WIRTSCHAFTSUNIVERSITÄT WIEN

Institute for Management Information Systems

Projektseminar aus Wirtschaftsinformatik (Schiseminar)

ao.Univ.Prof. Dr. Rony G. Flatscher

Wintersemester 2010/11

Seminararbeit

Introduction to Android Programming

vorgelegt von:

Student:	Dennis Robert Stöhr
Studiengang:	Bachelorstudium Wirtschafts- und Sozial-wissenschaften
Matrikelnummer:	0453244
Geburtsdatum:	29.08.1983
Adresse:	Am Hundsturm 12/16, 1050 Wien
Telefon-Nr.:	+43 676 4745320
E-Mail:	h0453244@wu.ac.at

Wien, den 20.01.2011

Abstract

This seminar paper provides an introduction into Android programming and can be divided into three parts.

The *first part* explains how to set up a development environment for Android, create a first example project and run it in an emulator. It also sheds a light on two basic tools (ADB and DDMS) that assist the programmer in the development process.

The *second part* gives details on the Android system architecture as well as the composition of a project. This includes the components of an application, how it is set up by and lives with the Android Runtime Environment as well as the project structure and files.

In the *third part* the five components of an application are put into one place in an exemplary way to illustrate their purpose (Activities, Intents, Services, Content Providers, Broadcast Receivers).

Table of Contents

List of Figures

List of Abbreviations

ADB	Android Debug Bridge
ADT	Android Development Tools
API	Application Programming Interface
APK	Android Package
AVD	Android Virtual Device
BSD	Berkeley Software Distribution
DDMS	Dalvik Debug Monitor Server
EDGE	Enhanced Data Rates for GSM Evolution
GPRS	General Packet Radio Service
IDE	Integrated Development Environment
JDK	Java Development Kit
PID	Process Identifier
SDK	Software Development Kit
SMS	Short Message Service
TCP	Transmission Control Protocol
UI	User Interface
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
VM	Virtual Machine
XML	Extensible Markup Language

1 Introduction

Android is an operating system for mobile devices. It was initially developed by Android Inc., which was acquired by Google in July 2005.

Market share for Android has been growing steadily. In the third quarter of 2010 “Android accounted for 25.5 percent of worldwide smartphone sales, making it the No. 2 operating system (OS)” according to Gartner analysts [GartPress]. Market analyst Canalys predicts that “Android will grow more than twice the rate of major competitors in 2011” [CanaPress]. The result of this process is a broad and growing user base that makes Android an attractive platform to build applications for.

From a technical perspective, “Android is an open source software stack that includes the operating system, middleware, and key mobile applications along with a set of API libraries for writing mobile applications that can shape the look, feel, and function of mobile handsets” [Meier, 2010].

Android applications are programmed in the Java language and make use of Java libraries which has been developed by Google's Android team. Therefore, at least basic Java skills are a prerequisite to program for Android.

Besides this seminar paper, the Android developers website [Dev] is a good place to start getting a general overview and also to find detailed information on all subjects related to Android, such as the architecture, framework design, the development environment, programming examples and many more topics.

While working on this seminar paper the following versions were used:

- Release number of the Android SDK: 8
- Installed Android API-Level: 9, revision 1
- Version of the ADT plugin: 8.0.1.v201012062107-82219
- Version of Eclipse: 1.3.1.20100916-1202
- Build id of Eclipse: 20100917-0705

2 Setting Up the Development Environment

2.1 Requirements

In order to start coding, the minimum software needed to be installed is Java and the Android SDK. The following instructions are for Microsoft Windows.

The Java Development Kit (JDK) 5 or 6 is recommended, which can be obtained from http://www.java.com/de/download/manual.jsp. The Android software development kit (SDK) is available for Windows, Linux and Mac OS X. It can be obtained from http://developer.android.com/sdk/.

2.2 Android SDK

End of 2010 the Android SDK is delivered via the ZIP-archive *android-sdk_r08-windows.zip* and needs to be extracted to a directory, e.g. *Z:\android-sdk-windows*. Figure 1 shows the SDK Manager, which can be started via the shortcut *SDK Manager.exe* (located in the root directory of the SDK). It must be started after extraction to select, download and install the latest SDK resources and to update the SDK and its components at a later date when desired. It is also used to create and configure AVDs (Android Virtual Devices) which are explained in greater detail in chapter 2.4.

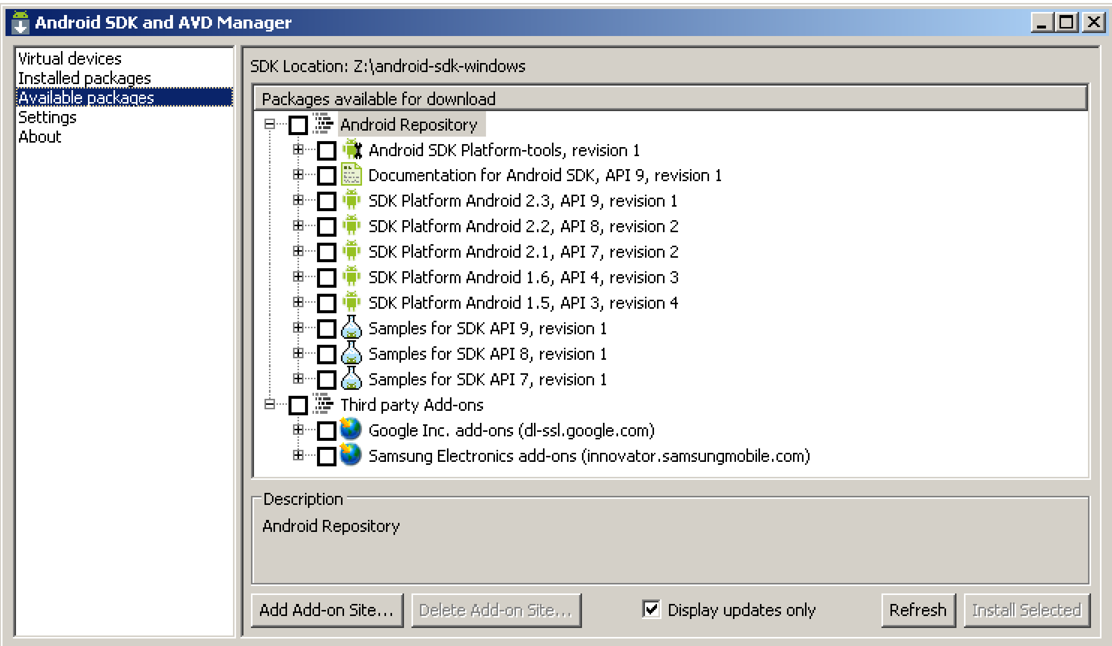

Figure 1: SDK Manager

Based on whether a basic or full development environment is desired, from a range of components can be selected in the “Available packages” tab. For example, at least one “SDK Platform” is needed in order to compile applications and create an AVD to run them on it [DevInst]. This should be the platform version of Android one want to develop for, maybe the latest one.

For the beginner it is recommended to set up a full environment, therefore selecting “Android Repository”, “Third party Add-ons” and then “Install Selected” to install all the packages. If no direct Internet access is available, a proxy can be configured in the “Settings” tab.

Figure 2 shows the directory contents of the Android SDK, Release 8 (December 2010).

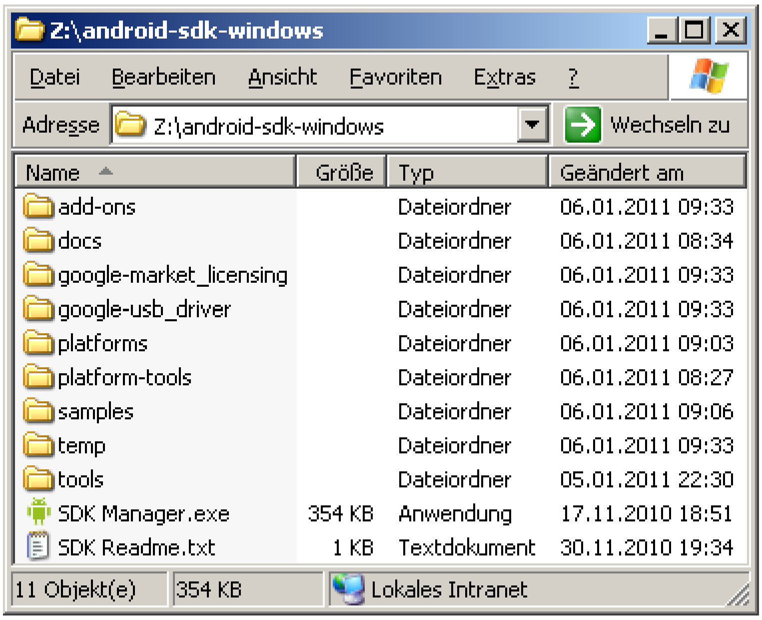

Figure 2: Directory contents of the Android SDK

The directory *platform-tools/* contains platform-dependent development tools like ADB (Android Debug Bridge). Chapter 3.3 provides more information on the ADB tool and it's usage.

The directory *tools/* contains platform-independent development tools like the SDK Manager, the emulator and the DDMS (Dalvik Debug Monitor Server). Chapter 3.4 provides more information on DDMS.

In *samples/* there is sample code and applications available for specific platform versions.

Different versions of the Android platform are located in *platforms/*. For example, the sub-directory *android-9* contains Android 2.3 Gingerbread. The number in the directory name represents the API Level, in this case 9.

See [DevInst] for more detailed information on the directory contents of the Android SDK and also for recommendations regarding the choice of components to be installed.

2.3 Eclipse

Installation of the JDK 5+ and Android SDK is basically enough to perform development. Yet a Java integrated development environment (IDE) can be very helpful because it aids the software development process. Features like syntax checking and highlighting, auto-completion and integrated debugging are available to the developer.

As an IDE, Eclipse is very popular among Java developers for its rich set of features and is directly supported for Android development via the Eclipse plug-in ADT (Android Development Tools), which on the one hand integrates access to and control of the SDK tools and features into Eclipse. On the other, it provides a New Project Wizard, a debug output pane, the DDMS as well as an Android code editor [DevADT]. Using Eclipse with ADT, creating and debugging applications is a lot easier and faster than working with a standard editor and the command-line based SDK tools.

Eclipse can be obtained from http://www.eclipse.org/downloads/. Like the Android SDK it comes as a ZIP-archive and needs to be extracted to a directory, e.g. *Z:\eclipse*.

The installation procedure for the ADT plug-in is as follows [DevADT]:

- Start Eclipse and select “Help” → “Install New Software” from the menu bar.
- Click on the “Add...” button on the top-right corner of the “Install” dialog. Then the “Add Repository” dialog appears.
    - For “Name” enter "ADT Plugin".
    - For “Location” enter the following URL:

      https://dl-ssl.google.com/android/eclipse/
    - Click the “OK” button.
- You return to the “Install” dialog and can now select the newly created “ADT Plugin” repository from the “Work with” drop-down list. Then select the check-box next to “Developer Tools” as shown in Figure 3.
- Click on the “Next” button, then you can review the items to be installed.
- Click on the “Next” button again, then you must review the Apache and BSD license terms.

- Finally click on the “Finish” button to initiate the download and installation process.
- After the installation has finished you are asked to restart Eclipse.

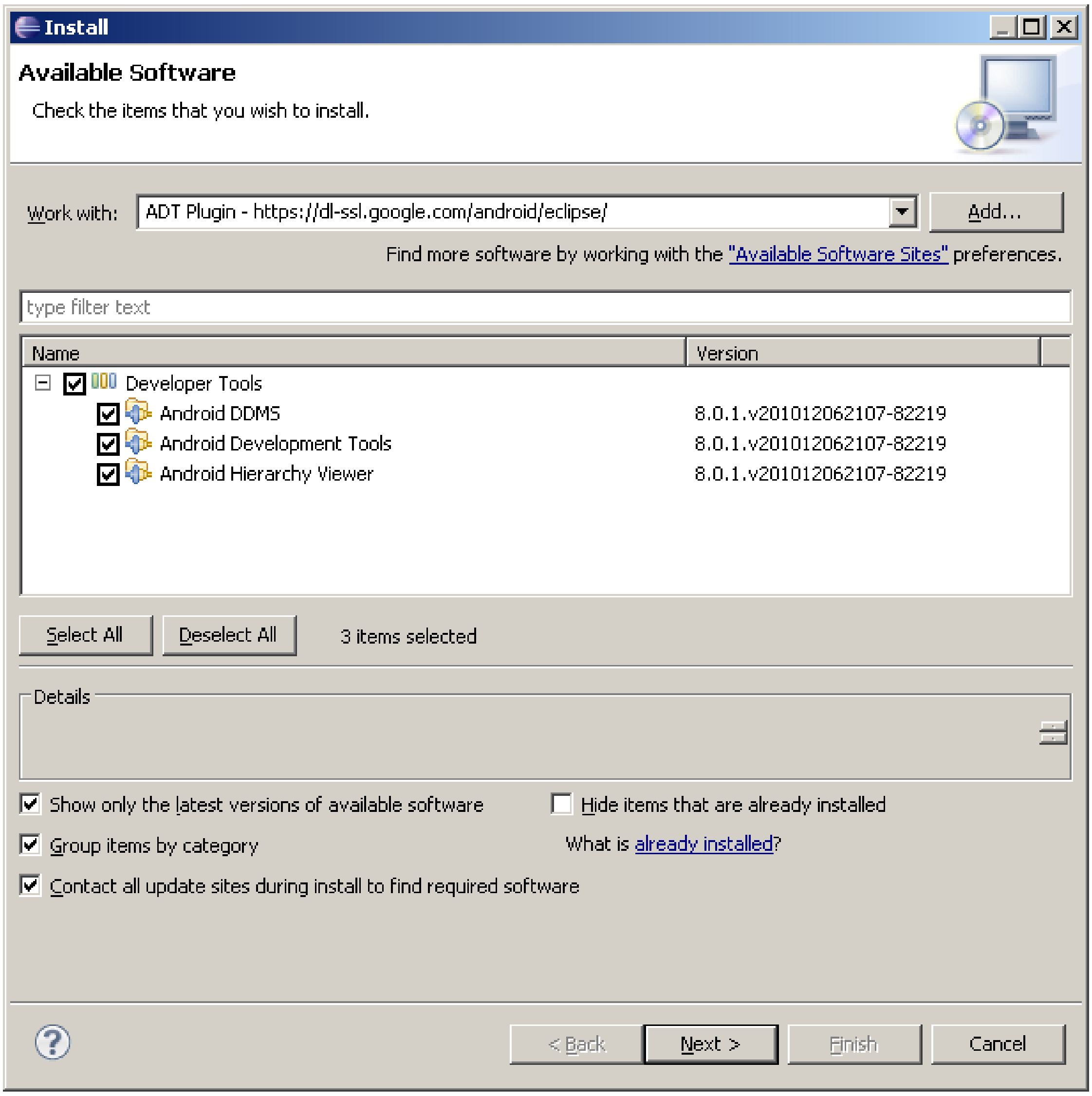

Figure 3: “Install” dialog with ADT Repository selected

After Eclipse has been restarted, what's left to do is to tell the ADT plugin the location of the SDK. Select “Window” → “Preferences” from the menu bar. Then click on “Android” on the left side of the “Preferences” dialog and type or browse the location of the SDK next to the field “SDK Location”. Click on the “Apply” button and a list of “SDK Targets” should appear against whose projects can be compiled and tested later (see Figure 4). Click the “OK” button to close the “Preferences” dialog.

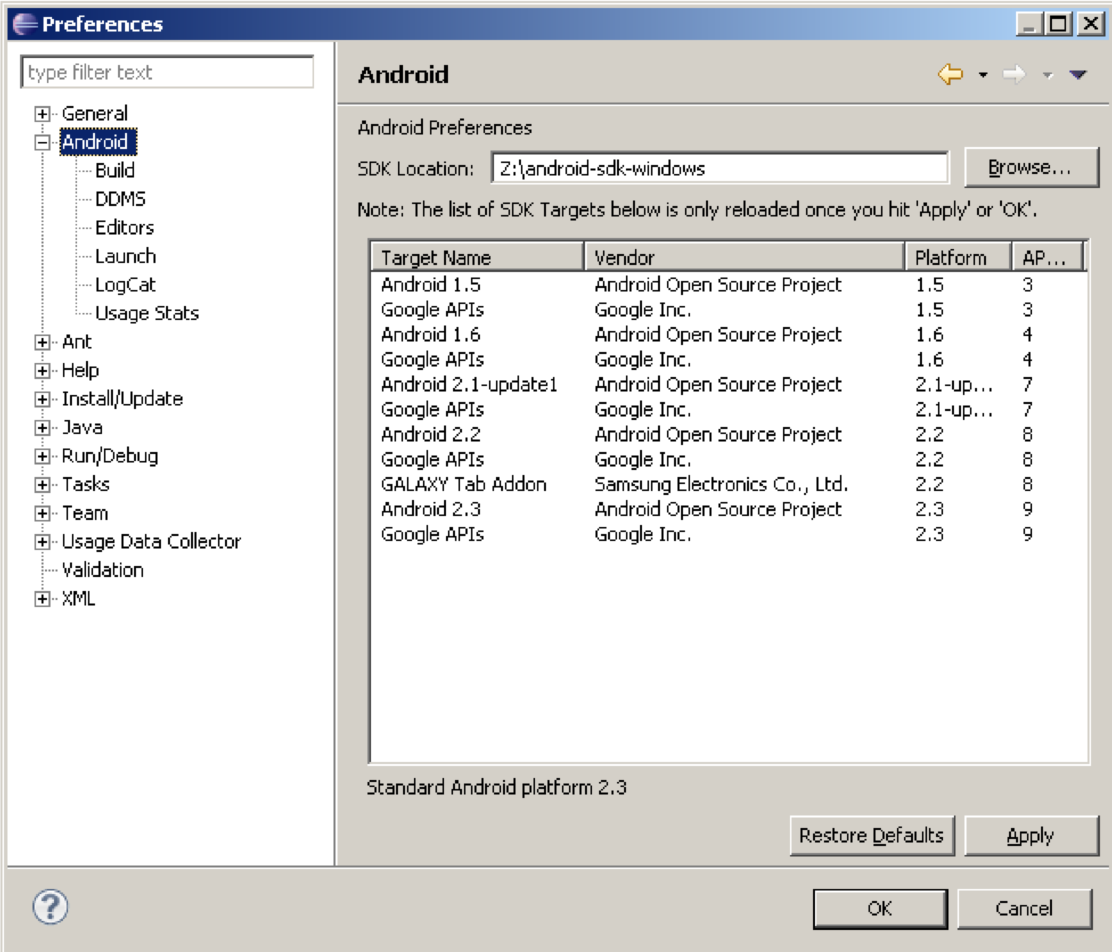

Figure 4: "Preferences" dialog of the ADT plugin

Now Eclipse has full Android development support. Android projects can be created, DDMS is available for debugging and managing devices (either virtual or physical ones) and applications can be launched directly from inside Eclipse like Java projects.

As a next step, it is recommended to set up an AVD if you don't have a physical Android device in order to be able to test your applications.

2.4 Android Virtual Devices

Development for Android doesn't necessarily require a physical Android device like a (smart-)phone. The SDK incorporates an emulator toolkit for creation of AVDs which can be used to easily run and test applications.

To set up a new AVD, start the SDK Manager and click on the "Virtual devices" tab on the left. On the right you see a list of existing Android Virtual Devices. To create a new one, click on the "New..." button. Fill in the required parameters and click on the "Create AVD" button at the bottom of the "Create new AVD" dialog (see Figure 5).

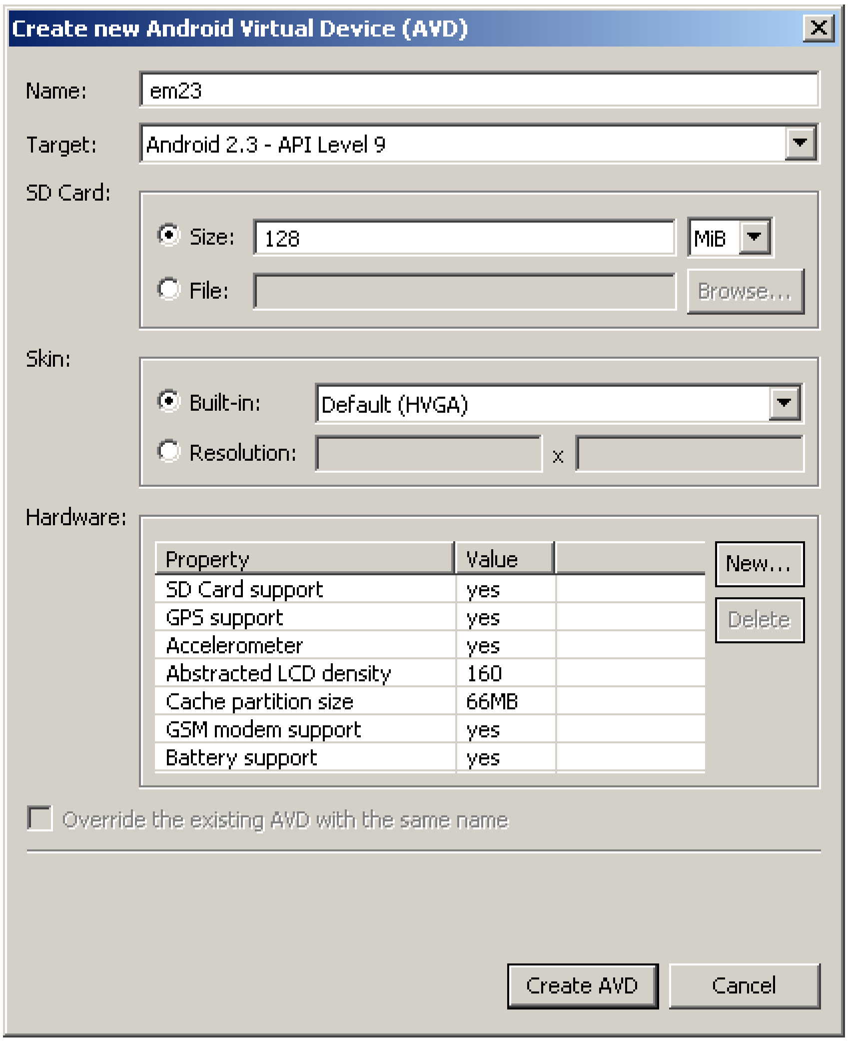

Figure 5: "Create new AVD" dialog of the SDK Manager with exemplary parameters

The "Target" defines the Android platform version (e.g. 2.3 Gingerbread) the AVD will run. Various display types with different resolutions can be selected via the "Skin" field.

There are also many hardware device emulations available like SD cards or GPS. Nevertheless in some cases it makes more sense to use a physical Android device, for example for testing touch-screen functionality or acceleration.

Chapter 3.2 will illustrate how to start up and use an AVD.

3 Features of the Development Environment

3.1 Creating a New Android Project by Example

To create a new Android project within Eclipse, select “File” → “New” → “Other” from the menu bar. Then click on “Android”, select “Android Project” and click “Next”. The “New Android Project” dialog appears (see Figure 6).

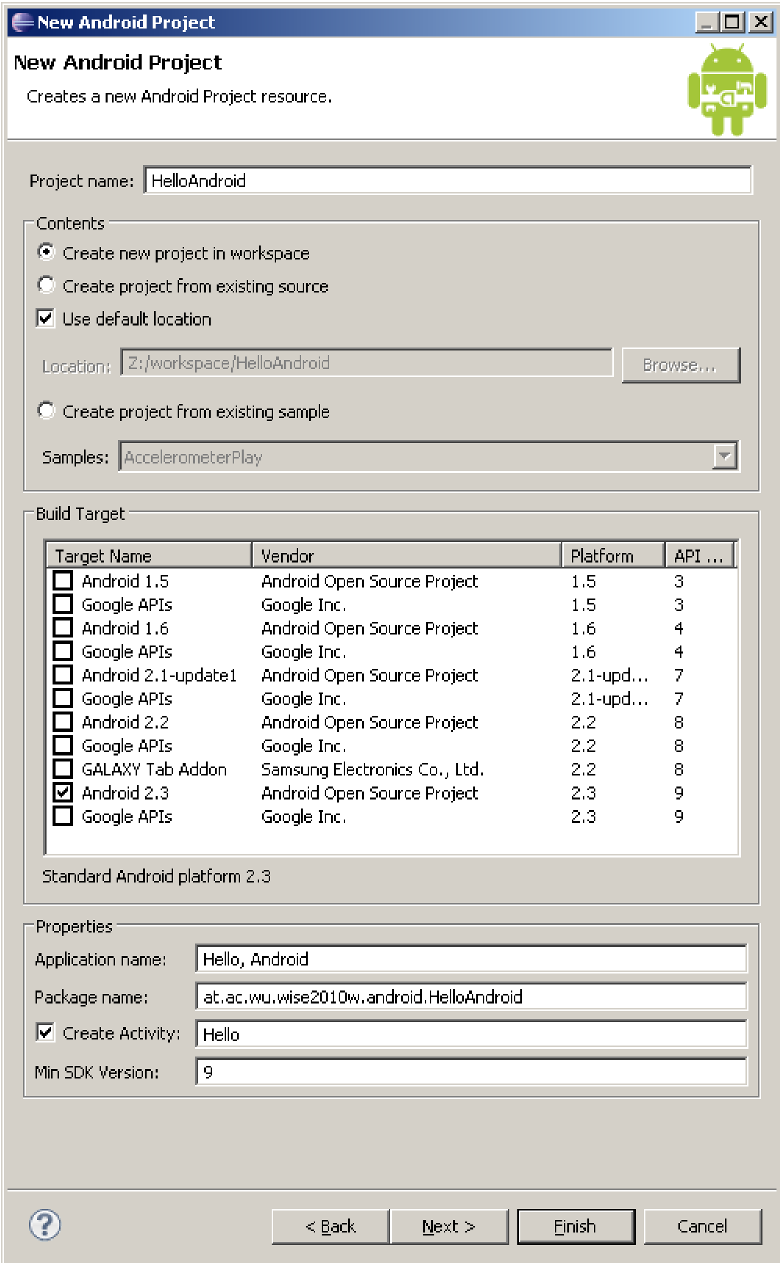

Figure 6: “New Android Project” dialog with parameters for HelloAndroid

“Build Target” specifies the Android platform version the application will be compiled against. This can be changed any time after the project was created.

“Min SDK Version” is the minimum API Level the application will need to run. In this case we entered “9” which means that the application will only run on Android 2.3 devices and not run on devices with Android Froyo or lower. This information is stored in the *AndroidManifest.xml* (located in the project's main directory) as an attribute called “android:minSdkVersion” of the element “<uses-sdk>”. More information on the manifest file is provided in chapter 4.4.

More details on “Activities” will be given in chapter 4.1.

Click the “Finish” button and the project will be created.

3.2 Running a Android project in an AVD

Right-click on the newly created HelloAndroid project folder in the Package Explorer on the left side of Eclipse and select “Run As” → “1 Android Application”.

This will do all the following in succeeding order:

- Compile and package the project files into a *.apk* file (Android Package)
- Start an AVD via ADB
- Transfer the APK file to the virtual device via ADB
- Start the application on the virtual device

Figure 7 shows the AVD “em23” (which we created in chapter 2.4) running the HelloAndroid application.

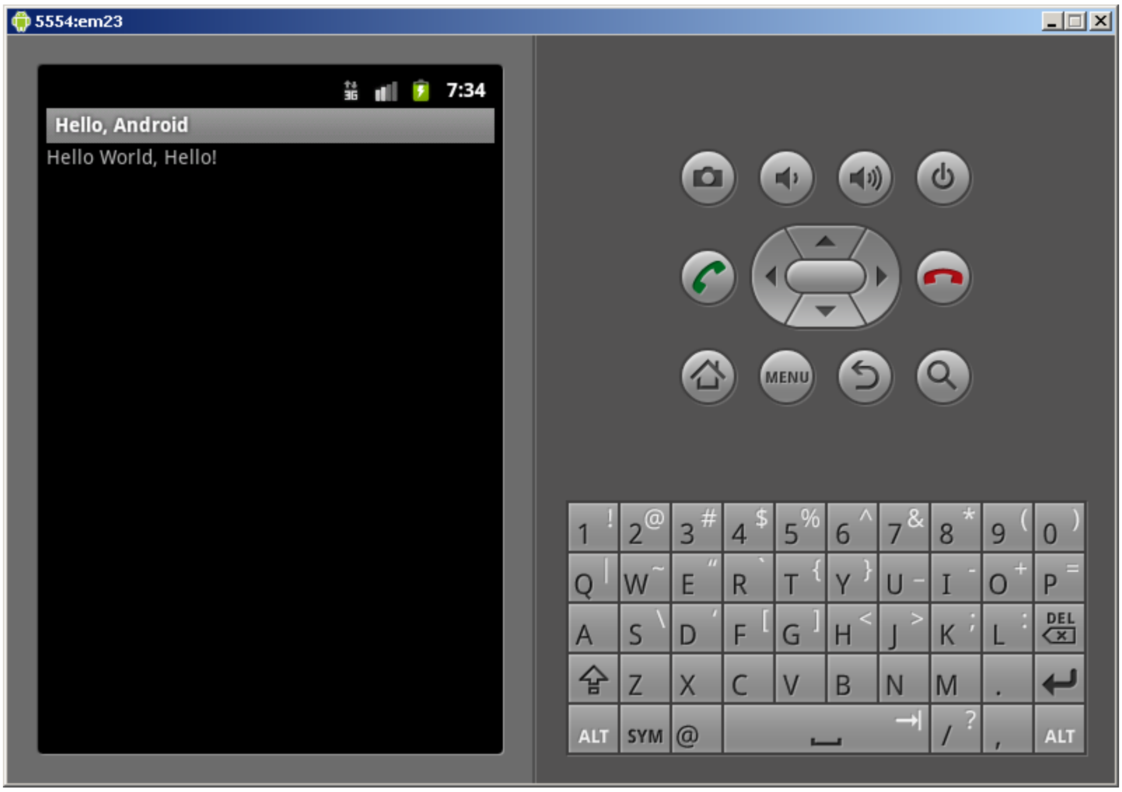

Figure 7: Android Virtual Device "em23" running HelloAndroid

Note that we did not write one single line of code, but what we see is Android's "Hello World" output. This is caused by the design specifications of an Android application. Layouts are defined in XML files and a *main.xml* layout file defining a standard layout is generated when creating a new project. Also strings are not hard-coded in Android, but put in a file called *strings.xml* which is also auto-generated, containing the above text "Hello World" plus the name of the activity by default.

More details on Layouts and Strings are given beginning with chapter 4.3.

3.3 ADB: Android Debug Bridge

As a client-server application, the task of ADB is to manage AVDs and physical Android devices. It is located in the sub-directory *platform-tools/* of the SDK.

When ADB is first started, it launches as a server daemon binding the local TCP port 5037 and listens for commands. When ADB is started again, it acts as a client.

ADB is a command-line tool which you don't need when developing Android applications in Eclipse, because "the ADT plugin provides a transparent integration of ADB into the Eclipse IDE. However, you can still use ADB directly as necessary [...]" [DevADB].

The syntax for calling ADB is as follows:

```
adb [-d|-e|-s <serialNumber>] <command>
```

- <command> specifies the ADB command. Examples are:
  - `devices` prints a list of all attached physical/emulated devices.
  - `logcat` prints log data to the screen.
  - install <path-to-apk> transfers and installs an application.
  - pull <remote> <local> copies a file from a device to to the PC.
  - push <local> <remote> copies a file from the PC to a device.
  - `shell` starts a remote shell in the device and connects to it.
- The *`-d`* option directs an ADB command to the only attached USB device.
- The *`-e`* option directs it to the only running emulator instance.
- The *`-s`* option directs it to the physical/emulated device with a specific serial number.

Figure 8 shows an example output of `adb logcat` when the em23 AVD is running.

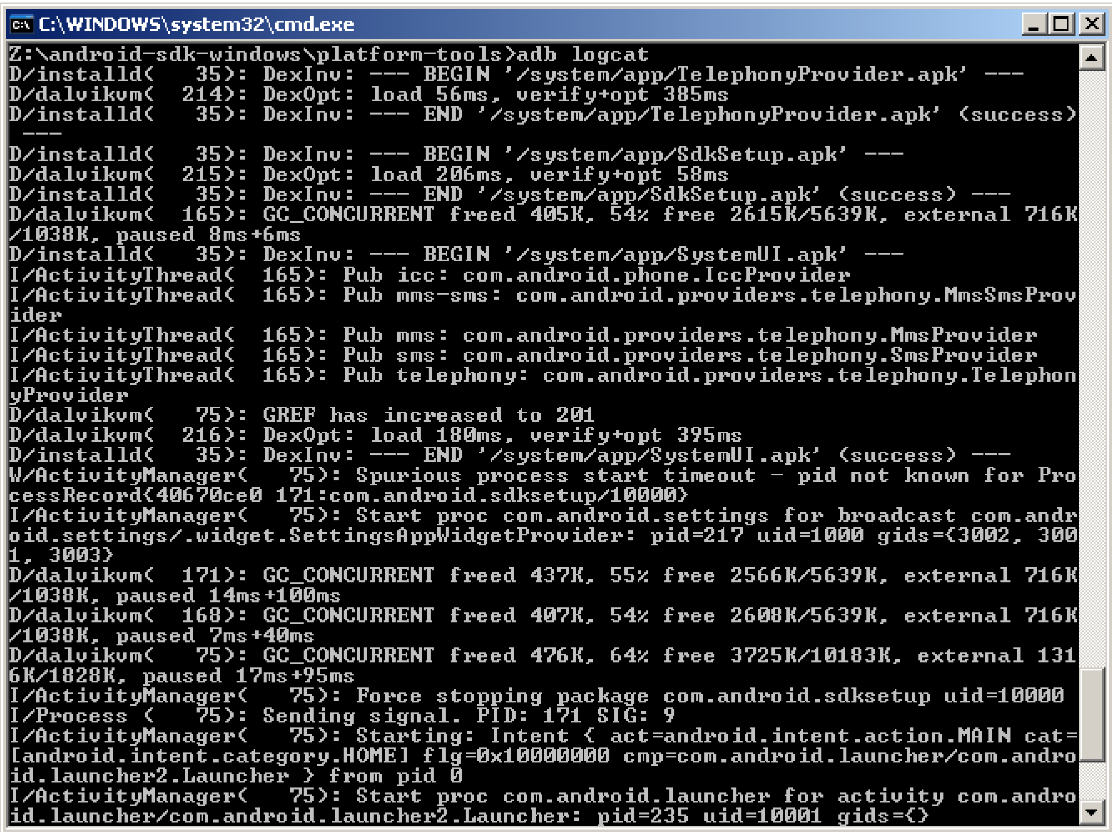

Figure 8: Output of `adb logcat` for the em23 AVD

Every log entry has a priority and a tag associated with it, e.g. D = Debug, I = Info, W = Warning priority. The tag is the relevant system component, e.g. “dalvikvm”.

Figure 9 shows a shell session created by the `adb shell` command on the em23 AVD with `ps` (listing all processes) executed.

```
C:\WINDOWS\system32\cmd.exe - adb shell
# ps
ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root     1     0     268    180   c009b74c 0000875c S /init
root     2     0     0      0     c004e72c 00000000 S kthreadd
root     3     2     0      0     c003fdc8 00000000 S ksoftirqd/0
root     4     2     0      0     c004b2c4 00000000 S events/0
root     5     2     0      0     c004b2c4 00000000 S khelper
root     6     2     0      0     c004b2c4 00000000 S suspend
root     7     2     0      0     c004b2c4 00000000 S kblockd/0
root     8     2     0      0     c004b2c4 00000000 S cqueue
root     9     2     0      0     c018179c 00000000 S kseriod
root     10    2     0      0     c004b2c4 00000000 S kmmcd
root     11    2     0      0     c006fc74 00000000 S pdflush
root     12    2     0      0     c006fc74 00000000 S pdflush
root     13    2     0      0     c00744e4 00000000 S kswapd0
root     14    2     0      0     c004b2c4 00000000 S aio/0
root     22    2     0      0     c017ef48 00000000 S mtdblockd
root     23    2     0      0     c004b2c4 00000000 S kstriped
root     24    2     0      0     c004b2c4 00000000 S hid_compat
root     25    2     0      0     c004b2c4 00000000 S rpciod/0
root     26    2     0      0     c019d16c 00000000 S mmcqd
root     27    1     248    152   c009b74c 0000875c S /sbin/ueventd
system   28    1     804    276   c01a94a4 afd0b6fc S /system/bin/servicemanage
r
root     29    1     3916   656   ffffffff afd0bdac S /system/bin/vold
root     30    1     3888   652   ffffffff afd0bdac S /system/bin/netd
root     31    1     664    248   c01b52b4 afd0c0cc S /system/bin/debuggerd
radio    32    1     5412   608   ffffffff afd0bdac S /system/bin/rild
root     33    1     63960  18308 c009b74c afd0b844 S zygote
media    34    1     20364  2584  ffffffff afd0b6fc S /system/bin/mediaserver
root     35    1     812    344   c02181f4 afd0b45c S /system/bin/installd
keystore 36    1     1796   540   c01b52b4 afd0c0cc S /system/bin/keystore
root     38    1     824    340   c00b8fec afd0c51c S /system/bin/qemud
shell    40    1     732    260   c0158eb0 afd0b45c S /system/bin/sh
root     41    1     4468   204   ffffffff 00008294 S /sbin/adbd
system   75    33    123360 26772 ffffffff afd0b6fc S system_server
app_12   161   33    75884  18216 ffffffff afd0c51c S jp.co.omronsoft.openwnn
radio    165   33    88252  19172 ffffffff afd0c51c S com.android.phone
system   168   33    76368  20332 ffffffff afd0c51c S com.android.systemui
app_1    235   33    79936  20536 ffffffff afd0c51c S com.android.launcher
app_9    291   33    76424  18592 ffffffff afd0c51c S android.process.media
app_13   339   33    75644  18024 ffffffff afd0c51c S com.android.email
app_26   357   33    73868  17020 ffffffff afd0c51c S com.android.quicksearchbo
x
app_28   391   33    72784  16080 ffffffff afd0c51c S com.svox.pico
app_30   414   33    73692  17604 ffffffff afd0c51c S at.ac.wu.wise2010w.androi
d.HelloAndroid
app_5    447   33    77344  20472 ffffffff afd0c51c S android.process.acore
app_15   458   33    75884  18708 ffffffff afd0c51c S com.android.browser
app_22   471   33    73364  16672 ffffffff afd0c51c S com.android.inputmethod.l
atin
root     485   41    732    332   c003da38 afd0c3ac S /system/bin/sh
root     488   485   888    332   00000000 afd0b45c R ps
# _
```

Figure 9: Terminal of the em23 AVD with `ps` executed

HelloAndroid is running with PID (process ID) 414 under user "app_30". It can be seen from the list that each process has its own unique user. It is the Dalvik virtual machine causing this by sandboxing each application it executes. Details on the Dalvik VM are provided in chapter 4.2.

3.4 DDMS: Dalvik Debug Monitoring Server

DDMS is a debugging tool which allows the developer to interrogate active processes, watch and pause active threats, explore the file system of connected devices, view logs generated by LogCat, simulate device states and activities (e.g. different kinds of network speed and latency like GPRS, EDGE or UMTS, simulate phone calls and SMS messages), and more. Generally speaking “it acts as a middleman to connect the IDE to the applications running on the device.” [DevDDMS]

It is available in Eclipse via the ADT plugin and has an own Eclipse perspective called “DDMS”. It can also be run from the command-line via the script *ddms.bat* which is located in the sub-directory *tools/* of the Android SDK.

Figure 10 shows the “Dalvik Debug Monitor” called via the *ddms.bat* script. Note that the SDK's sub-directory *platform-tools/* has to have been added to the system's PATH variable beforehand, otherwise *ddms.bat* does not find ADB and DDMS doesn't work.

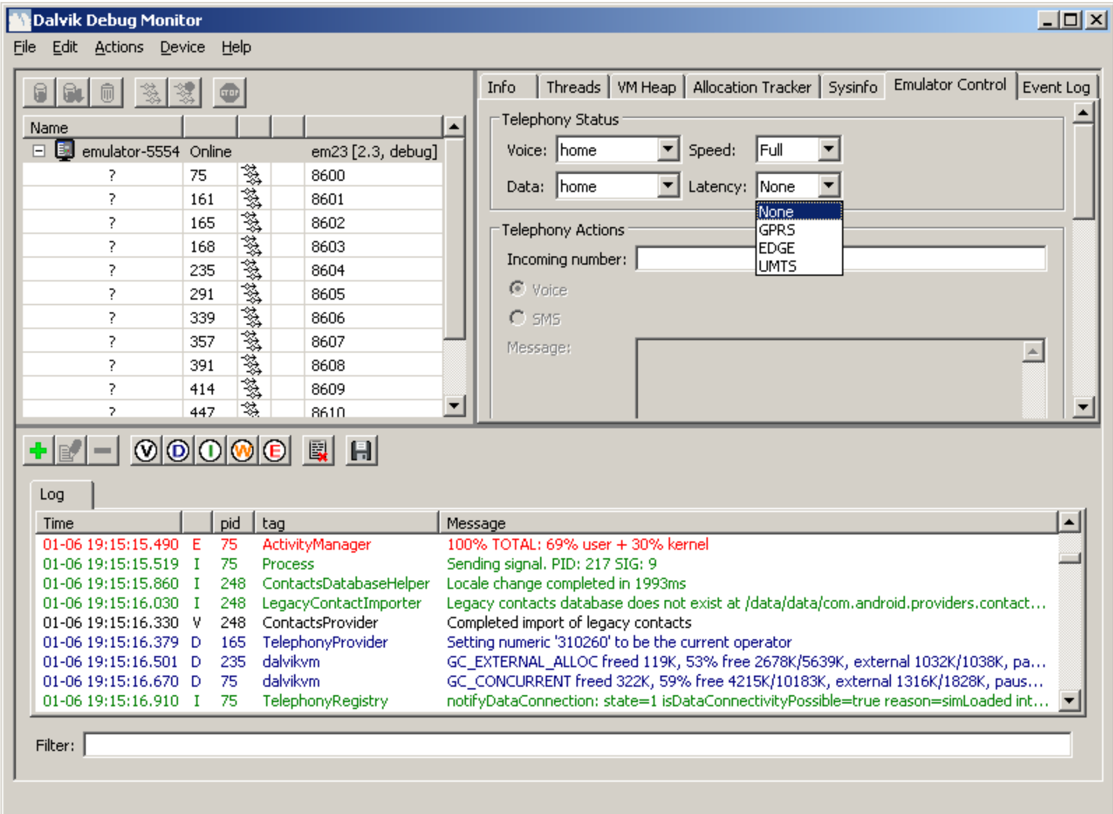

Figure 10: Dalvik Debug Monitor connected to the em23 AVD

Select “Device” from the menu bar to find other options like for example the “File Explorer”, “Dump device/app/radio state” or “Screen capture”.

On the top-right side of the window there are tabs allowing you to control and monitor the device and the applications running on it. In the “Sysinfo” tab for example you can monitor performance-relevant data like “CPU load”, “Memory usage” and “Wakelocks”.

Inside Eclipse, the DDMS perspective provides the same functionality, the only difference is that the window is embedded in the Eclipse IDE.

4 Basics of Android

4.1 Building blocks of an application

Five fundamental objects are defined in the Android SDK that are the building blocks of almost every Android application: Activities, Intents, Services, Content Providers and Broadcast Receivers.

4.1.1 Activities

"An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI […]." [DevAct]

Activities run through the activity lifecycle and have to take care of saving their states. More on this can be found in chapter 4.8.

4.1.2 Intents

"An intent is an abstract description of an operation to be performed. […] Its most significant use is in the launching of activities, where it can be thought of as the glue between activities." [DevInt]

An example would be that an application needs contact information and asks the system for this via an intent. The system knows which applications provide such contact information, because they registered themselves via an `IntentFilter`.

Another more illustrative example is an application that shows an URI to the user. When the hyperlink is clicked or touched, the application issues an intent to the system: "User wants to display URI: http://www.example.com". The user may get a prompt to select a browser if more than one has registered itself (see Figure 11).

Figure 11: An intent causes a Browser selection dialog [IntTut]

A very good introduction into the concept of Android intents is given by L. Vogel in "Android Intents – Tutorial" [IntTut].

The example in chapter 5 also includes an intent to launch a browsing activity.

4.1.3 Services

Like on other systems, a service is a program that runs in the background without direct user interaction [Burnette, 2010].

On Android, an idea would be to implement the base functionality of a music player as a service. If the user switches between applications, the music would not stop playing. Playback control would be implemented via one or more activities.

4.1.4 Content Providers

"Content providers store and retrieve data and make it accessible to all applications. They're the only way to share data across applications; there's no common storage area that all Android packages can access." [DevCont]

Standard Android content providers are for example the audio and video collections and personal contact information, which can be queried (if permissions to do so have been acquired).

4.1.5 Broadcast Receivers

“Broadcast receivers enable applications to receive intents that are broadcast by the system or by other applications, even when other components of the application are not running.” [DevBro]

This attribute can be defined in the *AndroidManifest.xml* (see chapter 4.4) for the components of an application.

4.2 Android Software Stack

On the lowest level of the Android architecture is the Linux Kernel 2.6, which “provides the hardware abstraction layer for Android, allowing Android to be ported to a wide variety of platforms in the future” [Burnette, 2010]. It is responsible for hardware drivers, memory and process management, networking and other operating system services. ADB allows to interact with the Linux system, e.g. via `adb shell` as shown in chapter 3.3. Figure 12 provides a graphical representation of the layer model.

One layer above the Linux kernel are the core C/C++ libraries, which are compiled for the specific hardware architecture used by the device Android is running on. For example the OpenGL and the SGL is available for 3D-/2D-graphics, SQLite provides database functionality, LibWebCore is a modern web browser engine (it powers the Android browser and the embeddable web view) and various media libraries “support playback of many popular audio and video formats, as well as status image files” [DevWhat]. The core libraries do not stand by themselves, but are called by higher-level programs.

Also on top of the kernel is the “Android Runtime” which is comprised of the Dalvik virtual machine and the core Java libraries. Like for the Java VM, bytecodes (the format is different from Java bytecodes) are generated from the sources and are then executed by the Dalvik VM on a mobile device, isolated in a sandbox with its own process ID and with an unique username. “Dalvik is optimized for low memory and allows multiple VM instances to run at once and takes advantage of the underlying operating system (Linux) for security and process isolation.” [Burnette, 2010]. Regarding the Android core Java libraries, which are sitting next to the Dalvik VM, it should be noted that they are different from Java SE (Standard Edition) or ME (Mobile Edition) libraries.

The “Application Framework” layer, which sits on top of the Android Runtime, enables the reuse and replacement of components. Those components are used by the core applications, for example the browser, calendar or contact list. Developers have full access to the APIs of the framework. For example “Content Providers” are objects which encapsulate shared data, an “Activity Manager” controls the life cycle of applications (more on life cycles in chapter 4.8), a “Resource Manager” takes care of the resources (images, sounds, textual data etc.) of an application and “Views” represents widgets which has an appearance on the screen.

The highest layer is the “Application Layer”. End users will see and interact with programs in this layer. Examples are pre-installed applications (like the browser or contact list), “apps” which were downloaded from an Android store or applications developed by us which we pushed onto the device via ADB.

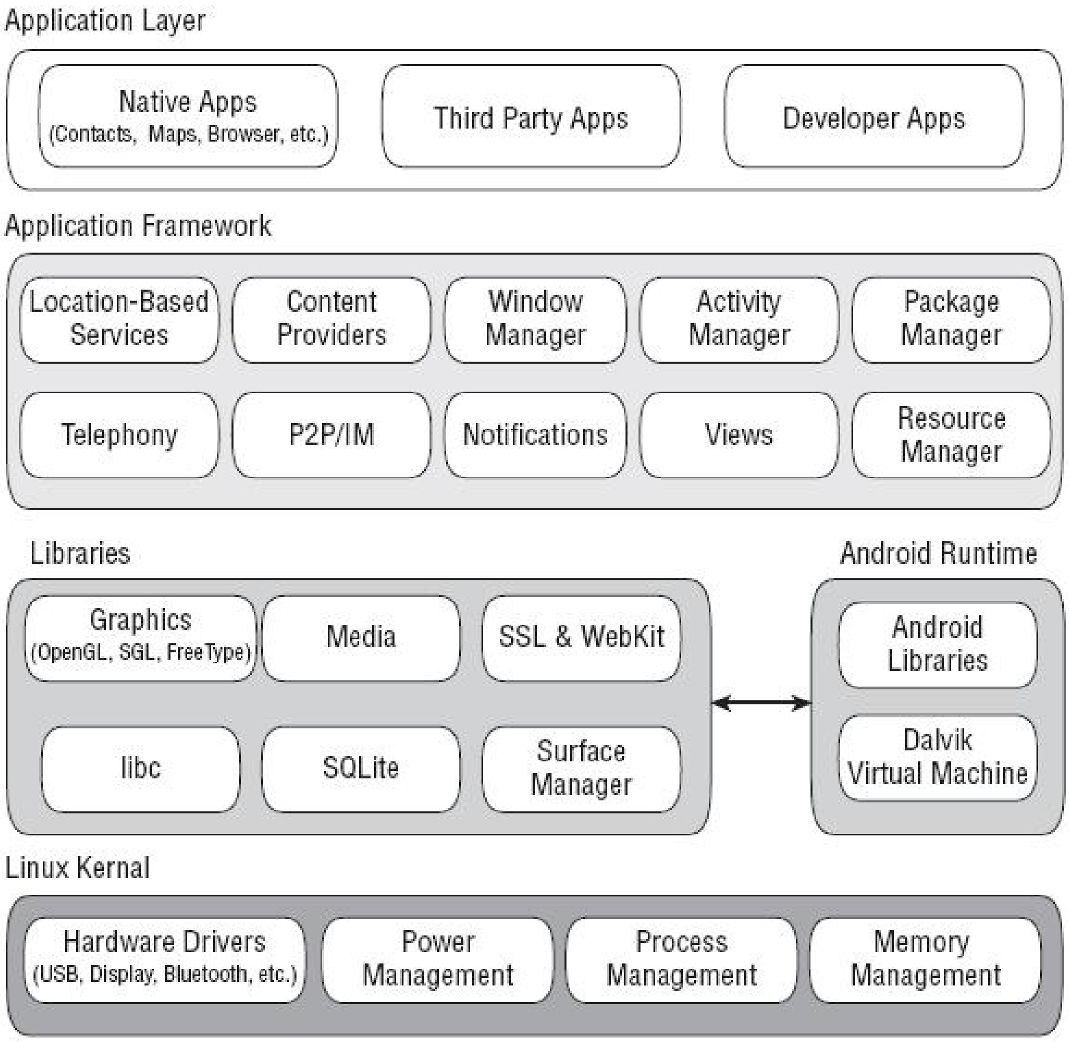

Figure 12: Android software stack [Samy, 2010]

4.3 Project Skeleton

When creating a project like shown in chapter 3.1, the SDK resp. Eclipse generates default project files automatically. Figure 13 shows this project skeleton.

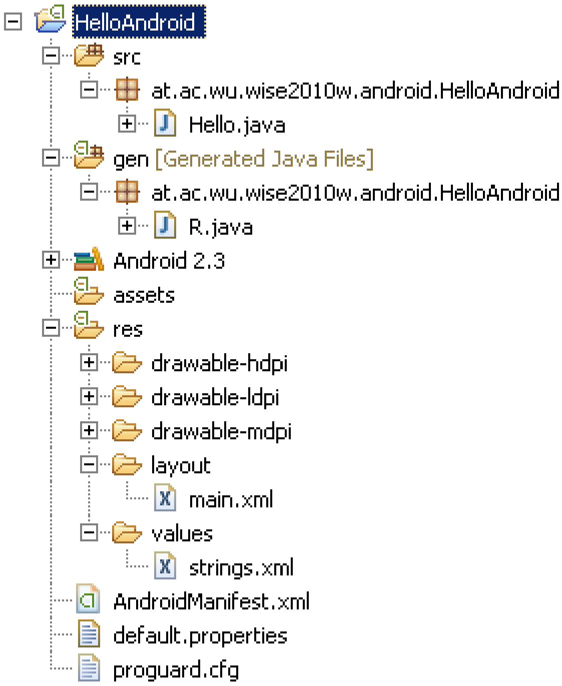

Figure 13: Project skeleton of the HelloAndroid project

The root directory holds the project's manifest file *AndroidManifest.xml* (see chapter 4.4). The *default.properties* file is used by the SDK's Ant build tool.

src/ contains the Java source code files.

gen/ contains only one file, *R.java*, by default. See chapter 4.7 for details.

res/ holds resources of the application (e.g. graphics, GUI layouts and values). See chapter 4.5 for details.

assets/ can hold other static files that are packaged with the application.

4.4 AndroidManifest.xml

The manifest file is the foundation for every Android application. Inside that file the contents of an application (activities, services, and so on – see chapter 4.1) are declared.

The root is a `<manifest>` element. Underneath it, the following elements are possible:

- `<uses-permission>` elements indicate the permissions the application will need to run properly.
- `<permission>` elements declare permissions that activities or services might require to use the application's data or logic.
- `<instrumentation>` elements indicate code that should be invoked on key system events (e.g. starting up activities).
- `<uses-library>` elements to hook in optional Android components.
- `<uses-sdk>` element indicates for which version of Android the application was built.
- `<application>` element specifies the application.

Figure 14 shows the manifest file for the HelloAndroid project created in chapter 3.1.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="at.ac.wu.wise2010w.android.HelloAndroid"
          android:versionCode="1"
          android:versionName="1.0">
 <application android:icon="@drawable/icon"
              android:label="@string/app_name">
  <activity android:name=".Hello"
            android:label="@string/app_name">
   <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
  </activity>
 </application>
 <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Figure 14: AndroidManifest.xml of the HelloAndroid project

Attributes of the `<manifest>` element are the package and version name (human-readable form) and version code (an integer) of the HelloAndroid application.

Attributes of the `<application>` element are the icon and application name. Both will be shown in the application launcher of Android devices.

The children of the `<application>` element represent the ingredients of HelloAndroid. There is one `<activity>` element with the name “Hello”. The `<intent-filter>` element defines under which conditions this activity will be displayed [Murphy, 2010]. In this case, the activity is started when the application is selected from the application launcher.

As pointed out in 3.1 the `<uses-sdk>` element defines the minimum API Level the application uses.

4.5 Resources

The *res/* directory contains resources of different types associated with the application. By default there are three types of directories:

- The *drawable/* directories hold graphics of low, medium and high resolution.
- The *layout/* directory holds layout information of user interfaces.
- The *values/* directory holds strings, color codes and other values.

4.5.1 Drawable

There are three sub-directories for graphics resources, *drawable-hdpi/* (high-resolution: 72x72 px and 240 dpi), *drawable-ldpi/* (low-resolution: 36x36 px and 120 dpi) and *drawable-mdpi/* (medium-resolution: 48x48 px and 160 dpi) [AndroidPIT].

By default, each one of these directories contains a standard Android logo of the respective dimension.

4.5.2 Layout

In *layout/* a file called *main.xml* resides. It specifies the layout of the application. Figure 15 shows the one of the HelloAndroid project.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Figure 15: main.xml of the HelloAndroid project

By default, a `LinearLayout` with a `TextView` is created. The `TextView` element represents a simple text label and its text is a reference to the string called “hello” that can be found in *strings.xml* (see chapter 4.5.3 for details on values).

A layout is a container for one or more child objects. It defines how these children are represented within its screen area. Other common layouts are `FrameLayout`, `RelativeLayout` or `TableLayout` [Burnette, 2010].

The `layout_width` and `layout_height` attributes of `LinearLayout` and `TextView` specify the size of the respective element. `fill_parent` means for example to take the full width or height of the parent element within which the element resides. `wrap_content` conversely means to take up only as much space as really needed by the element.

The `orientation` attribute with the value `vertical` has the effect that new components inside the element `LinearView` are added in vertical direction.

4.5.3 Values

Instead of hard-coding values, the Android framework calls for putting them into an XML file which resides in the *values/* directory. This eases portability to different languages, screen resolutions or the like because only value files have to be translated or adapted.

In case of providing string values for an application, it is the file *strings.xml*, which is also created by default. Figure 16 shows the file for the HelloAndroid project.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Hello!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

Figure 16: strings.xml of the HelloAndroid project

This is the place where the `TextView` element from chapter 4.5.2 gets its string value “Hello World, Hello!” from (see the output in Figure 7 on page 10) by referring to “@string/hello”.

4.6 Different screen sizes, input interfaces and languages

Android devices come in many different shapes and sizes. Even though Android tries to scale an applications user interface to fit the screen, it is not guaranteed that this works perfectly in every case.

To make sure that an application is displayed as intended by the developer, Android looks for certain directories that can host configuration files for specific device layouts.

Valid directory name qualifiers regarding display information are for example [DevScr]:

- `small`, `normal`, `large`, `xlarge` for screen dimensions
- `port`, `land`, `square` for screen orientation
- `long`, `notlong` for wider/taller screens
- `ldpi`, `mdpi`, `hdpi`, `xhdpi`, `nodpi` for screen pixel density
- `320x240`, `640x480` for screen dimensions

According to these qualifiers, the name of the directory holding graphics for low-density displays should then be: */res/drawable-ldpi/*

Besides screen property qualifiers, there are qualifiers which relate to input interfaces [Burnette, 2010]:

- `keysexposed`, `keyshidden`, `keyssoft` for keyboard availability
- `nokeys`, `qwerty`, `12key` for keyboard type
- `navexposed`, `navhidden` for navigation availability
- `nonav`, `dpad`, `trackball`, `wheel` for navigation type
- `notouch`, `stylus`, `finger` for touch screen type

Other qualifiers exist for languages and regions [DevLoc]:

- `fr`, `en-rUS`, `fr-rFR`, `es-rES` for the language and region (two-letter language code followed by optional two-letter region code which is preceded by a lowercase “r”)

According to this, strings in French for example should be put into the following file: */res/values-fr/strings.xml*

4.7 R.java

R.java is an automatically generated class file that holds references to the resources of an application. It should not be modified manually. Figure 17 shows the *R.java* file of HelloAndroid.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package at.ac.wu.wise2010w.android.HelloAndroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Figure 17: R.java of the HelloAndroid project

As can be seen, the class R has inner classes. Inside each of this classes are none, one or more static integer constants that hold hexadecimal numbers as references to the various data items.

The Android resource manager uses these references to load the real data, strings, graphics and other resources that are compiled into the application package.

4.8 Activity Lifecycle

An Android activity runs through different states during its lifetime. Certain callback methods are predefined which enable the activity to get prepared for a state transition, for example to preserve its objects, save data or states, refreshing elements and so forth. Figure 18 shows possible states and transition methods of an Android activity.

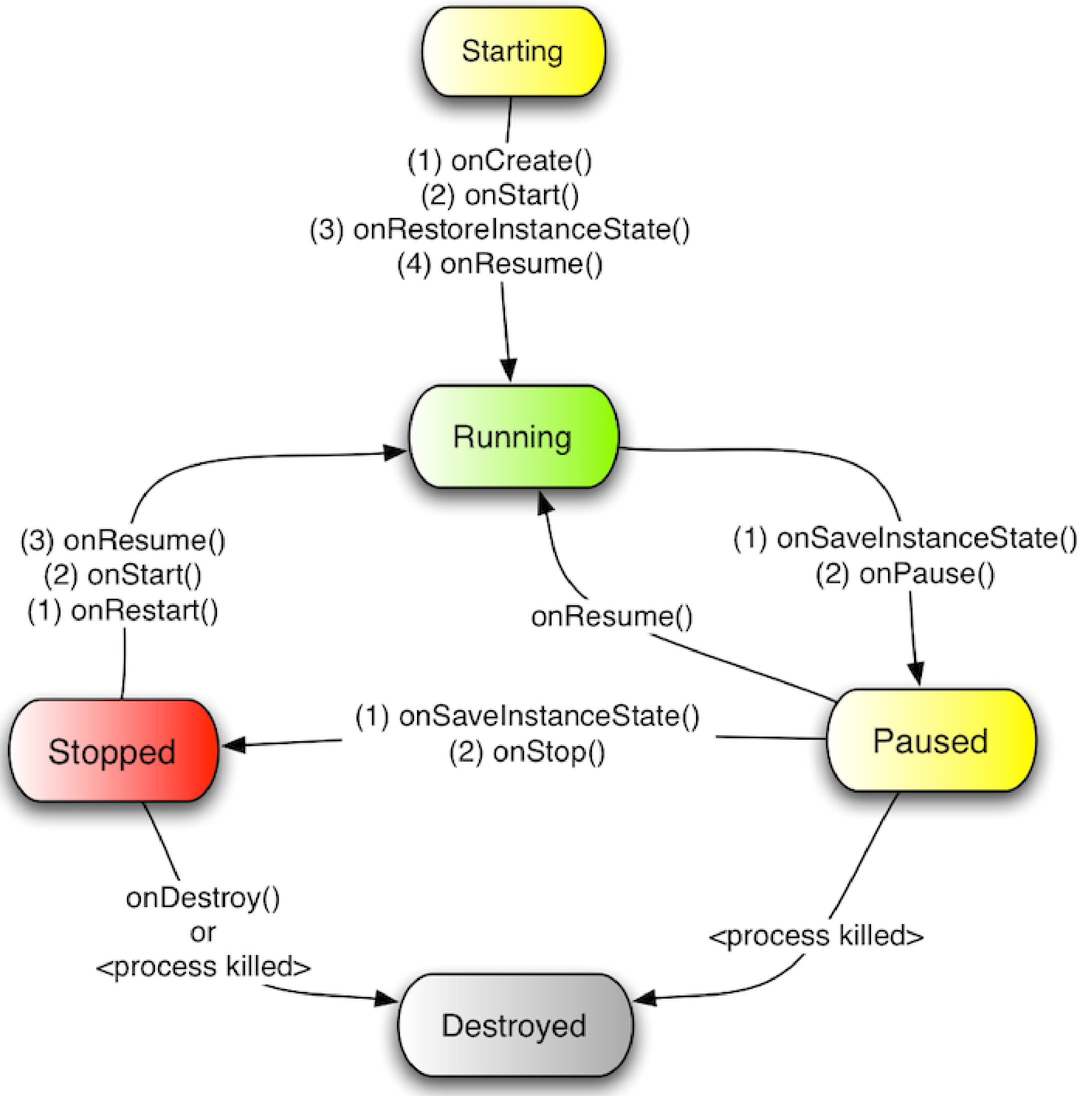

Figure 18: States and methods in the Activity Lifecycle [Gargenta, 2011]

Transitions between states happen for different reasons, one would be efficient memory management: The operating system sends an application to the background (= pausing the application) when another is getting focus (either by the system or the user who switches between different applications) in order to free up memory. For this transition, the activity's methods `onPause()` resp. `onResume()` (when returning to the application) are called when available.

Another example would be that the user selects an application in the menu of the Android device the first time since the device booted. The activity's `onCreate()` and `onStart()` methods are then called and the application enters the state “Running”, that means it is the one and only application which is presenting it's user interface and can be interacted with at the given time.

Running applications get the highest memory preference. Paused applications are also guaranteed a certain amount of memory unless there is no memory available for the running ones. Stopped applications could be destroyed at any point in time and get the least preference on memory, the main reason for this being caching for faster restarts.

Good knowledge of the Activity Lifecycle and keeping a focus on states and transitions when programming helps to improve performance and responsiveness, because launching an empty activity requires enormous resources, namely 3 million times longer than it takes the Dalvik VM to add a local variable [Mednieks, 2010].

5 Programming Example

The following example “BrowserIntent” is an application consisting of a single activity that displays a text field, a “Go” button and a status label. When the user enters an internet address (URI) into the text field and then touches the “Enter” key or the “Go” button, a browser will start and navigate to the specified resource. In addition, each time the activity runs through one of its life cycle methods, an entry is added to the status label.

From a technical perspective, the example shows how an intent is used to start an activity within another activity and illustrates the activity life cycle. It was partly taken from [Vogel, 2011] and extended by the author.

Figure 19 shows the projects manifest file (*AndroidManifest.xml*). The application has one activity, needs no permissions (these would be specified by the `<uses-permissions>` child-element of `<manifest>` as listed in chapter 4.4) and does not require permissions when used by other applications (specified by the `<permissions>` child-element).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
      android:versionName="1.0"
      package="at.ac.wu.wise2010w.android.BrowserIntent">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".BrowserIntent"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Figure 19: AndroidManifest.xml of the BrowserIntent example

Figure 20 demonstrates a feature of the ADT plugin. It shows the activity's UI layout as specified in the *res/layout/main.xml* layout file. Figure 21 shows the respective XML-code.

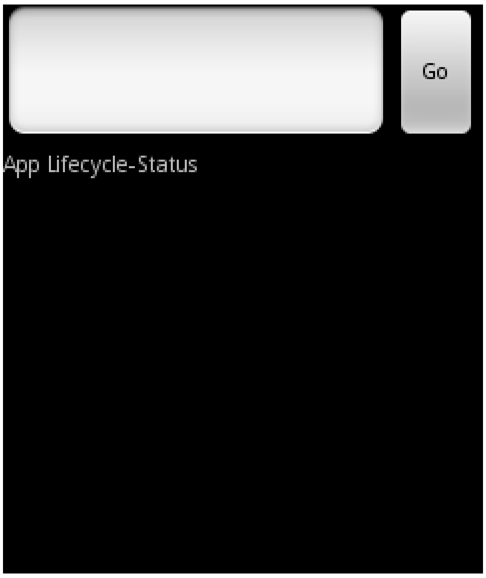

Figure 20: UI layout of the BrowserIntent example as shown by the ADT plugin

```xml
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent" android:layout_height="wrap_content">
   <EditText
      android:id="@+id/url_field"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1.0"
      android:lines="1"
      android:inputType="textUri"
      android:imeOptions="actionGo"
   />
   <Button
      android:id="@+id/go_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/go_button"
   />
</LinearLayout>

<TextView android:id="@+id/Status"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="App Lifecycle-Status">
</TextView>

</LinearLayout>
```

Figure 21: res/layout/main.xml of the BrowserIntent example

A top `LinearLayout` includes another `LinearLayout` that hosts the `EditText` field and “Go” `Button`. A `TextView` is the second component of the top `LinearLayout` that acts as a status label to which the information strings about called life cycle methods will be appended.

The `EditText`, `Button` and `TextView` elements have IDs associated with them so that they can be referenced inside the source code of the activity (*BrowserIntent.java*). These IDs are listed in the automatically generated *R.java* file.

Figure 22 shows the coding of the activity (*BrowserIntent.java*).

```java
package at.ac.wu.wise2010w.android.BrowserIntent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class BrowserIntent extends Activity {

   private EditText urlText;
   private Button goButton;
   private TextView status;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // Get a handle to all user interface elements
      urlText = (EditText) findViewById(R.id.url_field);
      goButton = (Button) findViewById(R.id.go_button);
      status = (TextView) findViewById(R.id.Status);

      // Activity life cycle info output
      status.setText("");
      status.append("\nonCreate() was called");

      // Setup event handlers
      goButton.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            openBrowser();
         }
      });
      urlText.setOnKeyListener(new OnKeyListener() {
         public boolean onKey(View view, int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_ENTER) {
               openBrowser();
               return true;
            }
            return false;
         }
      });
   }

   private void openBrowser() {
      Uri uri = Uri.parse(urlText.getText().toString());
      Intent intent = new Intent(Intent.ACTION_VIEW, uri);
      startActivity(intent);
   }

   // Demonstration of activity life cycle
   public void onPause() {
         super.onPause();
         status.append("\nonPause() was called");
   }

   public void onResume() {
         super.onResume();
         status.append("\nonResume() was called");
   }

   public void onSaveInstanceState() {
         super.onSaveInstanceState(null);
         status.append("\nonSaveInstanceState() was called");
   }

}
```

Figure 22: BrowserIntent.java

Because it is an activity, the BrowserIntent class extends the Activity class. Three class-wide fields are declared: `urlText`, `goButton` and `status`.

When the activity is started, the `onCreate()` method at first sets up the user interface via the method `setContentView()`. The parameter `R.layout.main` is a reference to the layout file *res/layout/main.xml* (see R.java for the hexadecimal reference). This means all elements with their stated attributes are set up as specified in the *main.xml* layout file. This method call is inserted by default.

Then the class-wide fields are initialized by creating handles to the instantiated UI elements from *res/layout/main.xml*.

After that, the class-wide fields are initialized via a method `findViewById()` with the respective references to the UI elements as parameters. Casting is necessary for the variable assignments.

Next, the text content of the status label is reset and a message is appended, saying that the `onCreate()` method was called. This way, the user gets informed about activity life cycle method calls. This is also done inside the overwritten methods `onPause()`, `onResume()` and `onSaveInstanceState()`.

Then action listeners are added to the text input field and to the "Go" button via the methods `setOnKeyListener()` resp. `setOnClickListener()`. The `onKey()` resp. `onClick()` methods define the following: If the Enter key is pressed within the text field or the button is clicked, the `openBrowser()` method will be called.

In the `openBrowser()` method the value of the text field is parsed to the URI format via the `parse()` method of the `Uri` class.

Next, a new intent is instantiated with the following two parameters: the action to be performed (in this case "`ACTION_VIEW`" which means "to display") and the data to operate on as an URI (in this case "`uri`" for the variable of type `Uri` to which the parsed input was saved).

Finally, the method `startActivity()` launches a new activity with the instantiated intent as the information carrier. This means it carries the description of the activity to be started.

Figure 23 shows the graphical state of BrowserIntent after the following procedure:

- An internet address was entered.

- The “Go” button was clicked.
- A browser was launched which displayed the specified resource.
- Returning to the application via the AVDs “Back” button.

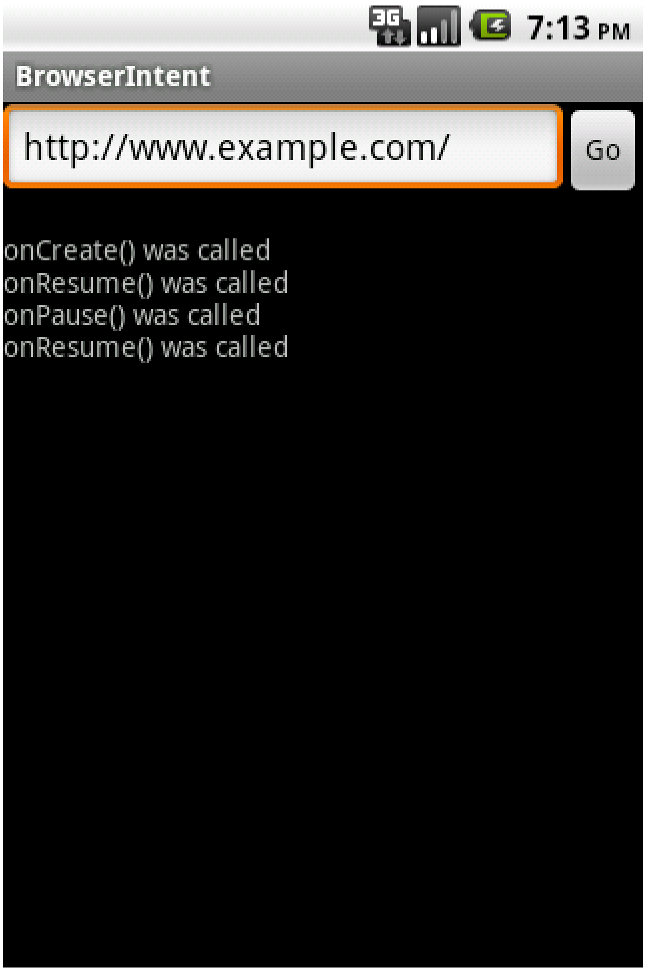

Figure 23: BrowserIntent executed in an AVD

6 References

[GartPress] N/A: Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent. Gartner website, 2011. http://www.gartner.com/it/page.jsp?id=1466313, as of January 16th, 2011.
[CanaPress] N/A: Canalys predicts Android will grow more than twice the rate of major competitors in 2011. Canalys website, 2011. http://www.canalys.com/pr/2011/r2011011.html, as of January 16th, 2011.
[Meier, 2010] R. Meier: Professional Android 2 Application Development. Wiley Publishing, Inc., 2010. ISBN: 9780470565520
[Dev] N/A: Android developers. Android developers website, 2011. http://developer.android.com/, as of January 16th, 2011.
[DevInst] N/A: Installing the SDK. Android developers, 2011. http://developer.android.com/sdk/installing.html, as of January 6th, 2011.
[DevADT] N/A: Developing In Eclipse, with ADT. Android developers, 2011. http://developer.android.com/sdk/eclipse-adt.html, as of January 6th, 2011.
[DevADB] N/A: Android Debug Bridge. Android developers, 2011. http://developer.android.com/guide/developing/tools/adb.html, as of January 7th, 2011.
[DevDDMS] N/A: Using the Dalvik Debug Monitor. Android developers, 2011. http://developer.android.com/guide/developing/tools/ddms.html, as of January 7th, 2011.
[DevAct] N/A: Reference (public class Activity). Android developers, 2011. http://developer.android.com/reference/android/app/Activity.html, as of January 9th, 2011.
[DevInt] N/A: Reference (public class Intent). Android developers, 2011. http://developer.android.com/reference/android/content/Intent.html, as of January 9th, 2011.
[IntTut] L. Vogel: Android Intents - Tutorial . vogella, 2011. http://www.vogella.de/articles/AndroidIntent/article.html, as of January 9th, 2011.
[Burnette, 2010] E. Burnette: Hello, Android. Pragmatic Programmers, LLC, 2010. ISBN: 9781934356562
[DevCont] N/A: Content Providers. Android developers, 2011. http://developer.android.com/guide/topics/providers/content-providers.html, as of January 9th, 2011.
[DevBro] N/A: The AndroidManifest.xml File (<receiver>). Android developers, 2011. http://developer.android.com/guide/topics/manifest/receiver-element.html, as of January 9th, 2011.
[DevWhat] N/A: What is Android?. Android developers, 2011. http://developer.android.com/guide/basics/what-is-android.html, as of January 7th, 2011.
[Samy, 2010] M. Samy: Introduction to Android App Development. Mobile Orchard, 2010. http://mobileorchard.com/introduction-to-android-development/, as of January 3rd, 2011.
[Murphy, 2010] M. Murphy: Beginning Android 2. Apress, 2010. ISBN: 9781430226291

[AndroidPIT] N/A: Android Anfänger Workshop. AndroitPIT, 2011. http://www.androidpit.de/de/android/wiki/view/Android_Anf%C3%A4nger_Workshop, as of January 7th, 2011.

[DevScr] N/A: Supporting Multiple Screens. Android developers, 2011. http://developer.android.com/guide/practices/screens_support.html, as of January 8th, 2011.

[DevLoc] N/A: Localization. Android developers, 2011. http://developer.android.com/guide/topics/resources/localization.html, as of January 16th, 2011.

[Gargenta, 2011] M. Gargenta: Learning Android. O'Reilly Media, Inc., 2011. http://ofps3.vz.oreilly.com/static/titles/9781449390501/images/04-ActivityLifecycle.png, as of December 29th, 2010.

[Mednieks, 2010] Z. Mednieks: Dissecting Google's Advice on Designing for Performance. O'Reilly Media, Inc., 2010. http://answers.oreilly.com/topic/1122-dissecting-googles-advice-on-designing-for-performance/, as of January 2nd, 2011.

[Vogel, 2011] L. Vogel: Android Tutorials. vogella, 2011. http://www.vogella.de/android.html, as of January 8th, 2011.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Wien, den 20.01.2011

Dennis Robert Stöhr