The FAIMS Mobile User to Developer Guide: How to Build and Deploy Field Data Collection Systems for Arbitrary Methodologies

Version 1.0.20180326

Dr Brian Ballsun-Stanton
Russell Alleen-Willems
Adam DeCamp

Funded by the NeCTAR V005 Documentation grant for RT043 (FAIMS Project).
Typeset in ConTEXt.

Code for this project and the latest version of this document can be found at http://github.com/FAIMS/UserToDev.

1 Forward

This document will teach individuals familiar with the FAIMS system of efficient, comprehensive data collection how to create their own modules. No programming experience is required; all you need is a computer, some idea how to use it, and patience.

Because this guide starts from simple explanations and builds up to more complex applications, it's important to understand a section completely before you move on. Keep an eye out for our Test Your Knowledge questions. If at any point you can't confidently answer one, it might be a good idea to back up and re-read.

This document was funded by the NeCTAR Project grant V-005 as a follow on grant by RT 043. It was directed, produced, and typeset by Dr Brian Ballsun-Stanton with the assistance of Russell Alleen-Willems and Adam DeCamp as primary composers. The design of FAIMS Mobile was by Dr Brian Ballsun-Stanton and the primary source texts for this document, the *Cookbook* and *Program Logic Guide* can be found on our wiki: http://wiki.fedarch.org

Quiz 1.1 Test your knowledge of ... the forward!

1. What does this guide teach you how to create?
2. What should you do if you don't know the answer to a Test Your Knowledge Question?

2 What is FAIMS?

As far as your team's day-to-day usage is concerned, FAIMS is a replacement for less precise or efficient data-gathering tools such as paper forms, notebooks, photo logs, and spreadsheets. In a more technical sense, it's a system that lets humans intuitively collect data and computers meaningfully organize it.

The basic feature the FAIMS system revolves around is the **module**. The module is the part you interact with and the part that manages the data. These are what you're going to learn how to make, and it's important to understand them fully.

2.1 What does the "module" do?

From the perspective of users (which in this case will include all members of your team who aren't responsible for the module's technical aspects), the module appears to be a digital form that can be accessed from an app and used to submit field data. What the user sees, what data the user are asked to provide, and even what kinds of data the user are allowed to submit are all functions of the module's design.

> **When you design your module, you will have to think about what you do and do not need from your team.**

More importantly, FAIMS modules act as databases that collect data submitted by your team while remembering when it was originally collected, by whom, and how the information is related. Regardless of how or when your team's data are collected, relevant context will be preserved and nothing will be accidentally discarded "saved over" by later findings. Not only does this provide a convenient and comprehensive centralized repository of your team's data, it means that team

members who collect data far outside the range of internet or mobile service can be confident that when they're able to upload their work won't impact how the data are ultimately presented.

2.2 Why should you use a FAIMS database?

Why not stick with older analog methods?

When projects rely on combinations of multiple data-gathering methods, ranging from basic paper forms to supplemental map notes, annotations, GPS readouts, and photographs, it eventually becomes necessary to organize the data into physical files, folders, and electronic databases. Even when the data can be organized centrally, so that everything can be found in the same place and format, important considerations of context–where something was found, when it was found, who recorded it, how it relates to other pieces of collected data–are frequently lost in the transition. It's all too easy to scan a photograph and fail to capture the date penciled in on the border, or enter a paper form's submissions into a spreadsheet and have nowhere to insert a margin note, or digitize a stack of records and lose track of which folders and archaeological contexts they came from.

The goal of FAIMS is to replace those diverse tools with one kind of tool, Android tablets, running modules through which all data are collected efficiently and compiled automatically and completely. There's no longer as much need for integrating separate cameras, notebooks, and forms; instead you can collect all those kinds of data on one device. In addition to simple preselected or text inputs, you can can record and directly attach video, audio, and other files such as PDFs and electronic sketches collected with your tablet.

Because your data are being recorded digitally from the start, you'll be able to mostly skip transcription once you get back from the field. In fact, a key advantage of FAIMS is that you can synchronize your Android devices with your main database at any time you have a

network connection, and then use the updated data to plan where to focus your next day's field activities - while still out in the field!

FAIMS modules can be exhaustively customized to suit your team's needs, and to an extent may even be altered and automatically updated to all devices even after your team's project has begun. As you test out your new digital modules, you'll naturally find places where you'll want to add features like taking photographs on the tablet, tracking GPS locations of each record, and including additional files such as videos in your digital documentation. Once FAIMS becomes embedded in your field recording, you'll also discover ways, usually through the feedback of your field crews, that you can optimize your modules to allow recording to proceed faster and more efficiently. For example, you may include tracking for individual FAIMS user accounts so that field technicians no longer have to worry about including their initials, automating time and date fields, and organizing fields so that they appear in the order of your recording workflow.

For more details as to the tradeoffs between traditional methods, see <<<PARKER>>>

2.3 What are the benefits of using the FAIMS database over other databases?

While any database can be used to collect your team's data, FAIMS is designed with the specific needs and tools of archaeologists in mind.

For one thing, FAIMS treats the records you collect with an eye towards preservation. If you and another researcher enter in conflicting records, one of you doesn't "save over" the other; both of your contributions are preserved. Similarly, if a researcher alters a record, the project manager can still go back and see what it used to say. This can be particularly important when your team is collecting data in remote conditions and may not always be able to submit their findings in a timely manner.

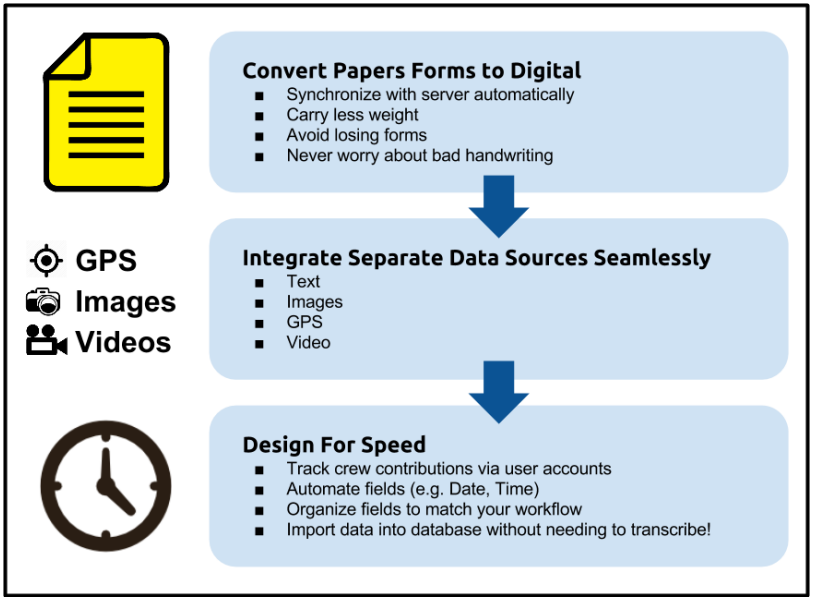

Figure 2.1 A prototypical workflow for a FAIMS Mobile module.

Furthermore, unlike simple databases which only allow for text input, FAIMS accounts for all the diverse tools the modern archaeologist has at their disposal, including video, photographs, audio logs, and GPS tags. You can attach these directly to your digital forms and store them seamlessly along with the rest of your collected data.

2.4 What are the benefits of using the FAIMS database over a spreadsheet?

The obvious reason is that spreadsheets are clunky and not really designed for records-keeping. Even if you put a painstaking amount of effort into creating a custom form for you and your team to use that clearly outlines where data should go and what form it should take, it's very easy for someone to accidentally undo that work by hitting the wrong key or saving at the wrong time. Furthermore, if you ever want

to look at an earlier version of your record, you'll need to have been saving multiple redundant versions.

But there's a more important reason, and it has to do with the difference between how humans like to format data and how computers have to interpret it. This reason is technical, but may be useful to understand once you begin formatting and designing your own modules.

Let's say a researcher turns in a typical data-collection spreadsheet like this:

SITE:	**SITE GROUND COVER:**	**ANIMAl:**	**COLOUR:**
Site A	Grasses	Bird 1	Red and White Wings
			Blue Plumage

Table 2.1 A sample spreadsheet table presenting complications with contextual whitespace.

We can immediately understand a lot about this spreadsheet from reading it. That's because we, as humans, can intuitively grasp things about it that a computer must be explicitly told.

We know that "Bird 1" is located somewhere called "Site A." We also know, without having to think about it, that a location like Site A can have multiple birds that belong to it or no birds at all. We can think of such birds we find there as instances of data (or a "record") which belong to another "record" called "Site A". Because there is a central piece of data to which zero or more pieces of data are hierarchically structured beneath, the relationship between Site A and Bird 1 is called a *parent-child relationship.*

We know also know from observing the spreadsheet that Bird 1 has red and white wings as well as blue plumage. Now, Bird 1 cannot have "zero or more" color in its wings or "zero or more" color to its plumage. It will obviously have a definable color for both, and the form expects this color to be described in straightforward terms. Similarly, the site

will obviously have definable ground cover; the only question is what the ground cover looks like. We call these *attributes of an entity*.

Because of our unspoken knowledge of these factors, we are able to make clear intuitive sense of the spreadsheet. The problem is that forms that make sense to humans often don't make sense to computers. This form, for example, is not digitally parseable. All a computer knows looking at those six cells is that at Site A there is a Bird 1 with Red and White Wings, but the quality of Blue Plumage has no connection to any other data. The attribute has no defined entity. The computer knows that something is Blue, but has absolutely no way of inferring further information from context, as humans invariably do when reading *and compiling* data.

When demonstrating *parent-child relationships* and *attributes of an entity* in a spreadsheet, inevitably human users will format data in a way a computer won't be able to meaningfully parse or store. The structure of FAIMS prevents natural human tendencies from making data unclear or digitally unmanageable.

2.4.1 Some framing questions for FAIMS Mobile

- How does FAIMS handle multiple users collecting data at the same time?
  FAIMS has no problem handling simultaneous usages or submissions. When two users create, edit, or delete records at the same time, FAIMS automatically takes note of who uploaded it and when and preserves both in the database. (Though if the two users are

working without talking to each other, the server will raise a "flag" for someone to review the submissions.)

- How does FAIMS handle international teams and multiple languages?
  We'll explain more in a section below, but when you design your module, you can include a translation file that will allow users to choose between differently-worded forms.
- How is data reviewed?
  If you just want to quickly review your data, and not export it to a format such as shapefile or json, you've got two options:
  1. You can navigate to your FAIMS server and use the Record View feature to look through individual records your team has made; or,
  2. using your device, you can navigate using your module to "table views."

2.5 Exporters: how to share data with others.

To get the data in a viewable, usable fashion, you'll need to find and download a type of program called an exporter. What form you want the data to be in decides which exporter you'll want to use. You can find exporters for formats like shapefile and json by visiting github.com/FAIMS/ and searching for the correct program. Look for something called "(x)Exporter or (x)Export," where (x) is the desired end format (e.g., "jsonExporter" or "shapefileExport").

On a PC, you can simply download the file from github. If you're using your UNIX virtual machine, you can do so by entering at the command prompt (denoted by $. Don't copy the $):

```
$   git clone https://github.com/FAIMS/shapefileExport/
```

Code 2.1 A command to use the program "git" to make a copy of the remote code onto your computer

...with the name of whatever exporter you want, in this example shapefileExport, in the final position.

Once you've got the exporter program, you're going to put it in a usable form. To do that with a PC, create a tarball from the exporter using a program like 7zip; if you're using UNIX, enter something like:

```
tar -czf shapefileExport.tar.gz shapefileExport/
```

Code 2.2 A command to: tell the `tar` command to compress, gzip, and save the folder "shapefileExport" to the file `shapefileExport.tar.gz`

Now, if you navigate on the server to your module, you'll see a tab at the top labeled "Plugin Management." Click that and you'll be brought to a page with the handy feature, "Upload Exporter." Choose the tarball you've just created and hit "upload." You now have an exporter permanently stored to your FAIMS server and may make use of it whenever you'd like.

From now on, whenever you'd like to use your uploaded exporter, navigate to your module from the main page on the server and click "export module." Select from the dropdown menu the exporter you'd like to use, review and select from any additional options, and click "export."

You'll be brought to your module's background jobs page while the server exports your data. After a few moments, you should be able to hit "refresh" and see a blue hyperlink to "export results." Clicking that will allow you to download your exported data in a compressed file.

Exporting data doesn't close down the project or prevent you from working any further on it, so feel free to export data whenever it's convenient.

Quiz 2.1 Test your knowledge of some of the fundamental concepts of FAIMS Mobile

1. Once you've set up your FAIMS module, will it be possible to alter it to suit the needs of your team?
2. Can you use FAIMS to collect audio or video files as part of your field data?
3. What's the difference between data as stored by FAIMS and data as stored by a spreadsheet?
4. If two researchers enter data at the same time, does one of them "save over" the other's work?
5. Can you view collected data via your tablet?

3 Making Your Own Modules

In the next chapter, you're going to make your module by following these steps:

- Learn what the components of a module are and what they do.
- Download tools that allow you to create module components (or "necessary files") based on a set of instructions.
- Learn how to write those instructions so that the necessary files you produce will fit together to assemble the module you need.
- Learn how to set up and operate the tools so that they'll follow your instructions and make the necessary files you need.
- Use software to hunt down and correct any mistakes you've made.
- Send the necessary files to the FAIMS servers and create a module you and your team can download and use.
- If you need to modify parts of your module, create new necessary files and send them to the FAIMS server. They'll replace the old ones and allow everyone on your team to update to the new, improved version.

3.1 The Parts of a Module

Modules are assembled by your FAIMS server from components called "necessary files[1]

Speaking practically, most "necessary files" are text files. Unless noted, they tend to end with the file extension .xml and can be opened and even edited with simple text editors, such as Notepad (although you may want to find something with a few more features, as we'll explain in the next section). Each necessary file serves a definite and distinct function in the final operation of the module.

[1] Brian's note: In our academic papers about modules we will sometimes use the term "Definition packets." The module is the product of the processing of the "necessary files" which as a set comprise the "definition packet.".

Your team members can use a module without ever learning exactly what necessary files are or what they do, but you'll need to become familiar with them if you plan to make a module or alter one already in use. Here are the kinds of necessary files you'll come across working with FAIMS.

3.1.1 Data Schema

This file, which should appear on your computer as "data_schema.xml", defines what kinds of data you want to record and how they're related to each other. We go into a little more technical detail about what a Data Schema is and does in the section below, "Tour from the Data Schema."

The Data Schema is one of the most fundamental and important necessary files of a FAIMS module. Unlike other necessary files, which can be replaced and updated even after the module's in use by your team, the Data Schema cannot be replaced. If for some reason your Data Schema no longer provides satisfactory results, you'll need to create a whole new module and instruct your team to transition over to it. This is the one part you should be absolutely certain you're happy with before you proceed.

3.1.2 UI Schema

The User Interface (UI) Schema, or "ui_schema.xml", defines what your module will actually look like and where your users will input their data.

3.1.3 Validation Schema

The Validation Schema, "validation.xml", defines what kinds of data your team *should* be collecting. It allows the module to "validate," or give a thumbs-up or thumbs-down to, your team's submissions to make sure the data being collected is thorough enough or makes sense.

For example, let's say your team is submitting data on handaxes they've excavated from a particular context. To complete your research goals you need to make sure that every time someone records a handaxe, they report how much it weighs in grams.

When you're designing your module, you will utlimately create a Validation Schema that ensures users must:

a. put something in "Weight of Handaxe" instead of leaving it blank
b. enter a number, not a phrase
c. list the weight in grams, not pounds, tons, or catties.

If a module checks a submission and discovers it isn't valid, it alerts the user who made the error, flagging the incomplete or problematic field as "dirty" and giving the user some idea what the problem is. However, the data are still collected and can be viewed or modified by the project manager.

3.1.4 UI Logic

The UI Logic performs a few functions. It tells a module's user interface how to behave, governs operations on the database, and facilitates interactions between FAIMS and devices such as GPS receivers, cameras, and bluetooth-compatible peripherals.

For example, when a record is created, the user who created it and a record of when it was created (also called a "timestamp") are automatically stored in the database. A UI Logic program can be used to 1) query the database to retrieve either of these points of data, and; 2) update the UI to display the retrieved data to the user.

3.1.5 Arch16n

You won't necessarily need to mess with your module's Arch16n file. It's there to allow you to provide synonyms and translations for the

"entities" in your module–useful if some of your team members speak a different language or use different terminology, in which case they would have their version of the module translated automatically.

3.1.6 CSS

The UI Schema defines the basic layout of your module's user interface, but the details, like how the entry fields and controls appear, are defined by the Cascading Style Sheet (CSS). You set these styles using the "ui_styling.css" file.

3.1.7 Picture Gallery Images

This part you handle more directly. Simply sort your images into folders, then put them all in a tarball (see "How do I share data with others?" for instructions). You can upload the tarball to the module generator directly, same as any other necessary file.

Quiz 3.1 Test your knowledge of some FAIMS jargon.

1. If data entered by a user isn't valid, is it collected?
2. What *necessary file* cannot be altered once a module has been created?
3. Which *necessary file* do you need to worry about if some of your team speaks English and some only French?

3.2 Tour from the Data Schema

We've already explained that the Data Schema describes what data your module is going to store–in other words, what your team is going to record and what you'll be able to export and analyze later. We've also explained that it defines the basic structure of the module and can't be altered once the module's been uploaded to your FAIMS server.

Because the Data Schema and its structure are very important to FAIMS, it may be useful (and for advanced coding, will be necessary) to understand what exactly is contained within the Data Schema.

Unlike the rest of the document, you don't need to understand this next part fully before you continue on to the next section. Instead, revisit this section every so often as you master a new portion of the User to Developer document and see how well you can apply your understanding of modules and their structure to this information.

A Data Schema contains "elements," or individual components that define some part of how the module works. The two kinds of elements you can find in a FAIMS Data Schema are called "Archaeological Elements" and "Relationship Elements," and they each do very different things to allow computers and users to collect and organize data.

3.2.1 Archaeological Elements

Archaeological Elements[2] are exactly what they sound like: they define an individual piece of archaeological data which can be collected. Each Archaeological Element has "property types" which define exactly what it represents and what kind of data are collected with regards to it.

[2] We will also use the term "Archaeological Entities or ArchEnts to refer to these. The term "Element" refers to *this specific file's definition* while entitiy refers to the thing living and working inside the module. Don't worry about it, but know that we use both terms."

They can contain multiple property elements, such as a name, value, associated file, picture, video, or audio file, as well as a description.

Codeblock 3.1 below is an example of an Archaeological Element for birds located within areas A, B, C, and D. Again: you don't really need to understand this yet, but it won't hurt to familiarize yourself with its structure.

3.2.2 Relationships

Archaeological Elements contain a lot of information about what is being collected, but they don't actually define how the data being collected is organized or related. They can be used to say "users enter their location" and "users identify what they found," but not "users identify what they found IF they are in a certain location." In other words, Archaelogical Elements explain how data is collected, but not how it is organized or related.

This is why the Data Schema also has something called a Relationship Element, a kind of element that describes how archaeological elements relate to one another; whether they are in a hierarchy, one contains another, or they are bidirectional. Relationship element can also contain child elements with more information about the relationship. These child elements can include descriptions, definitions of the parent and child entities, and multiple properties, such as "lookup," which lists controlled vocabulary terms.

An example of a Relationship Element can be found in CodeBlock 3.2[3]:

[3] Brian's note: While I built relationships to hold data all the way back in FAIMS Mobile 1.0, no one ever used them for that purpose. For now, we use them link Archaeological Elements usually with a directed 1:1 relationship.

```
<ArchaeologicalElement name="small">
    <description>
        A bird located within the area.
    </description>
    <property type="string" name="entity" isIdentifier="true">
    </property>
    <property type="string" name="name">
    </property>
    <property type="integer" name="value">
    </property>
    <property type="file" name="filename">
    </property>
    <property type="file" name="picture">
    </property>
    <property type="file" name="video">
    </property>
    <property type="file" name="audio">
    </property>
    <property type="timestamp" name="timestamp">
    </property>
    <property type="dropdown" name="location">
        <lookup>
            <term>Location A
                <description>This is the first location.
                </description>
            </term>
            <term>Location B</term>
            <term>Location C</term>
        </lookup>
    </property>
</ArchaeologicalElement>
```

Code 3.1 A demonstration archaeological element or “Archent”

```
<RelationshipElement name="AboveBelow" type="hierarchy">
    <description>
        Indicates that one element is above or below another
element.
    </description>
    <parent>
        Above
    </parent>
    <child>
        Below
    </child>
</RelationshipElement>
```

Code 3.2 A demonstration Relationship element or “Reln” allowing for a spatial relationship to be defined between two Archaeological Elements, one “above” the other.

You’ll understand more about how these elements are structured and what they mean after a thorough reading of "Coding Modules." In the meantime, as long as you understand in basic terms what’s contained within a Data Schema, it’s safe to proceed.

Quiz 3.2 Test your knowledge of element types

1. Which of the following are real types of element contained within the Data Schema: Archaeological Elements, Data Elements, Module Elements, Relationship Elements

4 Setting up the Development Environment

While it would be wonderfully exciting[4] to be able to jump right into module development, we must first get your computer ready to handle the "excitement".

Before you can code any new modules, you'll need to download some files and free programs. Some of those programs you'll need to configure to make coding new modules easier and more efficient. At the end of this chapter, you will be able to:

- Run the FAIMS Server, where your team members will download your module from and upload data to, on a virtual machine.
- Examine all the necessary files of a FAIMS module with a text editor.

4.1 Deploying the Virtual Machine

4.1.1 Hardware Assumptions

Every computer is different, and unless you're using exactly the machine we used when writing these instructions (an HP laptop with a Core i7 processor and 8GB of RAM, running Windows 7) you probably won't be able to follow each step exactly as it's written. Many of the differences will be very minor; an option might have a slightly different name or be located somewhere else on your computer, for example. When you find an option isn't present as we describe it, take a deep breath. Click around or use your computer's "search" feature to see if you can find the option or setting somewhere else. If necessary, web search the terms we use in our instructions along with the system you're using in

[4] Brian's note: Exciting for some folks.

order to find equivalents. A good rule of thumb for general computer configuration is: whatever you can't figure out right now, someone else has had the same problem and a third person has fixed it for them.

As a last resort, we offer technical support services. See the appendix for further information.

4.1.2 Before You Start

Before starting, you'll need to make sure you have enough room on your hard drive. We recommend about 25GB minimum for the server installation, as well as enough RAM to hold both your current operating system, the emulated machine, and any open programs in working memory. If you don't have that much space, delete files (especially large media files, like unnecessary videos) or uninstall programs until you're ready to begin. No good will come of trying to follow these next steps without enough room to work.

4.1.3 Installing the FAIMS Server and Virtual Machine (VM)

For various reasons, the FAIMS server is designed to run on a machine running an operating system called Ubuntu Linux 14.04 (as opposed to, say, Windows or OSX). You probably don't use Ubuntu, and you probably wouldn't like to use it all of the time. That's why our first task here is to set up a "virtual machine" on the computer you do use. Virtual machines let you emulate entirely different kinds of computer system without making any significant changes to your own–in this case, an Ubuntu Linux 14.04 machine. It's like you're installing an application that simulates an entirely different computer whenever you need it.

4.1.4 Enable VM extensions in your computer BIOS for better VirtualBox Performance (VT Hypervisor)

You'll have a much easier time running your virtual machine, as well as Android emulators you'll need later, if you first enable a feature specifically designed for this purpose called "VT Hypervisor."

To enable VT Hypervisor, enter your computer's BIOS or UEFI menu while booting your computer. This process differs a lot from computer to computer, but you should be able to find instructions here: http://www.howtogeek.com/213795/how-to-enable-intel-vt-x-in-your-computers-bios-or-uefi-firmware/

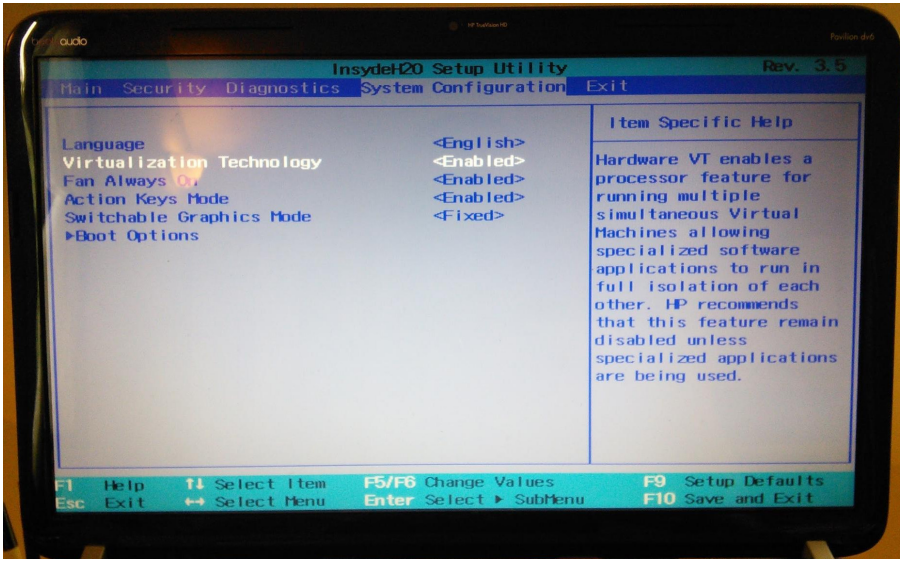

Figure 4.1 A rather literal picture of a computer's "BIOS" setup screen, showing the enabling of "VT-X extensions"

We found the VT setting under the "Virtualization Technology" option in the System Configuration menu. Again, unless you're using an HP dv6qt Laptop, yours will probably be somewhere else. This process

can feel a little intimidating if you're not used to messing with your computer on this basic a level, but relax: this step is perfectly safe.

4.1.5 Installing VirtualBox

Now you can download and install the VirtualBox client from: https://www.virtualbox.org/wiki/Downloads. This is what we're going to use to create a virtual machine that can run our Ubuntu Linux-based FAIMS server.

You probably won't have installed VirtualBox before, but on the off chance you have (say, if this isn't your first time trying this step) make sure you uninstall the old version **completely** before trying to install it again.

During installation, you may receive a couple Windows Security questions about Oracle drivers. Windows is naturally suspicious of this kind of installation, but nothing you're putting on your computer right now is dangerous. You can ignore each prompt individually or skip them all at once by selecting "always trust Oracle" in the popup window (Figure 4.2).

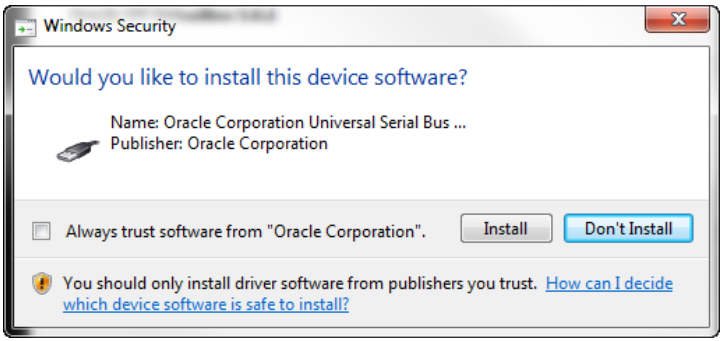

Figure 4.2 We are installing network device drivers here, so that your tablet can talk to this "Virtual Machine"

Once VirtualBox is installed, you can start the program. The default screen should look similar to Figure 4.3

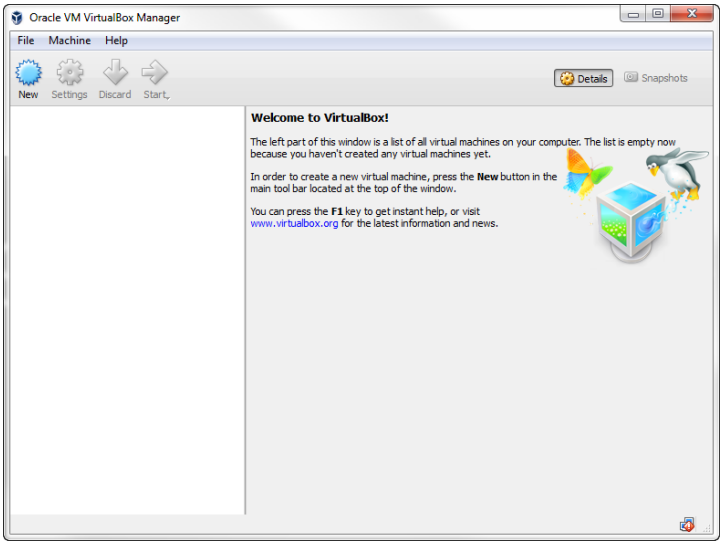

Figure 4.3 VirtualBox without any virtual... boxes. A box inside your computer which can hold boxes[5], but not cats.

If you need to take a break, this is a good stopping point[6]!

If you're ready to move on, download the FAIMS Server image (2.5GB) from: `FIXME`

The server image is packaged as an ".ova" file, which is a type of file that can be opened by VirtualBox. This file is the complete package: it represents an Ubuntu Linux 16.04 computer system that has the FAIMS server set to automatically configure and run. If you would like a server prepared for you, contact `support@fedarch.org`.

[5] We call a computer a box because the old fashioned tower that older or powerful computers come with is usually referred to as the Box by tech types

[6] Time for coffee! It's *always* time for coffee.

Before using the image, you'll need to unzip the compressed zip file using your favorite archival program (e.g. *7-ZIP*, *ICEOWs*).

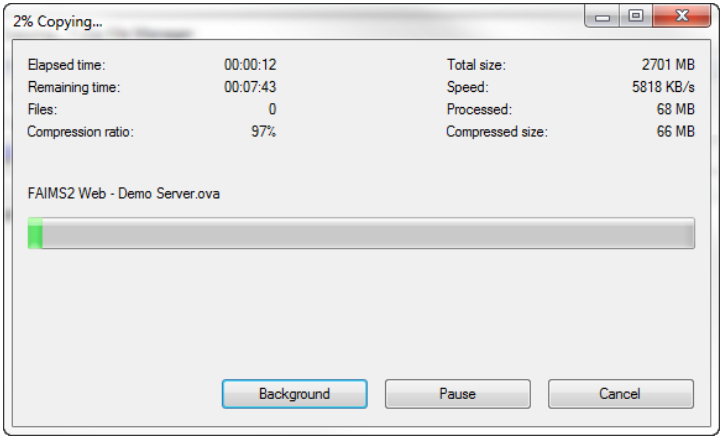

Figure 4.4

To set up the emulated machine, we'll use the preconfigured machine image we downloaded in the last step.

In VirtualBox, select the "File->Import Appliance" option.

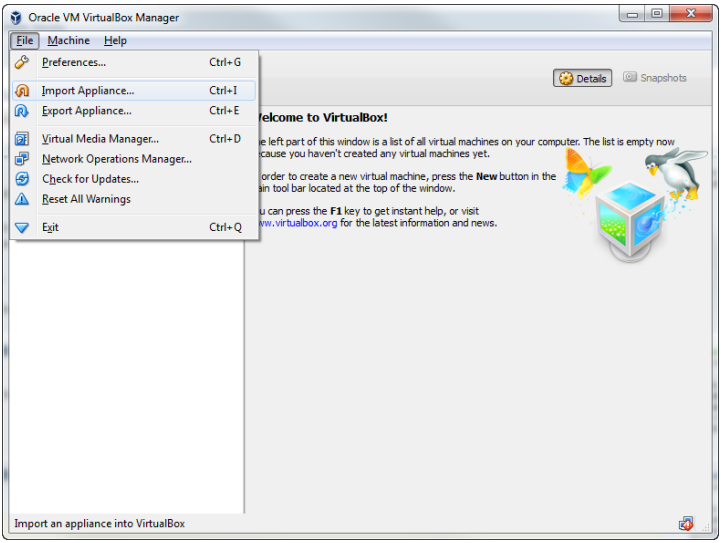

Figure 4.5

Using the file browser that appears, navigate to the directory where you downloaded the FAIMS server image and select to open it.

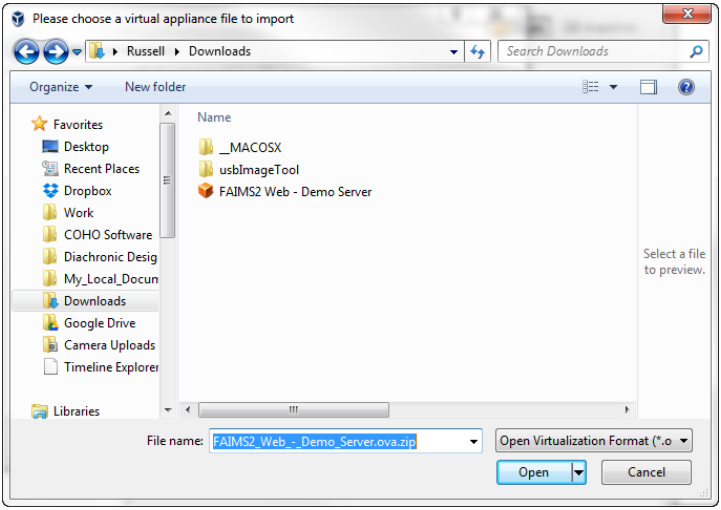

Figure 4.6

The pre-configured settings will appear. Select the “Import” option.

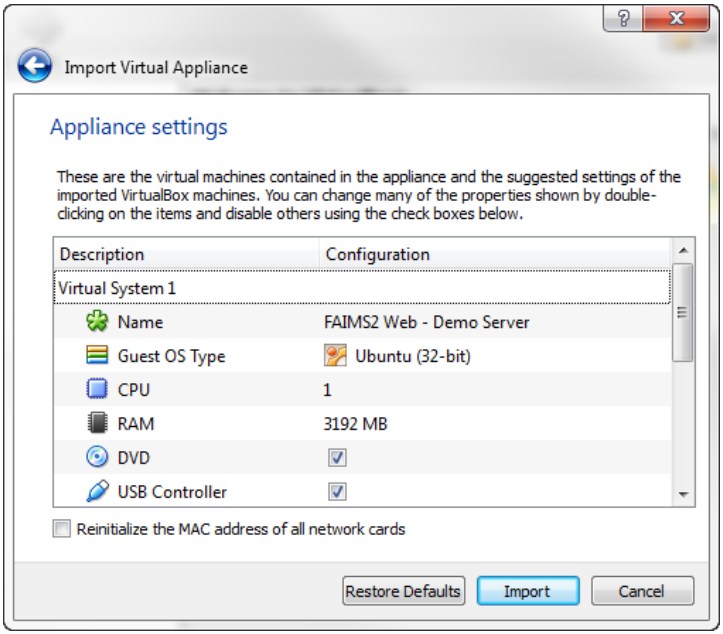

Figure 4.7

VirtualBox will now set to work importing the image and setting everything up. Depending on your system, this may take a few minutes.

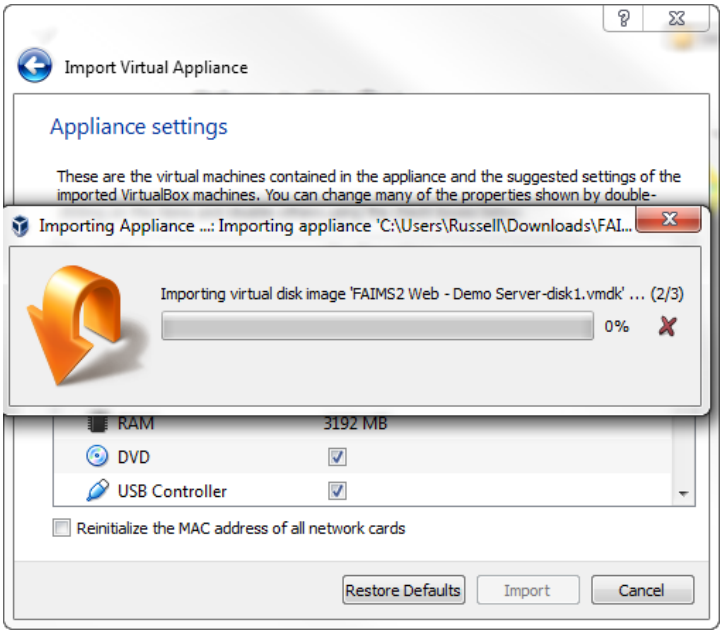

Figure 4.8

Once the setup is complete, "FAIMS2 Web - Demo Server" will appear in the menu on the left in VirtualBox.

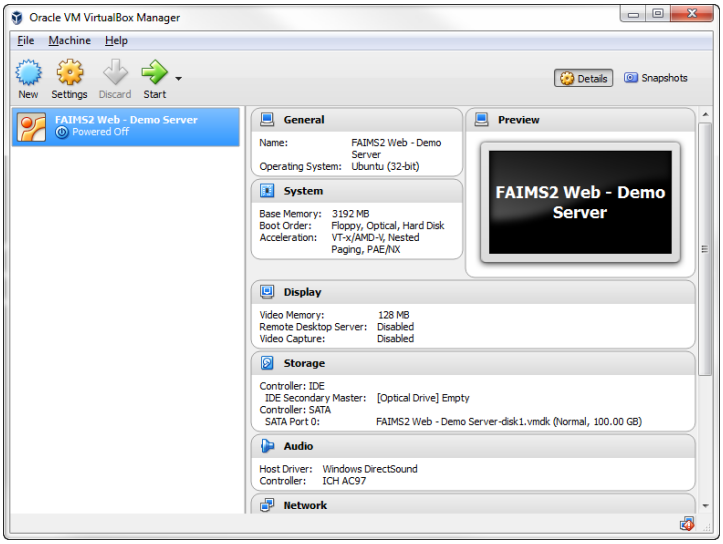

Figure 4.9

Click on the system and then click the "Start" button in VirtualBox to start the server. You *will* encounter the following error.

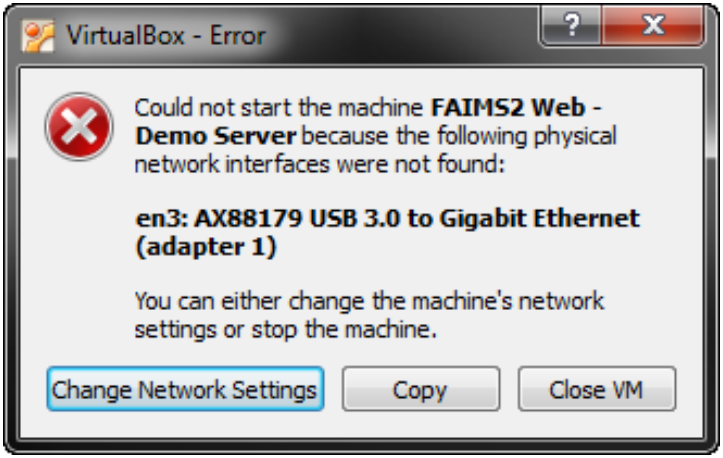

Figure 4.10

DON'T PANIC.

Simply, click the "Close VM" button. You're still on the right track; sometimes, things just have to be a little fussy.

Back on the main screen, click on the "Network" section.

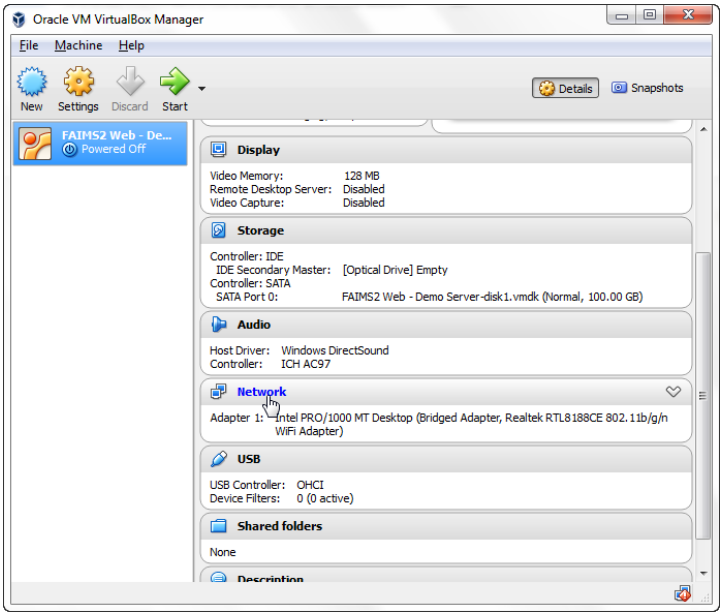

Figure 4.11

In the popup window that appears, check the "Enable Network Adapter" box on the "Adapter 1" tab, and select "Bridged Adapter" from the "Attached to:" menu.

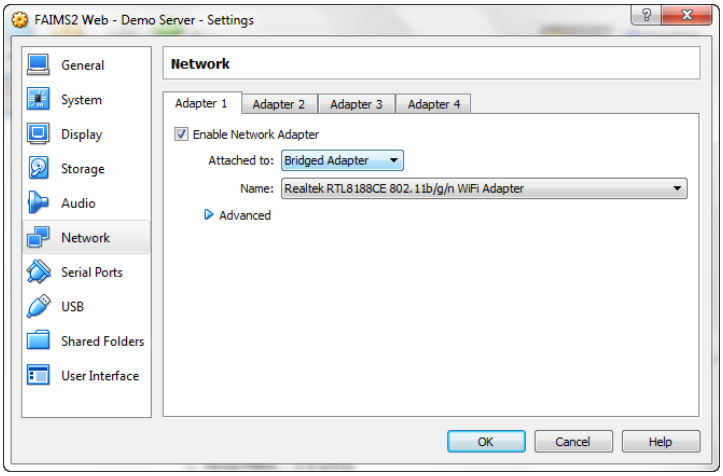

Figure 4.12

You may choose your machine's wifi adapter but, if you run into any issues, try using your hardline connection instead.

Click the "Start" button on the VirtualBox program. VirtualBox will pop up a new window in which your emulated machine will run. Give VirtualBox a few minutes to start your new emulated Ubuntu machine, and a few minutes longer for Ubuntu to start the FAIMS server. When they've finished, you will see the following message about the server having been set up.

Figure 4.13

If you like, you can skip ahead a little and, back on your real machine, you can open the FAIMS server by navigating to 192.168.1.121, or whatever address is listed by your FAIMS status page in your web browser (check your messages for the specific IP address on your machine). **Note:** The default FAIMS account is `faimsadmin@intersect.org.au`, password `Pass.123`.

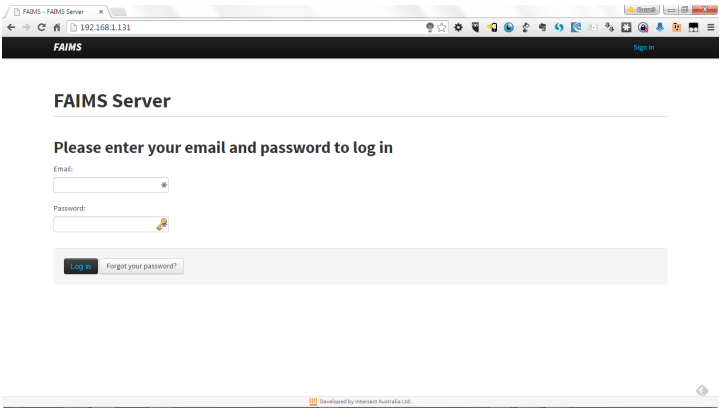

Figure 4.14

If you log out of the Ubuntu installation at any time, just use the password `Pass.123` to log back into the system.

Quiz 4.1 Test your knowledge about the VM

1. If the FAIMS server is supposed to be run on an Ubuntu machine, how are you going to run it on your non-Ubuntu computer?
2. What can you do if you can't find an option or setting we refer to in our instructions?
3. If you've installed VirtualBox before, what do you need to do before setting it up on your computer?

4.2 Finding the right Text Editor for your

The purpose of the FAIMS tools are to remove the need to individually craft all of your module's necessary files. Instead, you'll need to create one file the tools can use as a blueprint for making everything else. In order to make that one file (and to make any fancy edits to the

necessary files to add detail or functionality), you're going to need a simple text editor that's good for coding with.

If you're doing your coding within the virtual machine, we recommend using an editor called gEdit which came with your Ubuntu installation. It's simple, functional, and won't clog up your instructions with unnecessary formatting like a standard Word Processor[7].

If you decide to write the files outside your virtual machine, we don't recommend using the text editors (such as Notepad) that come with Windows or OSX. While they may not complicate your code like a word processor would, they also don't have little features like autocomplete (a feature that means you don't need to fully type out words you use often) and syntax highlighting (a feature that helps you keep track of how your code is structured with helpful visual cues). With a little searching around, you can find plenty of excellent free or commercial text editors.

We particularly recommend:

- *Notepad++* (Windows)
- *Atom* (Cross Platform)
- *TextWrangler* (OSX)

For each of these, download their installers and follow their installation wizards. If you're using a Linux device that isn't the virtual machine, you can install gEdit, Vim, Emacs, or another text editor using apt-get.

4.3 Access to local files via VirtualBox

[7] Note from your friends at FAIMS: One of the main reasons to use the text editors above is so that you can avoid "Smart Quotes," an automatic feature of many word processor programs that will insert differently styled or extra quotes that will cause the FAIMS server to reject your module. If you run into any issues with your module, ensure that "Smart Quotes" are disabled in your text editor.

4.4 Installing the Android SDK

Now we'll need to install the Android Software Development Kit (ADK). It's available at `https://developer.android.com/sdk/installing/index.html` Ignore the Android Studio link; we just need the SDK Tools.

Because you're going to use the tools for debugging an Android device plugged into your actual computer, you're going to want to download and install SDK Tools to your regular computer system, NOT your Ubuntu virtual machine.

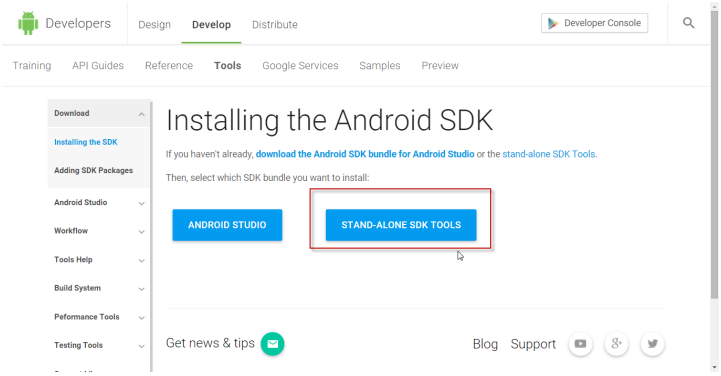

Figure 4.15 Click Stand-Alone SDK Tools

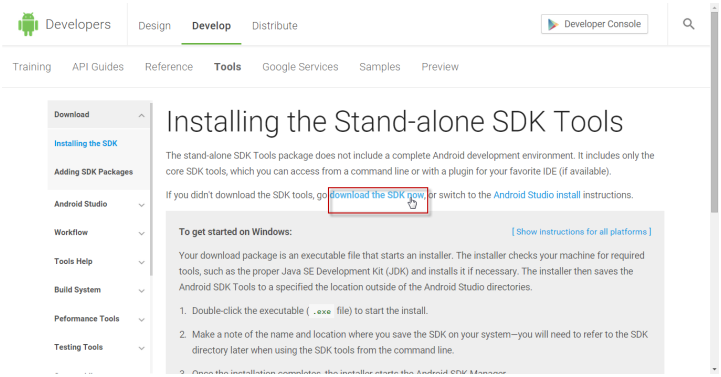

Figure 4.16 Choose to download the SDK Now

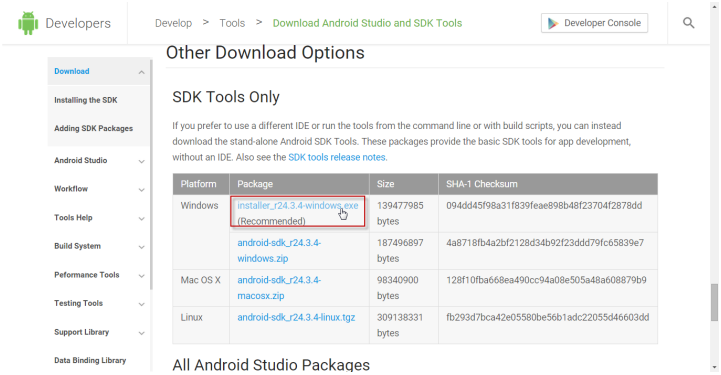

Figure 4.17 Make sure to get the SDK tools only. We won't need to compile a new Android application here.

Run the downloaded installer and follow its prompts to finish setting up the SDK tools.

4.4.1 Installing the FAIMS Debug APK on your Android Device

The final step to setting up your development environment is to install the FAIMS app on your Android tablet or phone. Instead of installing the app through the Google Play Store, navigate to `https://www.fedarch.org/apk/faims-debug-latest.apk` and download the special debug version.

Next, you'll need to place the FAIMS APK file on your phone, either by moving it over via USB connection or by placing the APK file in a cloud service like Dropbox and downloading it through the mobile app.

If you load the URL `https://www.fedarch.org/apk/faims-debug-latest.apk` on your Android devices, you'll be able to directly download the file.

To install the APK, you may need to change the security settings on your device to allow non-market apps. Go to the "Settings" app, find the Security settings, and check "Unknown sources" (device pictured is a Samsung Galaxy s5. Your settings menu may look different).

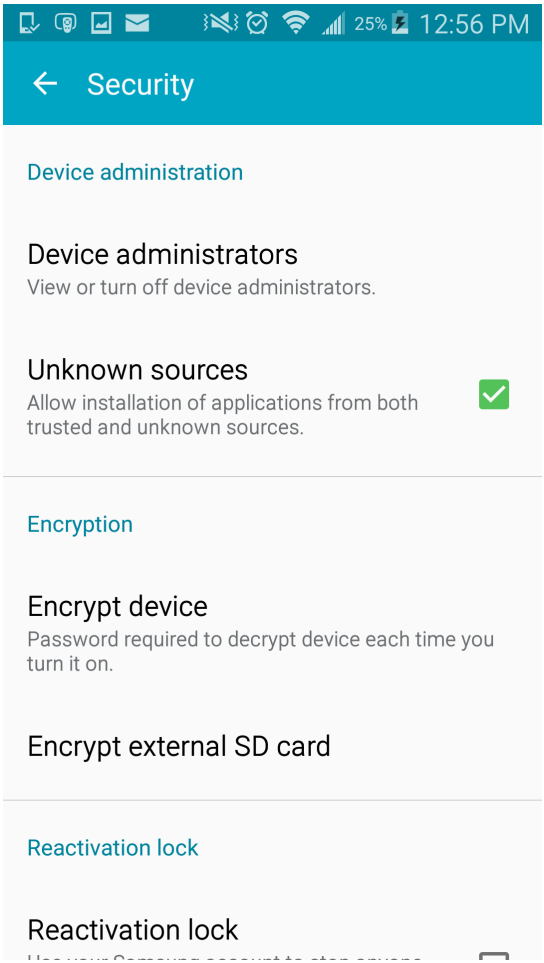

Figure 4.18

4.4.2 Making your Android Device Communicate with your Computer with USB Debugging

First, you'll need to enable the "Developer Options" on your device. This is another step where the instructions are different depending on what kind of device you're using.

On Android versions 6+ to enable the “Developer Options”:

1. Open the App drawer.
2. Launch Settings menu.
3. Find the open the ’About Device’ menu.
4. Scroll down to ’Build Number’.
5. Next, tap on the ’Build Number’ section seven times. (Yes, really.)
6. After the seventh tap you will be told that you are now a developer. (Again, this is really how it works.)
7. Go back to Settings menu and the Developer Options menu will now be displayed.

You will need to install (for some samsung models) specific debug drivers:

Note that if you want for USB Debugging to work when your Samsung device is connected with your computer, then you will first need to make sure that the Samsung USB Drivers are installed on your PC.

http://www.android.gs/download-samsung-usb-drivers-for-android/

For many non-Samsung devices, you can use Google’s ABD/USB driver instead: https://developer.android.com/studio/run/oem-usb.html#InstallingDriver

In order to enable the USB Debugging you will need to open Developer Options, scroll down and tick the box that says “USB Debugging”.

That’s it! Your FAIMS development environment should now be set up properly.

Quiz 4.2 Test your knowledge of all of FAIMS needed support tools.

1. What are two things you can do if you don't find an option or setting where we tell you it will be?
2. If the "Error: Network Device Not Found" error appears when you start VirtualBox, how do you fix it?
3. True or False: Microsoft Word is a good program to code your modules in.
4. Should you install your Android developer kit within the virtual machine? Why or why not?
5. Which of the following are true?
   - VirtualBox allows you to run a different operating system on your computer
   - VirtualBox connects to a computer at the FAIMS offices
   - You must install XSLT Proc and saxon-xslt before running the XML generator tool
   - FAIMS can use both Android and Apple devices
6. Exercise: Open up a command window. In this command window, write `adb --version` then enter. Some information about the Android Development Bridge (ADB) program should appear. Did you encounter any errors? If not, congratulations, ADB/Android SDK is successfully installed.

5 How to Code Modules

For this tutorial, we'll work together and create a simple FAIMS module. For illustrative purposes, we're going to recreate a simple module available from the FAIMS demo server–the "Oral History" module.

We're going to start by showing you how to make a basic but fully functional version of the module that can be made only by constructing a relevant module.xml. Then, after you've run that through the FAIMS-tools and produced a fully functioning but simple version, we'll teach you some ways to add complexity and functionality by modifying the necessary files directly.

5.1 Introduction to Module.xml

Earlier sections introduced the basic necessary files of a module. You should already have some idea what these were:

1. a Data Definition Schema (data_schema.xml).
2. a User Interface Schema (ui_schema.xml)
3. a User Interface Logic file (ui_logic.bsh)
4. a Translation (Arch16n) file (faims.properties)
5. A server-side validation file (validation.xml)
6. CSS Files for styling the modules (ui_??)

You've already learned that you don't have to design and code all of these files from scratch. Instead, you'll create one file that serves as a set of instructions for the FAIMS Tools to create the necessary files for you. That one file is called module.xml.

Quiz 5.1 Test your knowledge: a gentle descent into module.xml

1. What are some programs you can use when modifying module.xml? Which programs should you avoid? If you don't remember, review the previous section, "Setting Up Your Development Environment."