# Maemo Diablo Technology Overview

Training Material

for maemo 4.1

February 9, 2009

# Contents

# Preface

## Legal notice

## Disclaimer

## Licenses

# Chapter 1

# Introduction

## 1.1 Introduction to Maemo Technology Overview

Internet Tablets made by Nokia run on top of the maemo$^{TM}$platform. This material gives you an overview of the maemo platform architecture and shows what are the components and their functions inside the platform that runs on the Internet Tablet. Maemo is based on a Debian Linux, so it is quite logical that the material covers some basics of the generic Linux architecture also. This material does not include code examples, but some basic knowledge of programming is necessary to understand the concepts described within material.

Target audience: software developers who are planning to develop applications or services on top of the maemo platform.

Prerequisites: Basic knowledge on C or C++ programming, general operating system concepts (especially Linux), open source licenses and related IPR issues.

This version of the material covers maemo SDK version 4.x.

More information about the maemo training material is available from maemo training wiki pages http://wiki.maemo.org/Maemo$_T$*raining* maintained by maemo community. Notice that the information in maemo wiki is not verified by Nokia and thus Nokia cannot be responsible of that information.

# Chapter 2

# List of Terminology

## 2.1 Terminology and definitions

**ABI** Application Binary Interface provides object code level compatibility.

**ALSA** Advanced Linux Sound Architecture. Linux kernel component intended to replace the original Open Sound System (OSS) for providing device drivers for sound cards.

**API** Application Programming Interface provides source code level compatibility.

**applet** A small application that integrates to Hildon Desktop.

**ARMEL** A name that e.g. Debian uses for the little endian ARM EABI (ABI for the ARM architecture).

**Bluetooth** An open specification for seamless wireless short-range communications of data and voice between both mobile and stationary devices.

**BT** Bluetooth.

**cURL** cURL is a command line tool for transferring files with URL syntax.

**devkit** Part of the maemo SDK that contains software development tools. The SDK contains multiple devkits e.g. doctools devkit.

**ESD** Enlightened Sound Daemon. This program is designed to mix together several digitized audio streams for playback by a single device.

**GPL** GNU General Public License. A software license that provides a high degree of freedom in a collaborative software development effort.

**GStreamer** A cross-platform multimedia framework that serves a host of multimedia applications, such as video editors, streaming media broadcasters, and media players.

**GTK+ (GUI ToolKit+)** A library of object-oriented graphical interface elements (widgets) for developing X Window applications.

**GUI** Graphical User Interface. A graphical presentation of interface which allows user to interact with computers.

**Hildon** Application framework used in the maemo platform. Developed by Nokia and based on GNOME/GTK+ technologies, currently in the process of becoming an upstream project in gnome.org.

**Hildon Desktop** The main user interface component of the maemo release Chinook, rewrite of maemo desktop.

**Internet Tablet** Product category for Internet optimized mobile devices with touchscreen. The term was coined by Nokia but is being used more widely to include other devices.

**initfs** Initial file system used as the root file system during Linux kernel boot e.g. for hardware initialization (contains kernel modules and utilities for initializing them). Mounted after boot to /mnt/initfs.

**LGPL** GNU Lesser General Public License. A compromise between the strong-copyleft GNU General Public License and permissive licenses such as the BSD licenses and the MIT License.

**Linux** Strictly speaking, Linux is the kernel of a Unix-like operating system, though the word is more commonly used to describe the the whole Linux operating system, consisting of a kernel, application programs and utilities.

**maemo** Software platform for mobile devices developed by Nokia, based on GNU/Linux and GNOME/GTK+ technologies. It includes proprietary components to make it work on the Nokia Internet Tablets.

**maemo.org** Developer community web site maintained by Nokia, main point of reference for open source and third party developers in general.

**maemo desktop** version of main user interface component of the maemo release Bora.

**maemo-af-desktop** Same as maemo desktop.

**maemo SDK** Software Development Kit to create and port applications to the maemo platform using a PC.

**Nokia Internet Tablet OS** maemo platform + proprietary applications packaged to an official device image provided by Nokia.

**OpenSSL** OpenSSL is an open source implementation of the SSL and TLS protocols.

**OSSO** Open Source Software Operations, Nokia organization developing and integrating software for Internet Tablets.

**rootfs** Root file system on the device.

**rootstrap** Part of the SDK that contains selected software components from rootfs. Rootstrap is the root file system of a target inside Scratchbox.

**Sardine** An experimental distribution based on Hildon for maemo, primarily of interest for developers who wish to test "bleeding edge" features that are being developed for future releases of maemo.

**SSL** The Secure Sockets Layer. Commonly-used protocol for managing the security of a message transmission on the Internet.

**toolchain** Part of the SDK that contains ARM cross compilation tools like compiler and linker.

**TLS** Transport Layer Security. Internet Standard similar to SSL.

**Widget** Element of a graphical user interface (GUI) that displays information or provides a way for a user to interact with the application. Examples of widgets: buttons, menus, scrollbars, forms, etc.

# Chapter 3

# The Linux System Model

Linux is a free, multi-threading, multiuser operating system that has been ported to several different platforms and processor architectures. This chapter gives an overview of the common System Model of Linux, which is also a base for the maemo platform. The concepts described in this chapter include kernel, processes, memory, filesystem, libraries and linking.

## 3.1 The Kernel

The kernel is the very heart of a Linux system. It controls the resources, memory, schedules processes and their access to CPU and is responsible of communication between software and hardware components. The kernel provides the lowest-level abstraction layer for the resources like memory, CPU and I/O devices. Applications that want to perform any function with these resources communicate with the kernel using `system calls`. `System calls` are generic functions (such as `write`) and they will handle the actual work with different devices, process management and memory management. The advantage in `system calls` is that the actual call stays the same regardless of the device or resource being used. Porting the software for different versions of the operating system also becomes easier when the `system calls` are persistent between versions.

Kernel memory protection divides the virtual memory into `kernel space` and `user space`. `Kernel space` is reserved for the kernel, its' extensions and the device drivers. `User space` is the area in memory where all the user mode applications work. User mode application can access hardware devices, virtual memory, file management and other kernel services running in `kernel space` only by using `system calls`. There are over 100 `system calls` in Linux, documentation for those can be found from the Linux kernel system calls manual pages (`man 2 syscalls`).

Kernel architecture, where the kernel is run in `kernel space` in supervisor mode and provides `system calls` to implement the operating system services is called `monolithic` kernel. Most modern monolithic kernels, such as Linux, provides a way to dynamically load and unload executable kernel modules at runtime. The modules allow extending the kernel's capabilities (for example adding a device driver) as required without rebooting or rebuilding the whole kernel image. In contrast, microkernel is an architecture where device drivers and other code are loaded and executed on demand and are not necessarily always in memory.

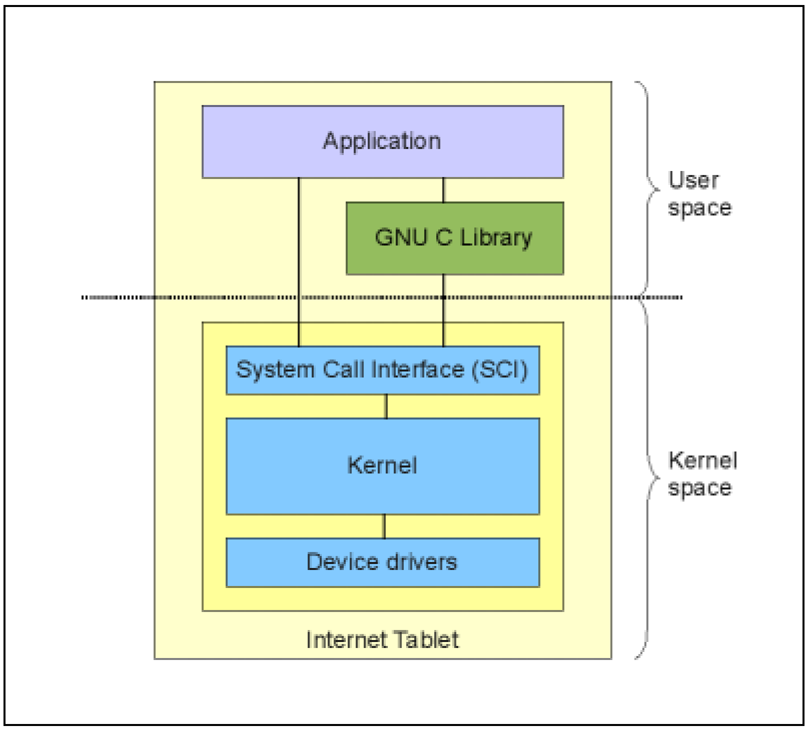

Figure 3.1: Kernel, hardware and software relations

## 3.2 What are processes

A process is a program that is being executed. It consists of the executable program code, a bunch of resources (for example open files), an address space, internal data for kernel, possibly threads (one or more) and a data section.

Each process in Linux has a series of characteristics associated with it, below is a (far from complete) list of available data:

**PID** Process ID, numeric identifier of the process

**State** State of the process (running, stopped etc.)

**PPID** Parent process ID

**Children**

**Open files** List of open files for the process

**TTY** The terminal to which the process is connected

**RUID** Real user ID, the owner of the process

Every process has a PID, process ID. This is a unique identification number used to refer to the process and a way to the system to tell processes apart from each other. When user starts the program, the process itself and all processes started by that process will be owned by that user (process RUID), thus processes' permissions to access files and system resources are determined by using permissions for that user.

## 3.3 Creating a process

To understand the creation of a process in Linux, we must first introduce few necessary subjects, `fork`, `exec`, `parent` and `child`. A process can create an exact clone of itself, this is called `forking`. After the process has forked itself the new process that has born gets a new PID and becomes a child of the forking process, and the forking process becomes a parent to a child. But, we wanted to create a new process, not just a copy of the parent, right? This is where `exec` comes into action, by issuing an `exec` call to the system the child process can overwrite the address space of itself with the new process data (executable), which will do the trick for us.

This is the only way to create new processes in Linux and every running process on the system has been created exactly the same way, even the first process, called `init` (PID 1) is forked during the boot procedure (this is called `bootstrapping`).

If the parent process dies before the child process does (and parent process does not explicitly handle the killing of the child), `init` will also become a parent of `orphaned` child process, so its' PPID will be set to 1 (PID of `init`), see Figure 3.2.

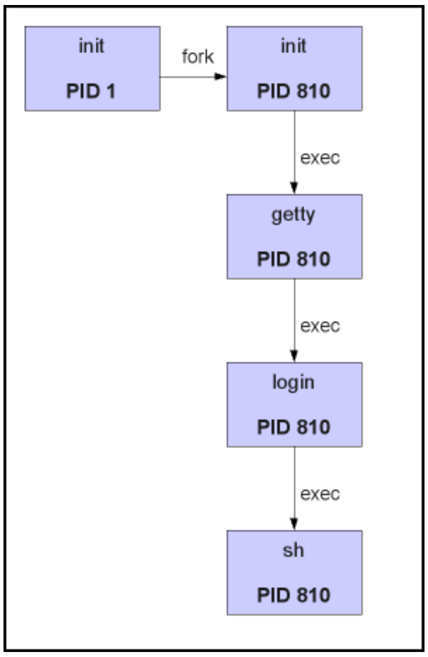

Figure 3.2: Example of the fork-and-exec mechanism

Parent-child relations between processes can be visualized as a hierarchical tree:

```
init-+-gconfd-2
     |-avahi-daemon---avahi-daemon
     |-2*[dbus-daemon]
     |-firefox---run-mozilla.sh---firefox-bin---8*[{firefox-bin}]
     |-udevd

...listing cut for brewity...
```

The tree structure above also states difference between `programs` and `processes`. All processes are executable programs, but one program can start multiple processes. From the tree structure you can see that running `Firefox` web browser has created 8 child processes for various tasks, still being one program. In this case, the preferred word to use from Firefox would be an application.

## 3.4 Ending a process

Whenever a process terminates normally (program finishes without intervention from outside), the program returns numeric `exit status` (return code) to the parent process. The value of the return code is program-specific, there is no standard for it. Usually `exit status 0`, means that process terminated normally (no error). Processes can also be ended by sending them a `signal`. In Linux there are over 60 different signals to send to processes, most commonly used are listed in Table 3.1.

| SIGNAL | Signal number | Explanation |
|---|---|---|
| SIGTERM | 15 | Terminate the process nicely. |
| SIGINT | 2 | Interrupt the process. Can be ignored by process. |
| SIGKILL | 9 | Interrupt the process. Can NOT be ignored by process. |
| SIGHUP | 1 | Used by daemon-processes, usually to inform daemon to re-read the configuration. |

Table 3.1: Most commonly used process signals in Linux

Only the user owning the process (or `root`) can send these signals to the process. Parent process can handle the `exit status` code of the terminating child process, but is not required to do so, in which case the `exit status` will be lost.

## 3.5 Filesystem hierarchy

Linux follows the UNIX-like operating systems concept called unified hierarchical namespace. All devices and filesystem partitions (they can be local or even accessible over the network) apper to exist in a single hierarchy. In filesystem namespace all resources can be referenced from the **root** directory, indicated by a forward slash (/), and every file or device existing on the system is located under it somewhere. You can access multiple filesystems and resources within the same namespace: you just tell the operating system the location in the filesystem namespace where you want the specific resource to appear. This action is called `mounting`, and the namespace location where you attach the filesystem or resource is called a `mount point`.

The mounting mechanism allows establishing a coherent namespace where different resources can be overlaid nicely and transparently. In contrast, the filesystem namespace found in Microsoft operating systems is split into parts and each physical storage is presented as a separate entity, e.g. **C:** is the first hard drive, **E:** might be the CD-ROM device.

Example of mounting: Let us assume we have a memory card (MMC) containing three directories, named **first**, **second** and **third**. We want the contents of the MMC card to appear under directory **/media/mmc2**. Let us also assume that our `device file` of the MMC card is **/dev/mmcblk0p1**. We issue the `mount` command and tell where in the filesystem namespace we would like to `mount` the MMC card.

```
/ $ sudo mount /dev/mmcblk0p1 /media/mmc2
/ $ ls -l /media/mmc2
total 0
drwxr-xr-x 2 user group 1 2007-11-19 04:17 first
drwxr-xr-x 2 user group 1 2007-11-19 04:17 second
drwxr-xr-x 2 user group 1 2007-11-19 04:17 third
/ $
```

In addition to physical devices (local or networked), Linux supports several `pseudo` filesystems (virtual filesystems) that behave like normal filesystems, but do not represent actual persistent data, but rather provide access to system information, configuration and devices. By using pseudo filesystems, operating system can provide more resources and services as part of the filesystem namespace. There is a saying that nicely describes the advantages: "In UNIX, everything is a file".

Examples of `pseudo` filesystems in Linux and their contents:

**procfs** Process filesystem, used to access process information from the kernel. Usually mounted under **/proc**

**devfs** Used for presenting `device files`, e.g. devices as files. An abstraction for accessing I/O an peripherals. Usually mounted under **/dev**

**sysfs** Information about devices and drivers from the kernel, also used for configuration. Usually mounted under **/sys**

Using pseudo filesystems provides a nice way to access kernel data and several devices from userspace processes using same API and functions than with regular files.

Most of the Linux distributions (as well as maemo platform) also follow the Filesystem Hierarchy Standard (FHS) quite well. FHS is a standard which consists of a set of requirements and guidelines for file and directory placement under UNIX-like operating systems 3.3.

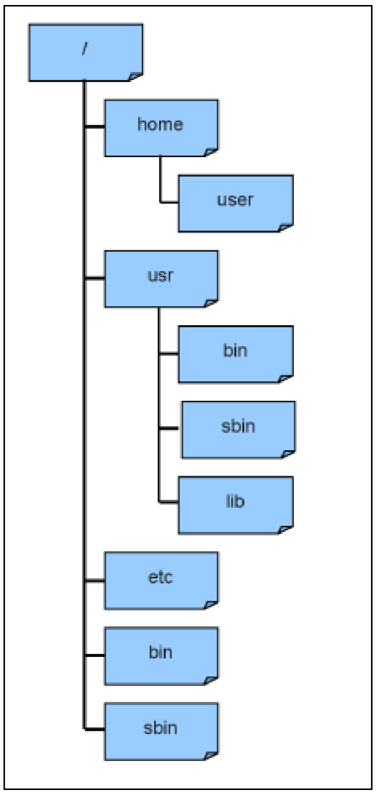

Figure 3.3: Example of filesystem hierarchy

The most important directories and their contents:

**/bin** Essential user command binaries that need to be available also in `single user mode`.

**/sbin** Essential system binaries (e.g. `init`, `insmod`, `ifup`)

**/lib** Libraries for the binaries in **/bin** and **/sbin**

**/usr/bin** Non-essential user command binaries that are not needed in `single user mode`

**/usr/sbin** Non-essential system binaries (e.g. daemons for network-services)

**/usr/lib** Libraries for the binaries in **/usr/bin** and **/usr/sbin**

**/etc** Host-specific system-wide configuration files

**/dev** Device files

**/home** User home directories (optional)

**/proc** Virtual file system documenting kernel and process status as text files

More information about FHS can be found from its' homepage at pathname.com/fhs/.

## 3.6 Files and inodes

For most users understanding the tree-like structure of the filesystem namespace is enough. In reality, things get more compilated than that. When a physical storage device is taken into use for the first time, it must be partitioned. Every partition has its own filesystem, which must be initialized before first usage. By mounting the initialized filesystems, we can form the tree-structure of the entire system.

When a filesystem is created to partition, a data structures containing information about files are written into it. These structures are called `inodes`. Each file in filesystem has an `inode`, identified by inode serial number.

Inode contains the following information of the file:

**Owner** Owner of the file

**Group** Group owner of the file

**Permissions** File permissions, more on these to follow

**Last read time** Time when file was last accessed (atime)

**Last change time** Time when file was last modified (mtime)

**Inode change time** Time when inode itself was last modified (ctime)

**Hard links** Number of hard links to this file

**Size** The length of the file in bytes

**Address** Pointer to the actual content data of the file

The only information not included in an inode is the file name and directory. These are stored in the special directory files, each of which contains one filename and one inode number. The kernel can search a directory, looking for a particular filename, and by using the inode number the actual content of the inode can be found, thus allowing the content of the file to be found.

Inode information can be queried from the file using `stat` command:

```
user@system:~$ stat /etc/passwd
  File: '/etc/passwd'
  Size: 1347            Blocks: 8          IO Block: 4096   regular file
Device: 805h/2053d      Inode: 209619      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2007-11-23 06:30:49.199768116 +0200
Modify: 2007-11-17 21:27:02.271803959 +0200
Change: 2007-11-17 21:27:02.771832454 +0200
```

Notice that the `stat` command shown above is from desktop Linux, as maemo platform does not have `stat` -command by default.

Some advantages of `inodes`:

- Multiple filenames can be associated with the same `inode`. This is called `hard linking` or just linking. Notice that `hard links` only work within one partition as the inode numbers are only unique within a given partition.

- As processes open files, the kernel converts the filename to an inode number at the first possible chance, thus moving the file (within same partition) does not prevent a process from accessing the file once it is opened (as long as the process knows the inode number).

In addition to hard links, Linux filesystem also supports `soft links`, or more commonly called, `symbolic links` (or for short: `symlinks`). A symbolic link contains the path to the target file instead of a physical location on the hard disk. Since `symlinks` do not use inodes, `symlinks` can span across the partitions. Symlinks are transparent to processes which read or write to files pointed by a symlink, process behaves as if operating directly on the target file.

## 3.7 File access permissions

The Linux security model is based on the UNIX security model. Every user on the system has an user ID (`UID`) and every user belongs to one or more groups (identified by group ID, `GID`). Every file (or directory) is owned by a `user` and a `group user`. There is also a third category of users, `others`, those are the users that are not the owners of the file and do not belong to the group owning the file.

For each of the three user categories, there are three permissions that can either be granted or denied:

**Read (r)** Whether the file may be read. In the case of a directory, the ability to list the contents of the directory.

**Write (w)** Whether the file may be written to or modified (includes deleting and renaming).

**Execute (x)** Whether the file may be executed. In the case of a directory, the ability to enter to or execute program from that directory.

The file permissions can be checked simply by issuing a `ls -l` command:

```
/ $ ls -l /bin/ls
-rwxr-xr-x 1 root root 78004 2007-09-29 15:51 /bin/ls
/ $ ls -l /tmp/test.sh
-rwxrw-r-- 1 user users 67 2007-11-19 07:13 /tmp/test.sh
```

The first 10 characters in output describe the file type and `permission flags` for all three user categories:

- First character tells the file type (- means regular file, `d` means directory, `l` means symlink)
- Characters 2-4 display the access rights for the `actual owner` of the file (- means denied)
- Characters 5-7 display the access rights for the `group owner` of the file (- means denied
- Character 8-10 display the access rights for `other users` (- means denied)

The output also lists the `owner` and the `group owner` of the file, in this order.

Let us look closer what this all means. For the first file, **/bin/ls**, the owner is root and group owner is also root. First three characters (rwx) indicate that `owner` (root) has read, write and execute permissions to the file. Next three characters (r-x) indicate that the `group owner` has read and execute permissions. Last three characters (r-x) indicate that all other users have read and execute permissions.

The second file, **/tmp/test.sh**, is owned by `user` and group owned by users belonging to `users` group. For `user` (owner) the permissions (rwx) are read, write and execute. For users in `users` group the permissions (rw-) are read and write. For all other users the permissions (r–) are only read.

File permissions can also be presented as octal values, where every user category is simply identified with one octal number. Octal values of the permission flags for one category are just added together using following table:

```
                   Octal value       Binary presentation
Read access             4                   100
Write access            2                   010
Execute access          1                   001

Example: converting "rwxr-xr--" to octal:

First group  "rwx" = 4 + 2 + 1 = 7
Second group "r-x" = 4 + 1     = 5
Third group  "r--" = 4         = 4

So "rwxr-xr--" becomes 754
```

As processes run effectively with permissions of the user who started the process, the process can only access the same files as the user.

Root-account is a special case: Root user of the system can override any permission flags of the files and folders, so it is very adviseable to **not** run unneeded processes or programs as `root` user. Using `root` account for anything else than system administration is not recommended.

## 3.8 Programs, daemons and libraries

As we earlier stated, processes are programs executing in the system. Processes are, however, also a bit more than that: They also include set of resources such as open files, pending signals, internal kernel data, processor state, an address space, one or more threads of execution, and a data section containing global variables. Processes, in effect, are the **result** of running program code. A program itself is not a process. A process is an active program and related resources.

Threads are objects of activity inside the process. Each thread contains a program counter, process stack and processor registers unique to that thread. Threads require less overhead than forking a new process because the system does not initialize a new system virtual memory space and environment for the process. Threads are most effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing.

Daemons are processes that run in the background unobtrusively, without any direct control from a user (by disassociating the daemon process from the controlling TTY). Daemons can perform various scheduled and non-scheduled tasks and often they serve the function of responding to the requests from other computers over a network, other programs or hardware activity. Daemons have usually `init` as their `parent process`, as daemons are launched by forking a child and letting the parent process die, in which case `init` adopts the process. Some examples of the daemons are: httpd (web server), lpd (printing service), cron (command scheduler).

`Linking` refers to combining a program and its libraries into a single executable and resolving the symbolic names of variables and functions into their resulting addresses. Libraries are a collection of commonly used functions combined into a package.

There are few types of linking:

**Static linking** Static linking copies a set of routines to the executable target application, thus expanding the target executable size.

**Dynamic linking** Dynamic linking means that the subroutines of a library are loaded into an application program at runtime. Dynamic libraries remain on filesystem as separate files from executable. Most of the work of linking is done at the time the application is loaded, thus creating a bit of overhead in application startup time.

**Run-time linking** Run-time linking (or dynamic loading) is a subset of dynamic linking where a dynamically linked library loads or unloads at run-time on request.

In addition to being loaded statically or dynamically, libraries can also be `shared`. Dynamic libraries are almost always shared, static libraries can not be shared at all. Sharing allows same library to be used by multiple programs at the same time, sharing the code in memory.

Dynamic linking of shared libraries provides multiple benefits:

- Common code of the applications is shared at runtime, which reduces the total memory usage.

- Application executable file size is reduced, which reduced the total storage usage.
- Fixing a bug from shared code fixes the bug from all the applications using the same code.

Most Linux systems use almost entirely dynamic libraries and dynamic linking.

## 3.9 Decomposition of a simple command-line program

Below is an image which shows the decomposition of a very simple command-line-program in Linux, dynamically linking only to `glibc` (and possibly `Glib`). As Glib is so commonly used in addition to standard C library (`glibc`), those libraries have been drawn to one box, altough they are completely separate libraries. The hardware devices are handled by the kernel and the program only accesses them through the system call (`syscall`) interface.

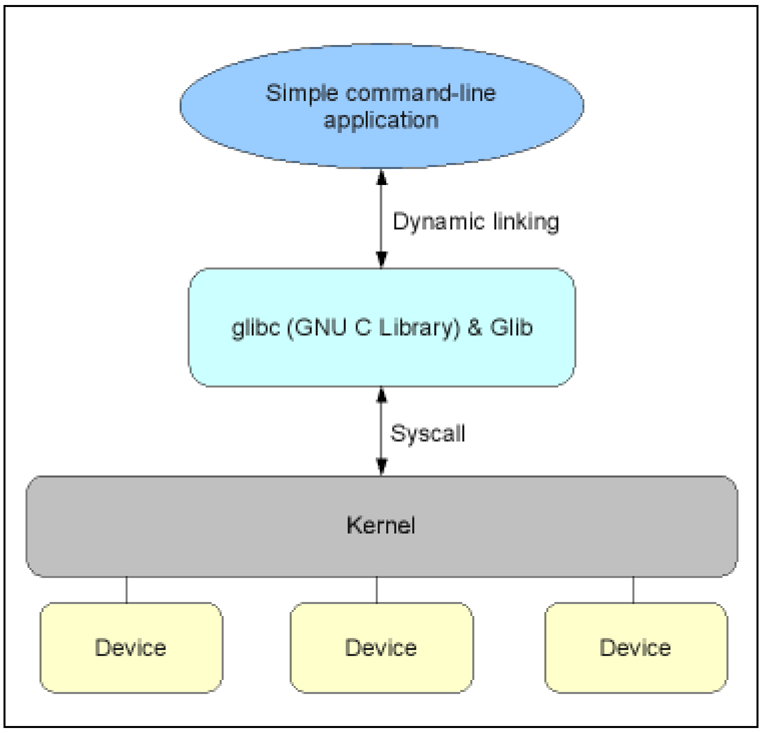

Figure 3.4: Decomposition of a simple command-line program

# Chapter 4

# The GUI Components of maemo

This chapter introduces the basics of the graphical user interface and GUI programming components used in the maemo platform, different views of Hildon desktop, `event-loop` -based GUI model and signals.

## 4.1 Decomposition of a simple GUI-program

Comparing the simple command-line-program (in previous chapter) and simple GUI-program running on maemo platform reveals the extensive usage of different libraries for GUI programs. Maemo platform provides many APIs to handle the GUI generation, resource management and application integration to the application framework. These APIs also hide the complexity of the `X library`, which normal maemo GUI programs don't have to care about, although it is used internally by the GUI libraries provided. A decomposition view of a simple GUI program running on the maemo platform is seen on Figure 4.1.

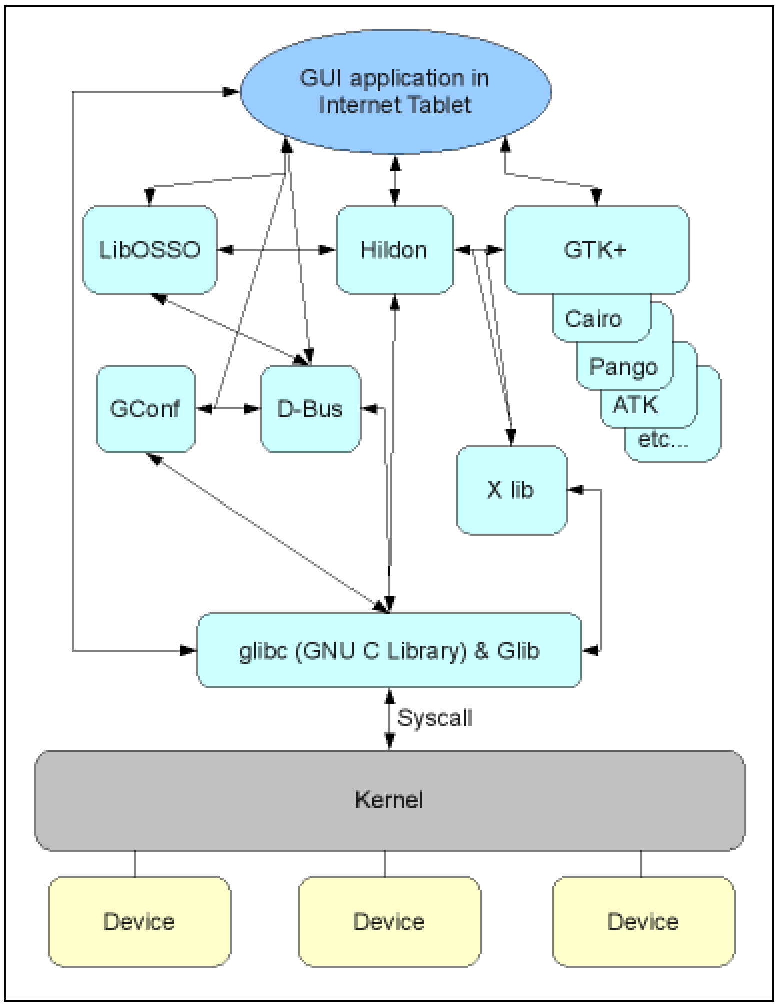

Figure 4.1: Decomposition of a simple GUI-program

## 4.2 The GUI components

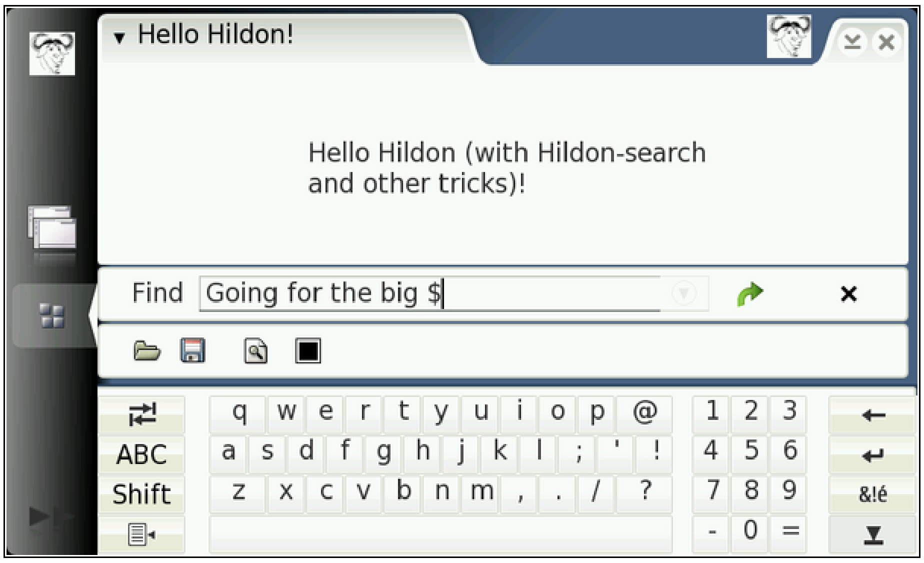

Figure 4.2: Internet Tablet graphical user interface

The graphical user interface application framework of maemo is called Hildon. It is based on the technologies that GNOME framework (used on many desktop Linuxes) is built on, most importantly the GTK+. Hildon contains several enhancements to GTK+ making it more suitable to use on the Internet Tablets: Hildon widgets, speed-improved Sapwood theme engine, image server, task navigator, control panel, status bar, touch screen input method, stylus support and a window management on a high-pixels-per-inch screen.

The GUI programming APIs for maemo are based on GTK+ widgets and Hildon extensions on top of it. Most GTK+ widgets and methods work in the Hildon environment without modification, the most important exception being the application main window widget which is replaced by Hildon window.

Only one application is visible at the time, as the application's window fills the whole `application area`. Switching between running applications is done from `task navigator`. Task navigator also includes menus for launching new applications. Statusbar (titlebar) area includes application menu and also buttons to close and minimise the running application. The application has only one main menu, with submenus spawning horizontally to the right, so the application developer must plan menus carefully as the space for menus is limited. The on-screen virtual keyboard is launched automatically when the user of the Internet Tablet with stylus-input activates a text-input UI element. The running application is resized as the on-screen-keyboard reserves space from the application area.

Statusbar/titlebar can be expanded by user defined plug-in applications, providing different status information of the applications. Also the main window (visible when no applications are running, or all applications are minimised) allows running plug-ins, called `home applets`, usually being some kind of small informational applications, e.g. news ticker, weather information or clock.

## 4.3 Hildon user interface views

The Hildon user interface has several layout modes which applications can use, and even switch between views dynamically.

### 4.3.1 Normal view

- Application area of 696x396 pixels
- Task navigator, statusbar/titlebar visible

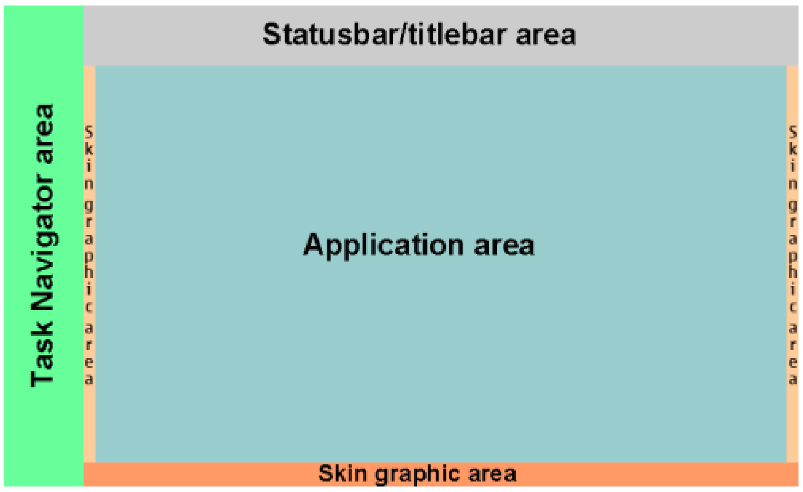

Figure 4.3: Normal view

### 4.3.2 Normal view with toolbar

- Application area of 696x360 pixels with a single toolbar
- Application area of 696x310 pixels with two toolbars (e.g. Application and Find toolbars)
- Task navigator, statusbar/titlebar visible
- Skin graphics area on the bottom of the screen replaced by toolbar

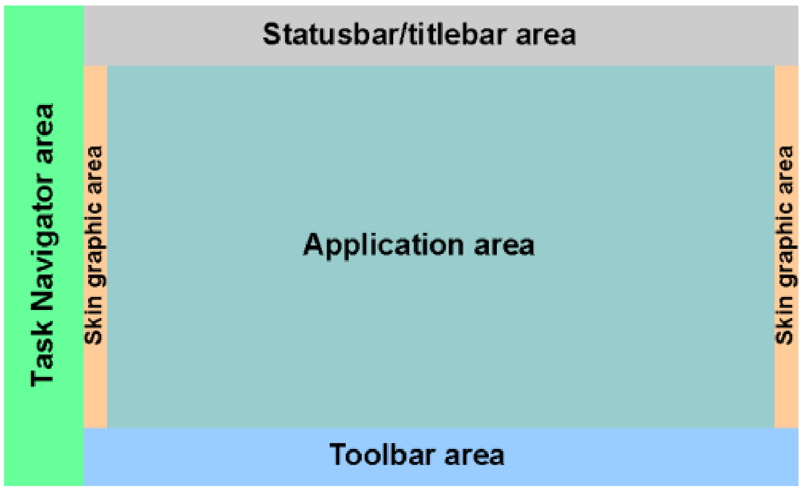

Figure 4.4: Normal view with Toolbar

### 4.3.3 Full screen view

- Application area of 800x480 pixels fully available
- Task navigator, statusbar/titlebar and skin graphic area not visible
- Mode can be activated and deactivated by a hardware button or by the application code

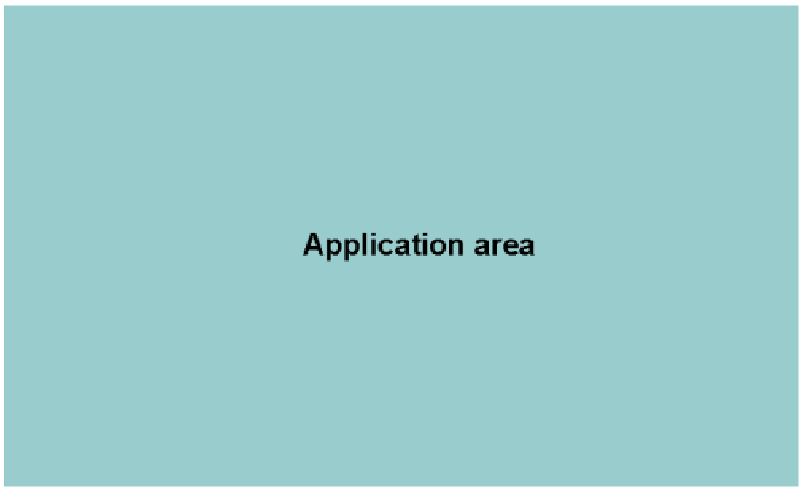

Figure 4.5: Full screen view

### 4.3.4 Full screen view with toolbar

- Variation of the full screen view

- Application area of 800x422 pixels with a single toolbar
- Application area of 800x370 pixels with two toolbars (e.g. Application and Find toolbars)
- Task navigator, statusbar/titlebar and skin graphic area not visible
- Toolbar should be scalable as it can be visible in both normal and full screen modes

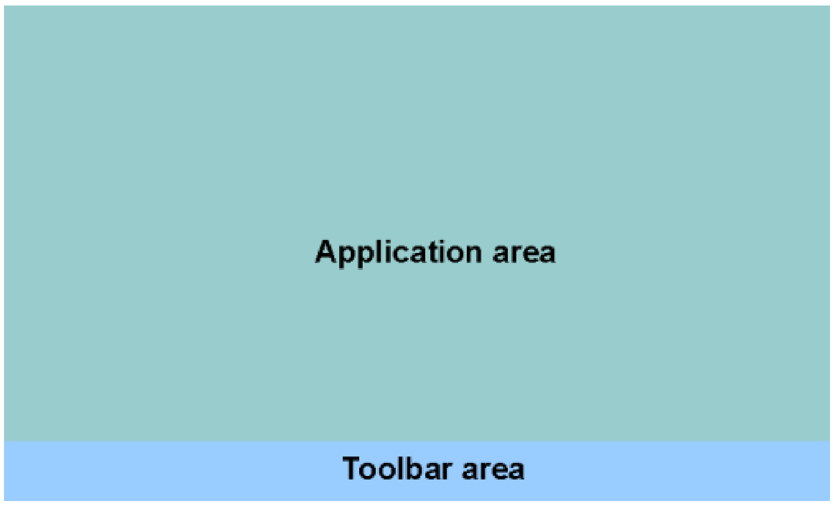

Figure 4.6: Full screen view with toolbar

## 4.4 Event-loop model

GTK+ is a event-loop based (or event-driven), cross-platform GUI library. In event-loop programming model when the user is doing nothing, GTK+ waits in its `main loop`, waiting for input. When user performs an action - for example a click of a button - the main loop wakes up and sends an event to one or more widgets. When widget receives an event, they usually emit one or more `signals`. These signals can be connected in the application code to functions performing certain action based on the signal emitted and the widget emitting the signal. Functions connected to a signal are referred as `callback` functions. After a callback finishes its task, GTK+ will return to the main loop and wait for more input. Events can also be sent by the application engine itself.

There are graphical GUI-builder applications that assist in creation of the UI-schema and even binding the signals from UI elements to the callback functions, speeding up the development process significantly. Most commonly used GTK+ GUI-builders are Gazpacho and Glade.

## 4.5 Asynchronous programming model

Sometimes it is necessary to perform actions in the application at the same time while the `main loop` sits and waits for events. This approach is called

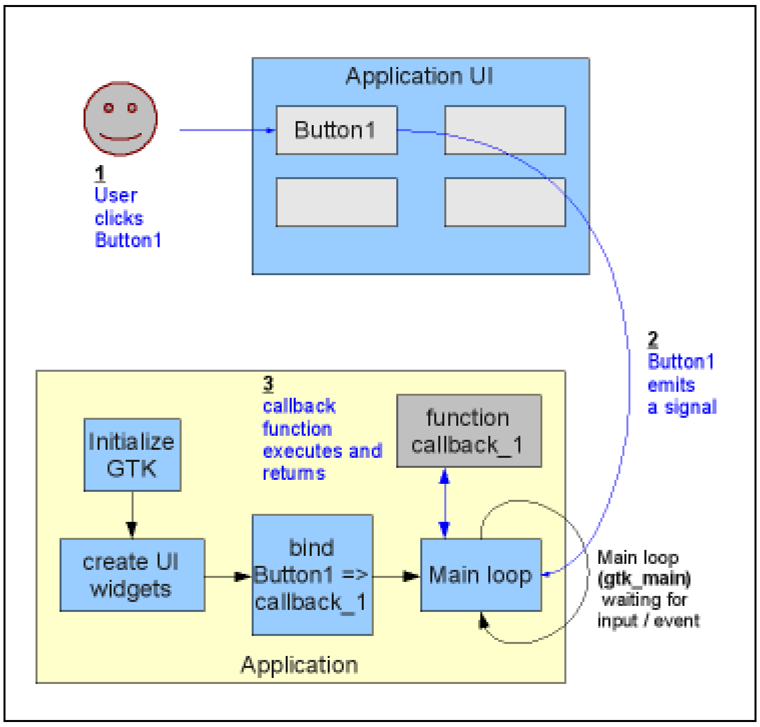

Figure 4.7: Event-loop model, signal and a callback

asynchronicity. Asynchronous actions are executed as non-blocking, meaning the control return to the caller immediately without interrupting the main program flow. The caller will have to specify a callback function that will be called when the operation is completed. Using asynchronicity makes the application UI more responsive and prevents the application "locking up" while, for example, reading data from over the network connection. For example GnomeVFS API has asynchronous counterparts to all functions. Callbacks for asynchronous operations are triggered in the normal event-loop, meaning that the application will be able to handle both GUI events and GnomeVFS events simultaneously. Other example of the API supporting asynchronicity is D-Bus, more information about using D-Bus asynchronously can be found from the maemo platform development course material.

Other option to achieve asynchronicity and improve UI responsiveness is to use threads, but this approach is not recommended unless you have experience on thread based programming. Threaded applications and other software components are hard to debug and may easily cause synchronisation problems.

# Chapter 5

# Maemo Platform Overview

This chapter decomposes the maemo platform into components and describes the scope of those components in the platform. Also the overall design is shortly introduced, with the most commonly needed programming APIs.

## 5.1 Overall design

Maemo is a `Debian GNU/Linux` based embedded operating system designed for networked mobile devices, called Internet Tablets. Being based on Linux and Debian which support the ideology of sharing the source code, collaboration and open development model, maemo is also open source.

Maemo runs on a recent 2.6 version of `Linux` kernel. The user space software links with the `GNU C library, glibc`. Maemo aims at being as much compatible with the mainstream Linux systems as is possible, reducing the time and effort needed for porting existing applications and developing new ones to maemo platform.

The package management framework comes from the Debian distribution, simplifying and automating the process of installation, upgrading, configuring and uninstallation of the software packages.

The user interface architecture is based on `GNOME` framework, especially on `GTK+` widgets. `GTK+` has been further extended by `Hildon` to better suit the needs of an Internet Tablet. The actual user interface engine under GTK+ is `X Window System` (X Server) with `Matchbox` window manager. GUI applications are built using Hildon framework and GTK+ widgets, although using X Server directly with `Xlib` API is possible, but not recommended.

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

| | | | | | | |
|---|---|---|---|---|---|---|
| Applications | | | | | | |
| Fonts | | Sounds | | | Icons | |
| Connectivity | | System UI | Search | Text Input | | MIME Types |
| Home Applets | | Control Panel | | Task Navigator | | Status Bar |
| Backup | | Installer | Alarm | Help | | Launcher |
| XML | E-D-S | | Telepathy | | GConf | |
| GStreamer | | GnomeVFS | | | | GSF |
| Sapwood | | Hildon Widgets | | Hildon File UI | | HTML Widget |
| GTK+ | | | | | | |
| GDK | | | | GdkPixbuf | | |
| Pango | | Cairo | | | Atk | |
| GLib | | | | GObject | | |
| Samba | GPS | Obex | ConIC | UPnP | JPEG PNG TIFF SVG | Matchbox |
| D-BUS | | HAL | SQLite | curl HTTP | Clipboard | |
| SSL | System SW | | Cert. mgnt | libosso | X | |
| Libstd C++ | | Compressi | dpkg | apt | Freetype | Fontconfig |
| Sysvinit | Base Files | Busybox | GNU C Library | Core Libs | Core Utils | Core Daemons |
| BlueZ | | Power mgnt | | WLAN security | ALSA | Video4-Linux |
| Bootloader | | Linux kernel including JFFS2, TCP/IP | | | | InitFS including uClibc dsme |

## 5.2 Core components

### 5.2.1 Linux kernel

The Linux kernel is the heart of the maemo platform. The kernel is loaded at very early stage during the boot process by a bootloader. Maemo platform is based on Linux kernel version 2.6 and current Internet Tablets utilising the maemo platform use the OMAP chipset containing a ARM processor and a DSP unit. The kernel implements the memory management, process management, networking services as well as hardware specific device and bus drivers. The device drivers include e.g. USB, LCD and WLAN. The bus drivers include e.g. I2C and Flash bus. Part of the kernel functionality, such as device drivers, network protocols or filesystem support can be implemented as loadable kernel modules which can be loaded or removed during runtime. Kernel is stored on a separate flash partition on Internet Tablet, called kernel partition. User space applications communicate to the kernel using system calls.

It is possible for developer to modify or configure the default kernel, as the sources for kernel are available. There is a guide at maemo community website describing the process. Modified kernel can also be flashed back to the device, but when doing so developer must take into account the size of the kernel partition on the Internet Tablet, as there is no way to change its size.

### 5.2.2 InitFS

At the last state of the kernel boot process the initial filesystem, `InitFS`, is mounted as the root filesystem. InitFS is a small filesystem used during the boot time, containing necessary binaries to bring the Internet Tablet normal state. After the boot scripts on InitFS are done, the final root filesystem is mounted from flash and the InitFS is mounted into /mnt/initfs.

### 5.2.3 Base system

The root filesystem includes the basic filesystem hierarchy of a Debian system and Debian based core distribution with few exceptions. The coreutils and Bash (shell) has been replaced with size-optimised `Busybox`, which combines many common UNIX utilities into a single small executable. Few utilities have also been dropped out because of space saving.

As the maemo platform includes all the basic UNIX utilities and a shell, it is possible to create and run shell-scripts and combine those small but powerful utilities to solve complex tasks easily. The user accounts and groups are handled just like stated in the first chapter. Internet Tablets have predefined user accounts, `user` and `root`. The root account should be used for administrative purposes only and is protected by default.

The filesystem hierarchy follows filesystem Hierarchy Standard (FHS) quite well. All applications are normally installed under the `/usr` directory and must use the hierarchy described for `/usr` in FHS. In addition to the directories specified in FHS, the following are placed under `/usr`:

**share/icons/hicolor/<size>/apps** Icon files

**share/sounds** Sound files

**share/themes** GUI themes

The user's home directory (`/home/user`) can be used quite freely, with the exception of the directory `/home/user/MyDocs` which is reserved for user's own files (e.g. documents, images) visible through the GUI.

File management operations for user files must be performed through an API provided by the application framework as maemo platform manages file operations case-insensitively, even when Linux filesystems are case-sensitive.

Below is an image of different partitions and their relational sizes on the internal flash of the Internet Tablet:

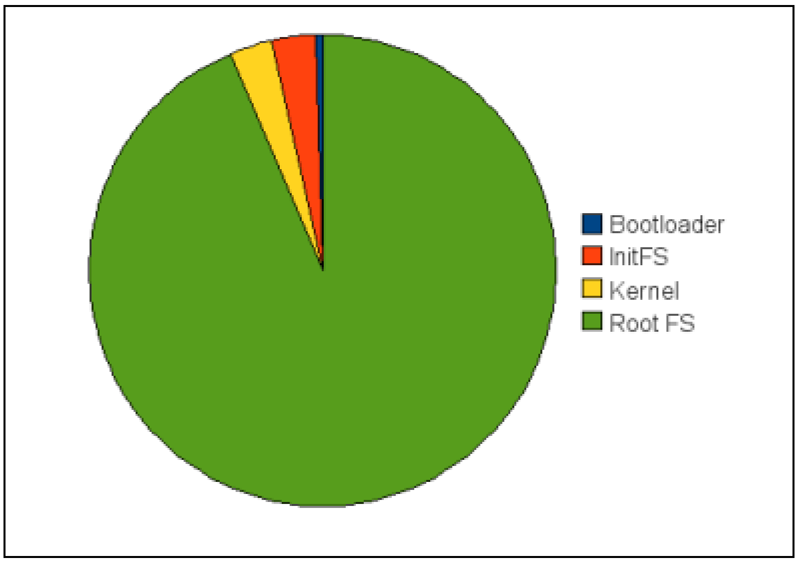

Figure 5.1: Partitions on Internet Tablet internal flash

## 5.3 Generic programming libraries

Below is a list of the generic programming libraries used for developing applications on maemo platform, and a short description of them.

**GNU C Library (glibc)** glibc is a standard C library released by the GNU Project. It provides the functionality required by POSIX `1c`, `1d` and `1j` standards as well as ANSI C and some of the functionality required by ISO C99. This library is used (indirectly at least) by every application running on maemo platform.

**GLib** General-purpose utility library providing many portable data types, macros, type conversions, string utilities, object-oriented framework (GObject), event mechanism, etc.

**GObject** GObject provides the implementation of flexible and extensible object-oriented framework for C language.

**GConf** Provides a centralised configuration management framework. Allows applications to store and retrieve their settings in a consistent manner, without the need to use configuration files.

**Gnome-VFS** Filesystem abstraction library, extendable by plug-ins which allow 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.

**LibOSSO** LibOSSO is a basic library containing required and helpful functions for maemo applications. LibOSSO also contains a wrapper allowing 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. Also provides a GUI application an interface to register itself to the application framework, preventing the application to be killed as a "stray" application.

**D-Bus** A service that allows related processes to pass events to each other. 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.

## 5.4 GUI programming interfaces

When developing GUI applications on maemo platform, following libraries are essential:

**GTK+** The GIMP toolkit, a multi-platform toolkit for creating graphical user interfaces. Graphical elements in GTK+ are called widgets. GTK+ also supports the notion of themes, which are user switchable sets of graphics and behaviour models. Uses GLib, GDK, Pango and ATK.

- **GDK** GDK (GIMP Drawing Kit) is a graphics library that acts as a wrapper around the low-level drawing and windowing functions provided by the underlying graphics system.
- **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. Used by GTK+ for all displayed text.
- **Atk** The Accessibility ToolKit. Provides generic methods by which an application can support people with special needs with respect to using computers.

**Cairo** Cairo is a 2D graphics library designed to produce consistent output on all output media while taking advantage of display hardware acceleration when available.

**Hildon** A library containing widgets and themes designed specifically for maemo, enhancing GTK+. This library is necessary when creating GUI applications on maemo platform since the screen has very high PPI and applications are sometimes controlled via a stylus.

## 5.5 Audio and Video programming interfaces

The maemo platform includes several APIs to handle multimedia (audio and video) in applications:

**ALSA** The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality. It also includes support for the older Open Sound System (OSS) API, providing compatibility with older software.

**ESD** The Enlightened Sound Daemon (ESD or EsounD) is the sound server for GNOME. It mixes several sound streams into on for output. It can also manage network-transparent audio.

**GStreamer** GStreamer is a plug-in expandable multimedia processing framework. The GStreamer framework is designed to make it easy to write applications that handle audio or video or both. Developers can write new plug-ins to add support for new formats to existing applications. GStreamer includes components for building a media player that can support a very wide variety of formats.

**Video4Linux** Video4Linux or V4L is a video capture API for Linux, also supported by the maemo platform and the integrated camera on some Internet Tablets. Video4Linux is closely integrated with the Linux kernel.

## 5.6 Communication interfaces

The Internet Tablet contains WLAN and Bluetooth hardware for connecting to the Internet wirelessly, as well as the USB port for cable connection. The maemo platform contains several APIs to handle the connectivity in the application:

**Sockets** Standard Linux sockets.

**TCP/IP** A standard TCP/IP protocol stack, provided by Linux kernel.

**BlueZ** Implementation of the Bluetooth$^{TM}$wireless standards specification. Includes kernel modules, libraries and utilities. Capable of communicating the tasks that should not be handled by kernel via D-Bus-interface.

**OpenSSL** Library providing a security layer for encrypted networking.

**curl HTTP** A client-side support library for URL-transfer, supporting protocols HTTP, HTTPS, FTP, FTPS, LDAP etc.

## 5.7 Other components and interfaces

The maemo software stack image contains a lot more applications and interfaces than the generally used listed above. Here is a short description of the most important ones:

**Alarm** The alarm framework provides an easy way to manage timed events in the device. It is powerful, and not restricted only to wake-up alarms. The framework provides many other features, e.g. setting multiple alarm events, setting custom icon and title of the shown alarm message, executing commands, booting up the device if it is turned off etc.

**Backup** The maemo backup application saves and restores application data stored in user's home directory ( /MyDocs) and setting directories and files. Other locations than the defaults can be configured to be backed up too.

**Camera** The built-in camera present in some Internet Tablets is compatible with Video-4-Linux version 2 API. Since the maemo platform delegates all multimedia handling to the GStreamer framework, applications that need access to the built-in camera should employ GStreamer for this instead of directly accessing Video4Linux devices, via the v4l2src GStreamer module.

**ConIC** An Internet connectivity library used to request connections, retrieve current statistics, proxies and settings for Internet Access Points (IAPs).

**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.

**GPS** The GPS framework in maemo platform consists of a GPS daemon and a library for controlling it, as well as methods to start and stop GPS devices (for power saving).

**E-D-S** The Evolution Data Server provides a single database for common, desktop-wide information, such as a user's address book or calendar events.

**HAL** The purpose of HAL (Hardware Abstraction layer) is to provide means for storing data about hardware devices, gathered from multiple sources and to provide an interface for applications to access this data.

**Help Framework** The Help Framework is a centralised way to offer help services to the user of the program. Maemo platform has an in-built help system that handles all the help documentation for the programs using the Help Framework. Libraries are used to register a program to the Help Framework, and after that the content of the actual help documentation can be used.

**Hildon** Hildon framework provides additional components on top of the GNOME components:

> **Home Applets** Home applets (or plug-ins) are small applications that run on the main window, providing different kind of information, e.g. news ticker or clock.
>
> **Task Navigator** Task navigator provides a menu used for switching between applications. To make application visible in Task navigator, you need to create a Desktop file for the application, containing information needed to show the application entry in the menu.
>
> **Status Bar** Status bar is a GUI component displaying status of the various tasks using tiny icons on the main window. The status bar can contain user defined items used by a plug-in, but with a limitation of two additional items (only the last two added are visible).

**Control Panel** Control panel is a standard and centralised place for application and server settings changeable by the user. Applications can provide Control Panel plug-ins to interface with the application settings.

**Installer** The Application manager is a GUI application used to install, upgrade and remove application packages for Internet Tablet. Internally the Application manager uses the Debian package management system.

**MIME Types** This component provides the Internet media type (MIME type) registry of two-part identifiers for file formats.

**OBEX** OBEX (OBject EXchange) is a communications protocol API that facilitates the exchange of binary objects between devices over a Bluetooth connection.

**Telepathy** Telepathy provides D-Bus-based framework that unifies all forms of real-time communication, such as instant messaging, IRC, voice and video over Internet. The framework provides an interface for plug-ins to extend the protocol support by implementing new connection managers.

**Text input (Hildon Input Method)** As the maemo platform is intended to be used on embedded devices, it is a quite logical that one might want to have different input methods from the ones available by default, or just simply want a different layout for the virtual keyboard. The maemo platform introduces a way to enable writing custom plug-ins for Hildon Input Method.

**Sysvinit** System V style init scripts that spawn or kill processes according to system run-level. On Internet Tablet mainly used during system startup and shutdown.

**Search** The maemo global search component provides a search framework.

# Chapter 6

# Runtime View of maemo

This chapter gives an overview of the application life-cycle on the maemo platform, introduces services used in the application state-management and resource-saving.

Components involved in the application life cycle management and switching are following:

- Task Navigator (TN)
  - Lists the applications in menu and launches them by user request
  - Performs a background killing of applications in case of low memory.
  - Lists applications, both running and background killed applications (user sees them as running applications).
  - Switches between running applications (or to background killed application) by:
    - Requesting the window manager to top the application window.
    - Sending the application a message requesting it to top a window.
    - Restarting the application if it was background killed.
- D-Bus session bus
  - Executes or activates applications by sending activation messages.
  - Takes care that only one instance of the application is running at a time.
  - Watches the application to register itself within given time-out. If application does not register, D-Bus assumes the application startup failed and will kill the process.
- Maemo launcher
  - Provides a way to speed up some applications startup time by providing a way to execute applications that has been compiled as a shared library.
- Window manager
  - Takes care of handling the window switching and window stacking.

## 6.1 Platform startup

The first phase of the platform startup is the bootloader, which will load the Linux kernel into memory. Linux kernel then loads the initial filesystem (InitFS) from its' own filesystem partition. InitFS, used as a root filesystem during the startup, contains necessary binaries and scripts to initialise the system and to access the actual root filesystem, which will be mounted after the InitFS scripts finish. Root filesystem contains several `init`-scripts, which will handle the startup of necessary daemons, services and the application framework itself, everything that makes the maemo platform. See chapter "maemo Platform Development" for more information.

## 6.2 Platform state management

Platform state is managed through several system software components:

**Device State Management Entity (dsme)** A daemon responsible for managing the states of the device, including shutdown and startup. It monitors the status of critical processes (such as D-Bus, X11 and Window Manager), initiates power saving operations based on inactivity etc.

**Mode Control (mce)** Provides interfaces for controlling device modes, such as offline mode (disabling of Bluetooth and WLAN), and various system level user interfaces, such as device lock, touch screen and keypad lock, LEDs, etc.

**Battery Management (bme)** Responsible for battery voltage monitoring and recognition, battery charging, and charger recognition.

One of the important (and interesting from developers point of view) components is D-Bus. The D-Bus message bus is a central part of the platform architecture. Applications should listen to D-Bus messages indicating the state of the device, such as "battery low" and "shutdown". When receiving these messages, the application may, for instance, ask the user to save any files that are open, or perform other similar actions to save the state of the application. There is also a specific system message, emitted when applications are required to close themselves.

## 6.3 Application startup

In maemo platform, user starts the applications primarily from the Task Navigator. There are also few other ways to start the application, either from the Status Bar (e.g. connection manager), from File Manager to view a file, or from other applications (e.g. "Send as E-mail").

User normally start applications from the Task Navigator. Task Navigator starts the application by sending a D-Bus message to the application service with the D-Bus `auto-activation flag` set.

Other applications can also start applications implicitly by sending a D-Bus message, for example to open a file of certain MIME-type that the application has registered to the MIME database.

Every application in the Internet Tablet has a well-known name uniquely identifying the application, e.g. "Browser" or "Email". D-Bus has a service name for each application, derived from the application name.

Where applications get the D-Bus service name:

**Task Navigator** The service name is specified in the applications' `.desktop` file

**File Manager and Browser** The service name is retrieved from the GnomeVFS MIME-type-handler registry via LibOSSO-MIME library

Other applications use the service application API libraries. The libraries know what services the application registers to D-Bus.

D-Bus daemon looks into application `.service` file to see how to execute the application before delivering the message. Applications launched by the D-Bus daemon will only have one instance of them running, because D-Bus does not allow the same service name to be registered by more than one process.

The application launched by the D-Bus daemon will inherit environment variables that were defined at the time when the D-Bus session bus was started. D-Bus does not provide a way to change the environment variables passed to an application. As the Task Navigator uses D-Bus to launch the application, it's same as directly launching the application using D-Bus.

Environment variable should not be used for dynamic configuration changes, since they require program restart and D-Bus does not support that. The language change is also communicated through environment variables. As applications and their libraries also cache locale state, changing the language in Internet Tablet requires restarting all applications and processes.

When the application is started, the window manager takes care of drawing the title bars, dialog borders and windows. Application windows are in a stack. When you top an application, it comes on top of the stack. The window manager also keeps the application dialogs stacked together with the application in the window stack, thus when the application is topped, it's dialogs are topped too.

The application can top itself, or the Task Navigator can top the application by sending a standard X message.

To conserve memory and resources, only single instance of an application can be running at the same time. If application is already running, it will only receive a message about the new invocation (for example "open file") and top itself (bring the window to foreground). User can switch to another, already running application either by using the Task Navigator UI, or by closing the topmost application.

## 6.4 Application state management

Application framework has a mechanism for shutting down GUI applications on the background to save memory so that other application can be run. This is called background killing.

Background killing is implemented by Task Navigator to transparently close an application when the user does not see it and to restart the application when user needs it again. This is possible because applications are required to be able to save their user interface (UI) states and Task Navigator has a list of all running UI applications. The application is required to save its UI state when it moves to background.

If in some cases Internet Tablets don't have enough memory to run all the applications at the same time, the system may kill an application running in the background that has indicated to be killable.

Saving the UI state may not always be feasible for the application (e.g. during a download in progress), that is why the application must notify the Task Navigator when it has saved the state and can be killed. Task Navigator will kill all the killable applications when system notifies that it is low of memory. When the application is started again, it is required to rebuild the UI according to the saved state. Task Navigator won't restart the application if there is not enough memory in the system for that.

### 6.4.1 UI State Saving

The application UI state saving is assisted by LibOSSO. The LibOSSO library creates the state file and provides the file descriptor to the application. Libosso will make sure that the real state file will not be updated until the state file write has completed and file closed. The application uses the standard POSIX filesystem API for writing and reading to the file descriptor.

If device is restarted, the UI states will be discarded, so applications start from their default state. Each application with different version number will have it's own UI state file, meaning that if the application is updated (version number changes), it will start from the default state.

### 6.4.2 Autosaving User data

Some applications are required to save the unsaved user data periodically when on the foreground (top), so that as little as possible is lost on battery failure. Applications should register a LibOSSO callback function for this operation and tell when their user data has changed (the application is in "dirty state"). LibOSSO will then tell the application when they should do the saving.

Note that applications should call "forced autosave" LibOSSO function when they go to background (LibOSSO does not know when this happens).

## 6.5 Application termination

Applications exit when user closes them from the application UI, or when system requests that. System will request (and force) the applications to exit when:

- The Internet Tablet battery charge drops low
- The Internet Tablet is switched off
- User changes the Internet Tablet language settings
- User resets the Internet Tablet to factory settings
- User tries close or to switch to an application that doesn't respond

  - User must accept to dialog whether to close a non-responsive application

If the Internet Tablet runs low on memory, the system can request application to be background killed. If there is not enough free memory to satisfy the applications' request for more memory, the application will be killed by the kernel.

# Chapter 7

# Software Development Process for maemo SDK

Software development for the maemo is possible using many different programming languages. This chapter describes the process and common practises of developing software for the maemo platform. It also introduces the tools that form the maemo SDK environment.

## 7.1 Overview of the software development process using the maemo SDK environment

The development environment used in the process is called `maemo SDK` which is freely downloadable from the maemo community website maemo.org. The maemo SDK utilises `Scratchbox`, which is a cross-compilation toolkit and a "sandbox", designed for embedded Linux application development. Scratchbox is downloadable from its' website scratchbox.org.

The maemo SDK provides a development environment for creating software to Internet Tablets using a Linux desktop computer. The SDK runs inside the Scratchbox and contains all necessary compilers, tools, libraries and headers to develop software for the two target hardware architectures, Intel (`x86`) and ARMEL. Application development and preliminary testing of the software is done in x86 environment, which also includes the Hildon Desktop for running the applications on the desktop computer (using virtual X server, such as `Xephyr`) like they would run on the actual Internet Tablet. Using desktop computer for development makes the application development quite similar to normal Linux application development. Maemo SDK also has a support-plug-in for Eclipse, Integrated Development Environment (IDE) which speeds up and helps the development process radically.

After the preliminary testing on desktop computer is finished, the next phase is to cross-compile and package the application for the ARMEL architecture using the ARMEL target of the Scratchbox, and the application is ready to be run and tested on the Internet Tablet. Testing-phase using the actual Internet Tablet is important even when the application runs fine on the desktop environment as the SDK is not exactly 100% identical to the device.

Development tools and resources used in maemo application development process:

**Scratchbox** Scratchbox is a cross compilation toolkit designed to make embedded Linux application development easier. It also provides a full set of tools to integrate and cross compile an entire Linux distribution. The toolkit supports ARM architecture and x86 and few more are under development. Scratchbox supports multiple configurations for each developer in the same host machine.

**maemo SDK rootstraps** The rootstrap is a target root filesystem image for Scratchbox that can be used as a basis for development. Maemo SDK provides rootstraps for both x86 and ARMEL development.

**Nokia binaries** Software packages that are not available as a source code but may provide public API, like the contact information import/export library, GPS (location) libraries, address-book and presence-information libraries.

**maemo tools** Several tools for power users requiring more sophisticated development tools than provided in the standard maemo SDK package. Includes tools for code analysis, debugging, resource usage, test automation etc.

**maemo.org repositories** maemo.org website has a lot of different repositories that are meant to be used with standard Debian package installation tools. Different repositories offer different software, tools, source code etc. for the developers.

**maemo.org documentation** Documentation for maemo software development include HOW-TOs, tutorials, API References, manual pages and several other guides, available from the maemo.org website.

**maemo examples** Maemo examples package includes demonstrative source code for using different APIs and can be fetched from the maemo repositories.

Phases of software development process using C or C++ language:

1. Create project (possibly using templates) for the application
2. Create or update the source code and needed resources
3. Create or update the UI schema (possibly using UI builder)
4. Build the application with `x86 rootstrap`
5. Launch and test the application on `x86 rootstrap`
6. Debug the application on `x86 rootstrap`
7. Cross-compile the application with `ARMEL rootstrap`
8. Launch and test the application on the Internet Tablet

9. Debug application on Internet Tablet
10. Create ARMEL installation package for the Internet Tablet
11. Install ARMEL application package to the Internet Tablet

Phases of software development process using Python (and other script-languages):

1. Create project (possibly using templates) for the application
2. Create or update the source code and needed resources
3. Launch and test the application on `x86 rootstrap`
4. Debug the application on `x86 rootstrap`
5. Launch and test the application on the Internet Tablet
6. Debug application on the Internet Tablet
7. Create ARMEL installation package for the Internet Tablet
8. Install ARMEL application package to the Internet Tablet

Following chapters take a deeper look into the phases of the development process.

## 7.2 Creating project for application

Creating the project for an application can be done either from scratch, or by using several examples available from maemo web site as templates. Source files can be edited using your favourite text editor. Scratchbox creates a "sandbox", a separate filesystem (called `rootstrap`) under your normal filesystem, so it is advisable to create a symbolic link to Scratchbox folders for easier access of files from your desktop environment. Of course, using a console-based text editor (such as `nano`) inside Scratchbox shell is also possible.

As maemo uses Debian-based package management system for applications, it is a good practise to take that into account already when creating a project and create necessary files for packaging or use helper-applications for creating them, such as GNU Autotools.

Notice, the Hildon Desktop must be started before running the maemo applications inside the SDK.

The official programming language for maemo application development is C, but several other languages can be used with maemo also, for example C++ and Python.

There exists several UI-builder applications to speed up the creation of UI schema, most commonly used are Gazpacho and Glade.

Complete guide of setting up and using the development environment can be found from maemo Getting Started and maemo Application Development materials.

## 7.3 Building and running applications

When source code has been created with necessary resource-files, the application is ready for compiling and testing. The SDK provides all the usual Linux development tools inside the Scratchbox as well as the maemo application framework so the applications look and behave like they would on the Internet Tablet.

List of the most commonly used development tools provided by the `maemo rootstrap`:

**GNU toolchain** An umbrella term used of the programming tools produced by GNU Project, including:

> **GCC (GNU Compiler Collection)** Compilers and linkers for C, C++ etc.
>
> **GNU Autotools** Suite of programming tools designed to assist in `Makefile` generation and portability-issues.
>
> **GNU Make** Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.
>
> **GNU Binutils** A collection of programming tools for the manipulation of object code in various object file formats.

**pkg-config** Pkg-config is a helper tool used when compiling applications and libraries which helps you to insert the correct compiler options to find libraries.

**Debian packaging tools** Tools to create Debian software packages.

Process:

- Build the application using x86 target
- Launch the application on x86 target
  There is a helper shell-script on maemo rootstrap called `run-standalone.sh` which must be used when launching applications in scratchbox. The script sets the correct environment for the application to use the maemo application framework
- Test the application
- If needed, modify the source code and repeat

Debugging on x86 rootstrap

- Use the x86 target of the Scratchbox
- Launch the application in debugger
- Run and debug the application
- Modify and re-compile if needed

Debugging the application in x86 target can be done with regular Linux development and debugging tools, like GNU Debugger (`gdb`), valgrind, ltrace and strace. Some tools also provide a graphical user interface to be used for debugging.

Notice also that the `valgrind` tool is not available in ARMEL target, only in x86 target. Valgrind is a really powerful tool for memory leak detection, profiling etc.

If the debugging tool of your choice is missing, as the Scratchbox is practically a full Linux system, it is also possible to add a tool into it by compiling the tool for Scratchbox from the source code.

Other possibility is to run the application on Internet Tablet and debug remotely using `gdbserver` on Internet Tablet and `gdb` on PC environment, using either cable or wireless connection.

For more information about debugging, there is a comprehensive debugging guide on maemo.org website.

## 7.4 Cross-compiling for ARMEL

Cross-compiling the application for the Internet Tablet is really straightforward. Activate `ARMEL` target of Scratchbox and re-compile the application. There is no difference in the process of compiling the application to x86 or ARMEL targets. After compiling, your application binary is ready for the Internet Tablet architecture.

The ARMEL target should only be used for cross-compiling and packaging the applications for the Internet Tablet device, not for running and testing as the ARM CPU emulation of Qemu may not provide accurate enough emulation to finalise testing only on PC.

## 7.5 Running, testing and debugging applications on the Internet Tablet

Even though the SDK is quite accurately identical to the target environment of the Internet Tablet, it isn't 100% identical. Especially if your application is using some special hardware of the Internet Tablet, the application can behave differently in SDK than on the device.

Fortunately, testing the cross-compiled binary transparently on the Internet Tablet has been made possible from Scratchbox, using either SSH or a CPU transparency tool called `sbrsh` (Scratchbox Remote Shell). Connection to the Internet Tablet can be handled either by USB cable or wirelessly.

CPU transparency is a technique where ARMEL binaries are copied from the Linux PC side over the connection to the device where the binary is then executed natively by the Internet Tablets own ARM CPU. This is a handy way to test your ARMEL-binaries in a native ARM CPU instead of running them in Linux PC under the QEMU emulator. The QEMU emulator may not be identical with the real ARM device so using CPU transparency with the real device is a convenient way to test drive your ARMEL application. The graphical environment is still the Scratchbox (and virtual X server, Xephyr) environment.

- Launch the application transparently on the Internet Tablet
- Test the application
- If needed, modify and re-compile the source code and repeat

Check the maemo.org documentation for more information about CPU transparency.

It is possible to use `gdb` debugger remotely with a `gdbserver` running in the Internet Tablet. The application to be debugged is then ran in the Internet Tablet, which makes debugging results 100% accurate. This makes remote debugging the application using desktop PC possible, again the same connectivity as with the CPU transparency is needed when remotely debugging.

For more information about debugging, there is a comprehensive debugging guide on maemo.org website.

## 7.6 Application Packaging and Installing

Maemo uses the Debian package management system for installing and managing application packages and their dependencies. For end-user the actual package management is invisible and the application installation and removal in the Internet Tablet is done by Application Manager. The Debian package management system uses packages which consists of application binaries, optional libraries, meta data describing the package, dependencies to other packages and optional pre-install and post-install scripts. Packaging the applications is done with standard Debian packaging tools.

After creating the Debian package (creation is identical to the desktop Linux environment) the application is ready to be installed to Internet Tablet. Application is either copied to the device and installed using Application manager, or by placing the package into the `package repository` (essentially a web or FTP site containing application packages) and creating a `single-click install-file`. The single-click install-file eliminates the need for user to manually configure repositories into the Application manager, providing an easy-to-use way for end-user to install the application.

The site maemo.org/downloads contains a vast amount of applications, ready to be installed to Internet Tablet, and utilises the single-click install method.

The maemo Application Development course material contains more information how to create Debian installation packages for Internet Tablets.