS-name or an IP-address of the server, but depending on local NSS (Name Service Switch) be something else as well. This material will assume that DNS or IP will be used.

Display/Screen-pair number of the X server instance, and a number of a screen within that instance. Normally 0 is the only X server instance you have and it will number its screens starting from 0. You may also omit the screen number and 0 will be used by default.

So, for example: `DISPLAY=remote.machine.com:0.0` would mean the first screen on the first X server running on `remote.machine.com`. When starting an X server, you normally can tell it which screen to create and control.

Wait a moment! What about the "similar to TCP but not quite" UNIX domain socket? Xlib will connect using a system specific filesystem path when the hostname-portion is empty. You can try it out on your Linux desktop like this: type `echo $DISPLAY` in a terminal emulator. Your graphical terminal emulator will connect to your X server knowing the `DISPLAY`-variable. It most probably is `:0.0` unless you have a more complicated system (e.g., split dual-head).

To instruct an X client to connect to another X server, it's then necessary to modify the environmental variable: `export DISPLAY=:2.0` for example. Then start your X client and it will at least try to connect to the X server specified. In the example above the server is running on the same system as the client (the hostname part is empty). Since the screen part is optional, you may also use `export DISPLAY=:2` .

You might be wondering what all of this has to do with maemo, but you'll see in a short while when we install the environment and start testing the applications.

In the case that you skipped the intro, this is a good point to remind you that X11 is architecture independent. This means that applications running on Internet Tablets (ARM-binaries) can connect to an X server running on a x86/PC Linux (or even to an X server running on Apple OS X, Windows or other operating systems).

As a side note, the Internet Tablet has an X server as well. It runs as `:0.0`. It is a special version of an X server that requires less memory and has been configured to support most of the extensions used on the Linux desktop. The version on Internet Tablet is based on the `Kdrive` version of the X.Org X server (yes, so many versions). However, your Linux desktop is running a regular X.Org server.

Also note that by default most modern Linux distributions ship with the X server not listening for network connections. They will only accept local connections through the UNIX domain socket (`/tmp/.X11-unix/X0` where `0` is server screen number).

For more information on X, please see the X.Org-project pages (x.org) and rahul.net/kenton/xsites.framed.html. Also, ssh can be used to tunnel X11 connections securely over networks. Please see X Over SSH2 Tutorial for examples.

1.5 Typical maemo GUI application

We next take a look at the components making up a typical GUI application developed for maemo (starting from the bottom):

C library Implements wrappers around the system calls to the kernel and a lot of other useful stuff. However, the libraries presented below also provide their own APIs to similar functions, so you should always check whether you can use them directly, and avoid doing POSIX-level and system-level calls when possible. This will make your application easier to debug and in some cases easier understand. This library is used (indirectly at least) by every application running on any Linux-based system. Not really covered in this material since most of the things that we need are in the higher-level libraries.

Xlib A library that allows an application to send graphics-related commands to the X server and receive HID events from the server. Normally an application wouldn't use Xlib API directly but would instead use some easier toolkit which in turn will use Xlib. Not covered in this material other than the introduction and on a "need to know"-basis.

GLib An utility library that provides portable types, an object oriented framework (GObject/GType), a general event mechanism (sometimes referred to as GSignal), common abstract data structure types like hash tables, linked lists, etc.

GDK A library that abstracts the Xlib and provides a lot of convenience code to implement most common graphical operations. Used by an application

which wants to implement drawing directly, for example in order to implement custom widgets. In theory, GDK is meant to be graphics system independent and mostly is. Complete abstraction however is not yet complete, but for us the original Xlib target will be enough. GDK Uses GLib internally.

Pango A portable library designed to implement correct and flexible text layout for various cultures around the world. This is necessary to support the different ways that people read and write text, since it's not always from top-to-bottom and left-to-right. Uses GLib and GDK. Used by GTK+ for all displayed text. Covered only where necessary in this material.

ATK The Accessibility ToolKit. Provides generic methods by which an application can support people with special needs with respect to using computers. Not covered in this material.

GTK+ A library that provides a portable set of graphical elements, graphical area layout capabilities and interaction functions for applications. Graphical elements in GTK+ are called widgets. GTK+ also supports the notion of themes, which are user switchable sets of graphics and behaviour models. These are called skins in some other systems. Uses GLib, GDK, Pango and ATK.

Hildon A library containing widgets and themes designed specifically for maemo. This is necessary since the screen has very high PPI (compared to "PCs") and applications are sometimes controlled via a stylus. Uses all of the libraries above.

Other support libraries of interest:

GConf A library from the GNOME-project that allows applications to store and retrieve their settings in a consistent manner (in a way, similar to the registry in Windows). Uses GLib. Basic operations are covered in this material.

GnomeVFS A library that provides a coherent set of file access functions and implements those functions using plug-in libraries for different kinds of files and devices. Allows the application to ignore the semantics of implementations between different kind of devices and services. By using GnomeVFS, an application doesn't need to care whether it will read a file coming from a web server (URLs are supported), or from within an compressed file archive (`.zip`, `.rpm`, `.tar.gz`, etc.) or a memory card. Modeled to follow POSIX-style file and directory operations. Basic operations are covered in this material.

GDK-Pixbuf A library that implements various graphical bitmap formats and also alpha-channeled blending operations using 32-bit pixels (RGBA). The Application Framework uses pixbufs to implement the shadows and background picture scaling when necessary. Uses GLib and GDK.

LibOSSO A library specific to the maemo platform that allows an application to connect to D-Bus in a simple and consistent manner.

Also provides an application state serialisation mechanism. This mechanism can be used by an application to store its state so that it can continue from the exact point in time when user switched to another application. Useful to conserve battery life on portable devices. Only basic parts of LibOSSO are covered in this material and more advanced use is covered in "maemo Platform Development" material.

Whew, that was quite a list. As you can imagine, an introductory material like this cannot even try to cover all the possibilities or the API functions available in these libraries. We will try to do our best describing the bare minimum in order for you to start writing applications for maemo.

There are some caveats related to API changes between major GTK+ versions, which will be mentioned in the text, so do not go and copy-paste existing code blindly. Also this is a good place to remind you that most of the source code that you can find easily is covered by either the GPL or LGPL licenses. This material tries to stay away from legalese, but to put it bluntly, **you cannot copy GPL-ed source code into your proprietary projects (i.e., closed source). This is against the license. GPL considers static linking as distribution (copying) too. Even linking dynamically from proprietary code against GPL-ed libraries might be interpreted as prohibited by the license!**

Note that it is normally allowable to read and learn from GPL-ed source code. Indeed, unless you're copying line by line from existing code base, this can be an invaluable tool to learn how things are done in the "real world". This is quite the opposite when reading proprietary source code as there is a risk of learning something that the code owner will consider "intellectual property". If you then use this "knowledge" in a free or open source project, you risk polluting that project with unnecessary legalese.

As a side note, you might be interested to learn that almost all of the GNOME-libraries are released under the LGPL-license. This means that you can create proprietary software which will link into LGPL-libraries dynamically at runtime. If you however modify the LGPL-library, the modifications must be available in source form to the entities you distribute the binaries to under the original license (LGPL).

1.6 Battery Doesn't Last Forever!

Low power consumption is one of main hardware design goals with mobile devices because of the limited electric charge in their power supplies. If the hardware is designed correctly, it may itself contain logic and rules to enter different power saving states. To enter these power saving states the hardware requires that there is no activity in the system, in other words, there is no task ready to be run by the OS kernel scheduling mechanism. Even if power saving functionality is implemented in the hardware, activating it might not always be possible. If the applications running on the hardware are "misbehaving", then the system will be active all of the time and this makes it impossible for the power saving features to be activated at hardware level. Some of these power saving features include: changing the clock frequency dynamically, supporting multiple operating voltages and switching integrated peripherals' sleep modes.

Different parts of the hardware will require different amounts of power to run. The following pie diagram is not based on any real measurements but roughly shows how power consumption is distributed between different subsystems in a device:

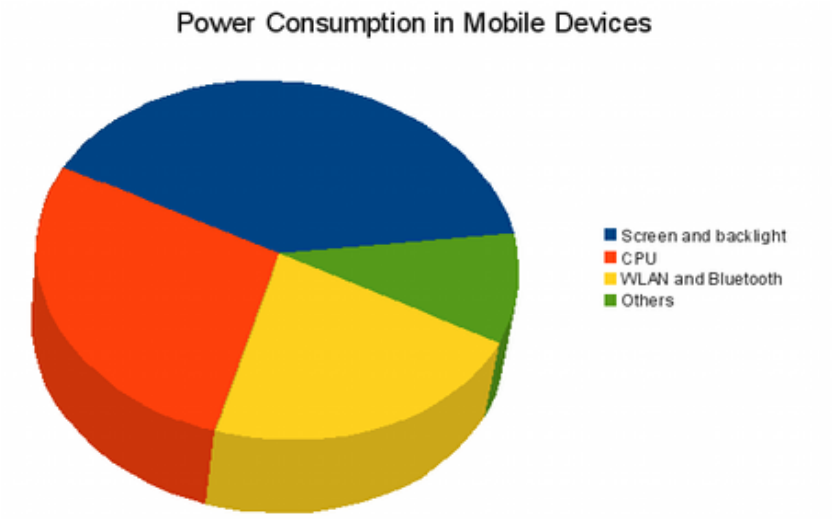


Figure 1.4: Diagram of power consumption in Mobile Devices

Keep the previous diagram in mind when you consider your application's "electricity needs". For example, in a GPS mapping application, the GUI interface is a graphical map with the position and other relevant information. If such display would keep the backlight active all the time, it would cause excessive power usage. In this case, it might be prudent to ask the user whether they really want to keep the backlight on, or provide a user-controllable button for switching the backlight on/off.

When designing for mobile devices, it's important that you (as the application designer) consider different approaches to problems. You should also consider system-wide power usage, i.e., how your application will change the power performance of the device. The above diagram might seem rather obvious, but it is often too tempting to start premature optimisation of memory usage or code speed, and forget about power optimisation. In mobile environments, all three are important.

1.7 maemo development resources

maemo has its own website (at maemo.org), which includes:

- Links to mailing lists (and their archives)
- Link to the defect tracking system (bugzilla) at bugs.maemo.org
- White papers and tutorials

- Licenses used
- Trademark usage guidelines
- Development news
- Download information
- API references library versions used in maemo SDK
- The maemo wiki and other community supported resources (garage)

There is also an IRC channel for developers (#maemo@FreeNode). You can find people related to Scratchbox and maemo hanging there lot of times. Scratchbox also has its own channel.

Note that everything you discuss on the IRC channel and mailing lists (as well as bugs you post using bugzilla) is *public information*. You might want to ask someone whether your working environment has a policy on using public resources before using them.

1.8 Other programming interfaces

Using the previously presented software library stack is not the only possibility. The platform also includes SDL (Simple DirectMedia Layer) which is a library that was originally developed by Loki Entertainment, a company that specialised in creating Linux-versions of popular commercial games. It contains most of the code necessary to write games and implement low-level graphics. SDL is not covered by this material, but you can find the API (and other) documentation at libsdl.org .

If you like to work with APIs that are hard to understand, you can use the Xlib-interface and implement your interactive program directly with it (not recommended for the faint hearted).

Using either SDL or Xlib is not directly supported by the environment and will lead to applications which will not conform to the "look and feel" common to applications designed for maemo. Such problems should be avoided so that end-users do not get confused (they would expect an uniform interface for all the applications they use). You won't be able to utilise the virtual keyboard or handwriting recognition in your application either. For some full-screen pointer driven games this might not be a big issue, but for "normal" GUI applications that the user will enter data and text, it is an issue.

It is also be possible to use the Python programming language to write GUI programs (there are bindings for the Hildon toolkit as well) and there are community driven projects for using Ruby and Vala as well as others.