ECE155: Engineering Design with Embedded Systems Winter 2016

Lecture 5 — XML, Android & Eclipse

Jeff Zarnett, based on original by Patrick Lam

XML

You need to know a bit about XML (e**X**tensible **M**arkup **L**anguage) to build Android applications, so here's a quick description. By the way, this is examinable material.

For further reading, you can consult many resources on the Web. Here's one:

`http://www.w3schools.com/xml/default.asp`

XML in one line.

XML is a *structured document format*.

All XML documents therefore have the same format. However, XML has no intrinsic meaning. It just separates content from structure, and to be human readable. If you are familiar with HTML, there are a number of similarities, except where HTML (HyperText Markup Language) is specific to drawing web pages, XML is intended to be general and human-readable.

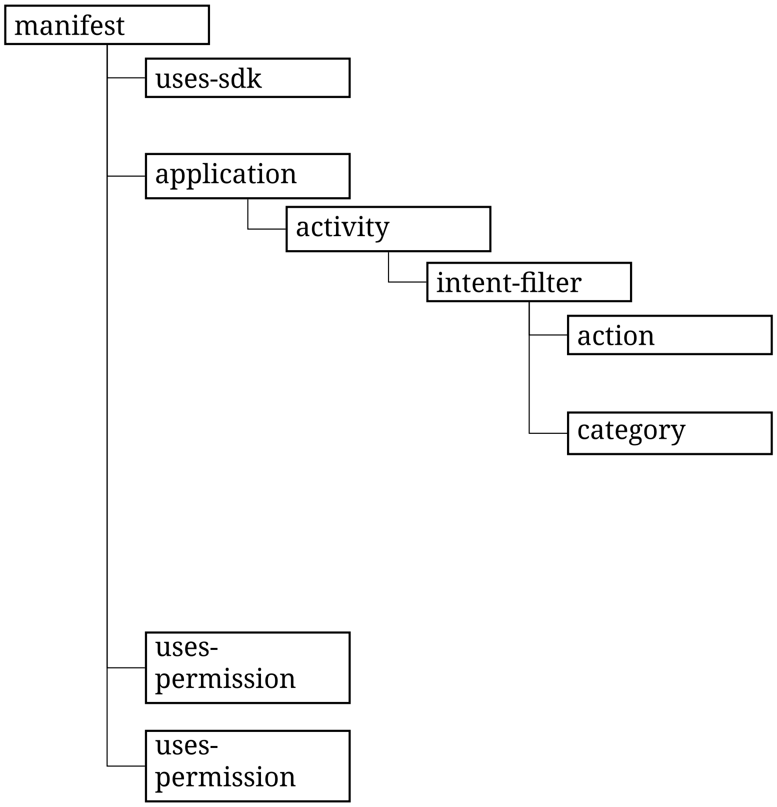
Every XML document begins with `<?xml version="1.0" encoding="utf-8"?>`, declaring that the document is XML. After that it has *tags*, which are elements of the XML document and appear between angle brackets like `<example>`. Tags are opened and closed, so that tag must be followed by `</example>`, which closes the `example` tag. We can also have "self-closing" tabs, a notational convenience where a tag is opened and closed all at once, such as `<approval/>` (the slash at the end indicates self-closing). A tag may also have attributes, like colour in this example: `<square colour=‘‘red’’/>`.

XML is intended to be human-readable and human writeable. Computers can read and comprehend XML by "parsing" the document. There are a number of well-known and well-developed XML parsers; use one of them if you need to programmatically examine some XML and do not write your own.

Let's look at an example of an Android manifest. It's an XML format document.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest     → root node
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="ca.uwaterloo.Lab1_plam"    → attribute, name is “package”, value is “. . . Lab1_plam”.
  android:versionCode="1"          → must quote all values, e.g. “1”
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="16" />   → self-closing tag

  <application   → application tag is nested within manifest tag
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity        → more nesting
      android:name="ca.uwaterloo.Lab1_plam.MainActivity"
      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-permission android:name="android.permission.READ_CONTACTS" />
  <uses-permission android:name="android.permission.WRITE_CONTACTS" />

</manifest>
```

Structure of XML files. XML is always tree-structured: at the top level, there is a *root* element. (Ignore the XML declaration.) So we can convert the textual form above into a tree. Tags must be well-nested: you can’t open a tag `<a>` and then close a tag `</b>` without closing `</a>` first.



Android

By now you have no doubt taken a look at the labs and seen that you need to implement them using Android, either on your own phone or one of the devices we provide in the labs. We will take some time to introduce you to Android programming.

Android Activity class

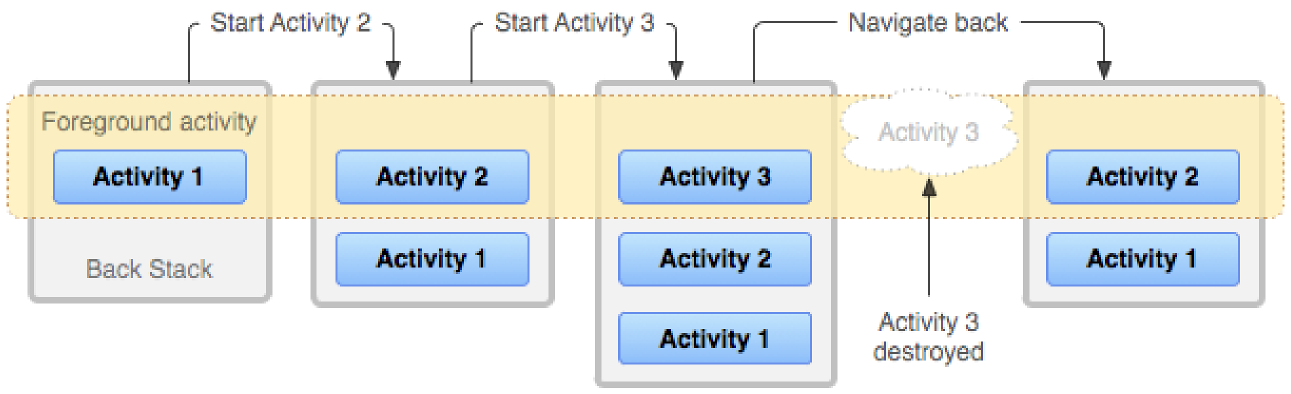
“An activity is a single, focused thing that the user can do.”

Usually, an Activity corresponds to a full-screen window which the user may interact with. For instance, the user may:

- set up a timer;
- read off sensor values; or
- make a phone call.

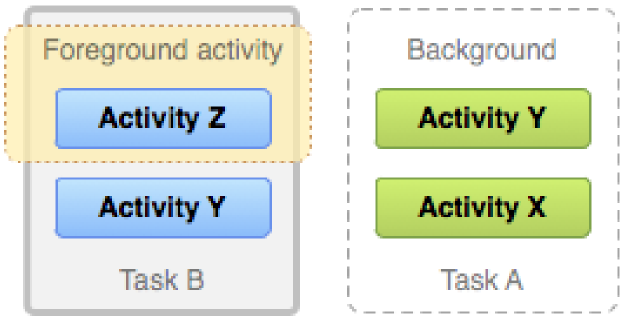
Applications may contain multiple activities, each of which corresponds to a thing that the user wants to do. Android organizes activities into tasks. A task consists of a last-in, first-out stack of activities, possibly from different applications.

Task Navigation: the Back button. The Back button pops the topmost activity off the stack and gets rid of it.



[Gui13b]

Task Navigation: switching tasks. It is also possible to switch between tasks. Switching tasks puts a different activity and its stack in the foreground, and puts the old activity in the background.



[Gui13c]

Doing Something in Your Activity

The most useful activity method, where you'll be writing a lot of code, is `onCreate()`. It gets executed when the activity starts. Typically, it will set up the user interface, namely:

- creating widgets;
- setting up event listeners;

Note that you must call `super.onCreate()`; this is taken care of for you in the autogenerated boilerplate code.

Retrieving Widgets If you've declared widgets in the XML file, you can use the `findViewById()` method to get a hold of them. Note:

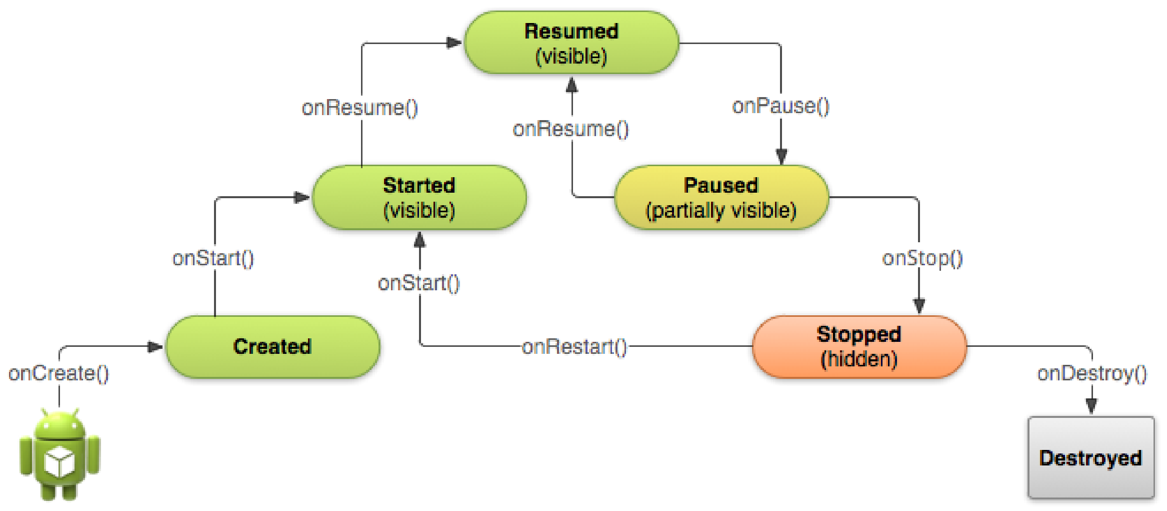
- you need to cast the return value, e.g.
  `tv = (TextView) findViewById(R.id.t);`
- you must save the XML file to get the right ids on the `R` object.

Programmatically Adding Widgets We ask you to add widgets programmatically in Lab 1. There are two steps:

1. Create the widget: `tv = new TextView(getApplicationContext());`
2. Add it to the Activity: `addView(tv);`

Activity Lifecycle

Although we've only talked about `onCreate()`, there are numerous other methods on `Activity` which Android calls at various times (inversion of control!)



[Gui13a]

Integrated Development Environments

We've seen a bit now about the basics of Android, but let's take a few moments to look at the tools used to work with Android.

In the Dark Ages

In the dark ages we used to write our code in an editor (`vi`, `emacs`, `notepad.exe`), then call the command line compiler. Actually, sometimes in later courses, you're still going to do this. When writing code in a plain text editor, it just looks like text. Then when you go to compile your code, the compiler prints out a whole pile of errors and you have to sort through and figure out what they all mean.

Then some clever people came up with an idea: what if we changed the tools to make it easier to write and make it less likely people will make a mistake? Hence, integrated development environments.

Modern IDEs

IDEs combine a number of tools in a single environment to improve programmer productivity. These tools include an editor, a compiler, and a debugger. The IDE concept has been around for over 25 years.

By now you have worked with Eclipse in the labs and presumably you used Microsoft Visual Studio when programming in C# for ECE 150. Eclipse is free software: you can download the code and modify it yourself. Also, it was initially developed by IBM in Ottawa.

Beyond the three core tools, IDEs can also contain support for collaboration (revision control systems, which we'll discuss); documentation and modelling, e.g. UML diagrams; and extensible programmer tools like autocomplete and refactoring. I'll talk a bit about autocomplete today.

Let's examine, however, some features of IDEs and how they make programming easier.

Syntax Highlighting The IDE highlights different syntax of the program, by making them different colours. It makes it easier to see and understand different parts of the code, and makes it possible to detect errors. For example, if you forgot a " (close-quote) character at the end of a string, then subsequent code after where you expect will be string-coloured, revealing the error. Similarly, comments are often coloured so you can see at a glance if half a line is commented out.

Templates. IDEs often allow you to start your project from a template, which is easier than starting from scratch. You've used the Android Application Project template. Some other useful templates:

- Android Test Project (you'll use these);
- Java Project
- Java Class
- Java Interface

Project Development Workflow. I'll just recap the steps here.

0. Figure out what you'll need to do.
1. Start a new project from a template or, more realistically, check it out from a version control repository.
2. Make the edits that you need.
3. Test your edits by running the application.
4. Debug your edits.
5. Commit your files to the version control repository.

Content Assist. Even if you type really fast, Eclipse can help you enter code more quickly, with the Content Assist helper. Start typing a name and hit Ctrl-Space (still ctrl-space if you're a Mac user). Eclipse will give you a choice of names to choose from, and you can choose the correct one. For further reading, see:

`http://wiki.eclipse.org/FAQ_How_can_Content_Assist_make_me_the_fastest_coder_ever%3F`

Content Assist can also help you with filling in templates. You can type "for" into Eclipse, hit Ctrl-Space, and choose the type of for loop you're trying to write. Choosing "for — iterate over array" inserts the template:

```
for (int i = 0; i < array.length; i++) {

}
```

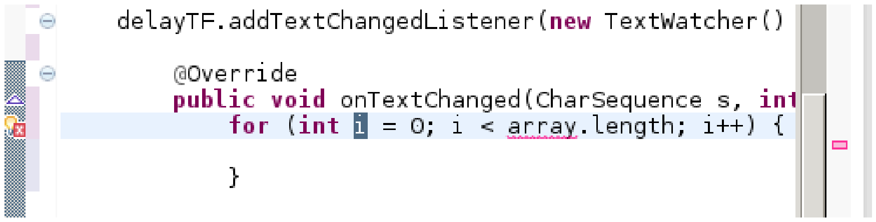
and puts your cursor at the "i" so that you can choose your own index variable. Of course, you still have to figure out what your code needs to do.

Continuous Compiling Instead of manually invoking the compiler manually, modern IDEs are constantly running the compiler when a file is changed. So as you're working, you'll see errors and warnings right away.

When you get a red (or yellow) squiggle underneath your code (see the figure below) there's an error (or warning). Eclipse and other very clever IDEs can sometimes suggest a solution, in a feature known as...

Quick Fix. The other really useful Eclipse feature is Quick Fix.

If the squiggle's corresponding editor marker bar (on the left) contains a lightbulb, you can Quick Fix it.



```
delayTF.addTextChangedListener(new TextWatcher()

    @Override
    public void onTextChanged(CharSequence s, int
        for (int i = 0; i < array.length; i++) {

        }
```

Put the cursor at or near the squiggle, hit Ctrl-1, and Eclipse will propose some fixes. They may be correct (or not).

You can read more about Quick Fixes in the Eclipse help:

Help >Help Contents >Java Development User Guide >Concepts >Quick Fix

Debugging You may have noticed that sometimes software doesn't quite do what you want it to do. We call this a *bug*. We'll discuss some content from [Zel09] (highly recommended), in particular ways to remove bugs from programs.

Three things have to happen before you can observe a bug:

1. A programmer puts a defect in the code—some code doesn't do what it's supposed to do.
2. This defect sets some program state (e.g. a variable) to an incorrect, or "infected" value.
3. The infected value has to propagate to program output to cause an observable failure.

Here is a high-level overview of the debugging process.

- Identify the bug and steps to reproduce it.
- Figure out the cause of the bug.
- Correct the bug.

Eclipse and other IDEs also have tools to help you find out what's going on in your program to help you find a bug. Probably at this point you've already used some of them in the labs (or even in ECE 150!). As an example, it's possible to pause the execution of your software, sometimes, and take a look at the variables while you have it paused.

Soon, we'll have a number of lectures on the subject of debugging, and explain some techniques (tactics) to help diagnose and fix problems. Eclipse can help, but the real action is inside your head.

References

[Gui13a] Android Developer Guide. Android reference: Basics: Lifecycle. http://developer.android.com/images/training/basics/basic-lifecycle.png, 2013. Online; accessed 18-January-2013.

[Gui13b] Android Developer Guide. Android reference: Fundamentals: Backstack. http://developer.android.com/images/fundamentals/diagram_backstack.png, 2013. Online; accessed 18-January-2013.

[Gui13c] Android Developer Guide. Android reference: Fundamentals: Multitasking. http://developer.android.com/images/fundamentals/diagram_multitasking.png, 2013. Online; accessed 18-January-2013.

[Zel09] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging, Second Edition*. Morgan Kaufmann Publishers, 2009.