

User to Developer Guide: How to Build and Deploy Modules for the FAIMS Mobile app

Version 1.20171011

Dr Brian Ballsun-Stanton

Russell Alleen-Willems

Adam DeCamp

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

Code for this project and the latest version of this document can be found at <http://github.com/FAIMS/User-ToDev>.

This document is licensed under CC-BY-SA International 4.0, Copyright 2017.

1	Forward	3
2	What is FAIMS?	5
2.1	What does the “module” do?	5
2.2	Why should you use a FAIMS database?	6
2.3	What are the benefits of using the FAIMS database over other databases?	9
2.4	What are the benefits of using the FAIMS database over a spreadsheet?	9
2.4.1	Some framing questions for FAIMS Mobile	12
2.5	Exporters: how to share data with others.	13
3	Making Your Own Modules	17
3.1	The Parts of a Module	17
3.1.1	Data Schema	18
3.1.2	UI Schema	19
3.1.3	Validation Schema	19
3.1.4	UI Logic	20
3.1.5	Arch16n	21
3.1.6	CSS	21
3.1.7	Picture Gallery Images	21
3.2	Tour from the Data Schema	23
3.2.1	How to format data schema for the FAIMS system	23

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 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>>>

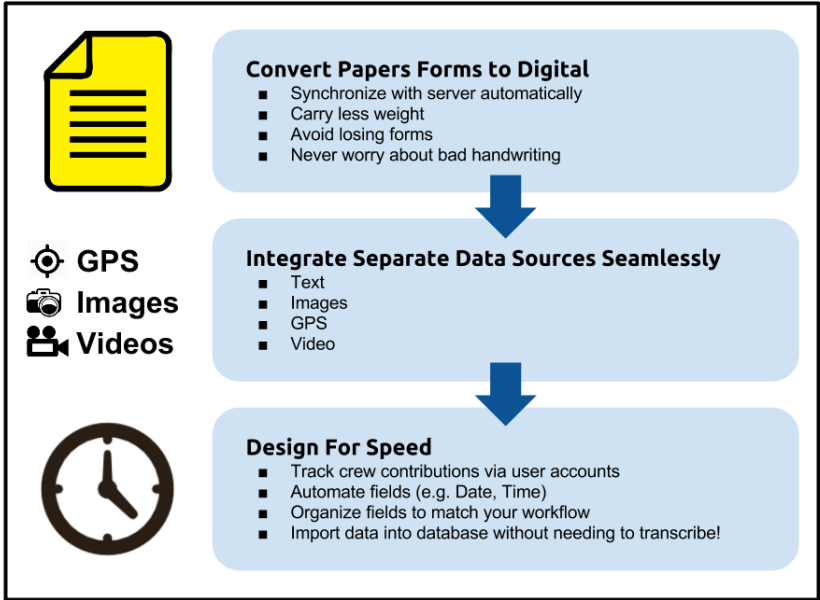


Figure 2.1 A prototypical workflow for a FAIMS Mobile module.

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.

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 few chapters, 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 create the module you want.
- 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 made from parts, called “necessary files.”

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 (as you’ll learn in the next section). Each necessary file serves a definite and distinct function in the final operation of the module.

You can simply use a module without ever learning what they 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 proofread, 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 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 cattles.

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

3.2.1 How to format data schema for the FAIMS system

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.

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.

Archaeological Elements 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. 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. You don't really need to understand this yet, but it won't hurt to familiarize yourself with its structure.

```

<ArchaeologicalElement name="small">
  <description>
    A small entity
  </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</term>
      <term>Location B</term>
      <term>Location C</term>
      <term>Location D</term>
    </lookup>
  </property>
</ArchaeologicalElement>

```

Code 3.1 A demonstration archaeological element or “Archent”

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."

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:

```
<RelationshipElement name="AboveBelow" type="hierarchy">
```

```
<description>
```

Indicates that one element is above or below another element.

```
</description>
```

```
<parent>
```

Above

</parent>

<child>

Below

</child>

<property type="string" name="relationship" isIdentifier="true">

</property>

<property type="string" name="name">

</property>

<property type="dropdown" name="location">

<lookup>

<term>Location A</term>

<term>Location B</term>

<term>Location C</term>

<term>Location D</term>

</lookup>

</property>

</RelationshipElement>