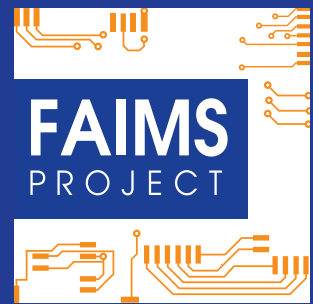


Software Documentation Collection



30 June 2014

Brian Ballsun-Stanton, Adela Sobotkova



www.fedarch.org

FAIMS Deliverable #19

prepared by: Brian Ballsun-Stanton, Adela Sobotkova

This collection combines the developer- and user-level documentation for the FAIMS Platform. It has been collected from various sources into a single document for Steering Committee approval. Much of this documentation can be found on wiki.fedarch.org. This document functions as a pointer to the resources available on our wiki.



Australian Government

Australian Research Council



UNSW
AUSTRALIA



THE UNIVERSITY
OF QUEENSLAND
AUSTRALIA



THE UNIVERSITY OF
SYDNEY



LA TROBE
UNIVERSITY



Flinders
UNIVERSITY



Southern Cross
University



OCHRE Data Service
of the Oriental Institute



Contents

| | | |
|-----|--------------------------|-----|
| 1 | Mobile Application | 3 |
| 1.1 | Cookbook | 3 |
| 1.2 | Program Logic | 60 |
| 1.3 | Known Issues | 82 |
| 2 | Heurist | 84 |
| 3 | FAIMS Repository | 105 |
| 3.1 | Developer Instructions | 105 |
| 3.2 | Mobile Import | 106 |
| 3.3 | Backups | 107 |
| 3.4 | Command Line Import Tool | 110 |
| 3.5 | Storage | 117 |
| 3.6 | Authentication | 122 |
| 3.7 | CAA 2013 Talk | 127 |
| 3.8 | Internal VerSI talk | 130 |

1 Mobile Application

All materials listed here can be found at the [***FAIMS Mobile Application Wiki***](#).

1.1 Cookbook

FAIMS Mobile Application Cookbook - Data and UI schema documentation.

The FAIMS Cookbook is a guide to the internal structure of the FAIMS Mobile Application recording module. It is aimed at both developers and enthusiastic users who want to become conversant with the FAIMS Mobile Application. The Cookbook lists the lines of code in a set of design patterns, which can guide manual creation of modules from scratch. The Cookbook aims to demonstrate all aspects of module data structure and user interface, and to provide instructive examples that can be copied from. It is the formal reference for FAIMS Mobile Application data schemas and ui schemas. For information on the logic file, please refer to the next section.

FAIMS Data, UI and Logic Cook-Book

Use this guide to help create your own projects by following the examples below.

- 1 Use this guide to help create your own projects by following the examples below.
- 2 FAIMS Data, UI and Logic Cook-Book
- 3 Module Creation
 - 3.1 Simple Module
 - 3.2 How it works
 - 3.3 Data Schema Construction
 - 3.4 Relationship Element
 - 3.5 Archaeological Element
 - 3.6 Constructing a UI
 - 3.7 Creating a Text View
 - 3.8 Creating a Number Text View
 - 3.9 Creating a Dropdown
 - 3.10 Creating a Checkbox Group
 - 3.11 Creating a RadioButton Group
 - 3.12 Creating a List View
 - 3.13 Creating a Date Picker
 - 3.14 Creating a Time Picker
 - 3.15 Creating a Map View
 - 3.16 Creating a Picture Gallery
 - 3.17 Creating a Camera Gallery
 - 3.18 Creating a Video Gallery
 - 3.19 Creating a Files List
 - 3.20 Creating a Button
- 4 Customising the UI
 - 4.1 Hiding labels
 - 4.2 Tab Scrolling
 - 4.3 Hiding Tabs
 - 4.4 Making Text Views readonly
 - 4.5 Binding Tabgroup to Arch Entity
 - 4.6 Binding Tabgroup to Relationship
 - 4.7 Binding Views to Entity/Relationship Attributes
 - 4.8 Disabling Certainty Buttons on views
 - 4.9 Disabling Annotation Buttons on views
 - 4.10 Syncing files
 - 4.10.1 Styling
- 5 Using Logic
 - 5.1 Automated Saving/Loading Arch Entity
 - 5.2 How it works
 - 5.3 Automated Saving/Loading Relationships
 - 5.4 Fetching Entities and Relationships
 - 5.5 Saving Archaeological Entities and Relationships
 - 5.6 Deleting Archaeological Entities and Relationships
 - 5.7 Deleting Attributes
 - 5.8 Relating Entities
 - 5.9 Getting Field Values, Annotations and Certainty
 - 5.10 Setting Field Values, Annotations and Certainty
 - 5.11 Event Handling
 - 5.12 Alert, Toast, Busy and Warning
 - 5.13 Drop Downs, Radio Button Groups, CheckBox Groups, Lists, Picture Gallery
 - 5.14 Showing Tabs & Tab Groups
 - 5.15 Leaving Tabs and Tab Groups
 - 5.16 Date & Time
 - 5.17 Module Metadata
- 6 Maps and GIS
 - 6.1 Rendering map
 - 6.2 Add base map
 - 6.3 Add raster map
 - 6.4 Add shape vector layer
 - 6.5 Add spatial vector layer
 - 6.6 Add database vector layer
 - 6.7 Add canvas layer
 - 6.8 Map Controls
 - 6.9 Center map using GPS
 - 6.10 Locking / Unlocking the map

- 6.11 Adding map event listeners
 - 6.12 Styling Vectors
 - 6.13 Drawing points
 - 6.14 Drawing lines
 - 6.15 Drawing polygons
 - 6.16 Moving vector geometry
 - 6.17 Clearing Geometry
 - 6.18 Saving GIS to Entities / Relationships
 - 6.19 Loading GIS from Entities / Relationships
 - 6.20 Add Database Layer Queries
 - 6.21 Add TrackLog layer Queries
 - 6.22 Adding Selection Tool Queries
 - 6.23 setLayerVisible
 - 6.24 Convert projection of points
 - 6.25 Enable / Disable tools view
 - 6.26 Bind events to tools
 - 6.27 Refresh map
- 7 GPS
 - 7.1 Get GPS position
 - 7.2 Get GPS accuracy
 - 7.3 Get GPS heading
 - 7.4 Set GPS interval
 - 7.5 Setup Track Logs
- 8 Syncing
 - 8.1 Pull database from server
 - 8.2 Push Database to server
 - 8.3 Database syncing
 - 8.4 Stop syncing
 - 8.5 Sync Event
 - 8.6 File syncing
- 9 File Attachments
 - 9.1 Selecting files
 - 9.2 Taking Photos
 - 9.3 Recording Video
 - 9.4 Recording Audio
 - 9.5 Saving / Loading Files, Photos, Videos and Audio
 - 9.6 View Attached Files
- 10 Misc
 - 10.1 UI Logic persistence
 - 10.2 Set User
 - 10.3 Execute
- 11 Validation
 - 11.1 How it works
 - 11.2 Evaluator
 - 11.3 Blank Checker
 - 11.4 Type Checker
 - 11.5 Query Checker
- 12 Arch16n translations

FAIMS Data, UI and Logic Cook-Book

Use this guide to help create your own modules by following the examples below.

Module Creation

When creating module there are some files that are needed to create the module such as:

1. data schema
2. ui schema
3. ui logic

In addition there are some optional files such as:

1. properties file
2. validation schema

Below is the example of a creation for simple module by providing required files

Simple Module

This is an example of a simple module that renders a single text view

data_schema.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="data_schema.xsl"?>
<dataSchema name="SimpleModule" preparer="Your Name">
</dataSchema>
```

ui_schema.xml

```
<h:html xmlns="http://www.w3.org/2002/xforms"
        xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:ev="http://www.w3.org/2001/xml-events"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:jr="http://openrosa.org/javarosa">
  <h:head>
    <h:title>Simple Example</h:title>

    <model>
      <instance>
        <faims id="simple_example">
          <tabgroup1>
            <tab1>
              <text></text>
            </tab1>
          </tabgroup1>
        </faims>
      </instance>
      <bind nodeset="/faims/tabgroup1/tab1/text" type="string"/>
    </model>
  </h:head>

  <h:body>
    <group ref="tabgroup1">
      <label></label>
      <group ref="tab1">
        <label>{tab_name}</label>
        <input ref="text">
          <label>Text:</label>
        </input>
      </group>
    </group>
  </h:body>
</h:html>
```

ui_logic.bsh

```
setFieldValue("tabgroup1/tab1/text", "Hello World!");
```

How it works

- Use the module files to create a module on the server. Then download the module onto the android app.
- The data_schema.xml specifies what archaeological entities and relationships to store. This data schema is empty as we are not saving any data at the moment.

- The ui_schema.xml specifies how the ui should render. More information on how to construct these will be provided in later examples. Currently this specifies a single tabgroup, a single tab and a single text view.
- The ui_logic.bsh specifies how ui elements interact on screen and with the database. Current this example simply sets the value of the text view to "Hello World!".

Data Schema Construction

This section will explain about the structure of the data schema. An example of data schema:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="sampleDataXML.xsl"?>
<dataSchema name="SyncExample" preparer="Nobody">

  <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">
      <bundle>DOI</bundle>
    </property>
    <property type="string" name="name">
      <bundle>DOI</bundle>
    </property>
    <property type="dropdown" name="location">
      <bundle>DOI</bundle>
      <lookup>
        <term>Location A</term>
        <term>Location B</term>
        <term>Location C</term>
        <term>Location D</term>
      </lookup>
    </property>
  </RelationshipElement>

  <ArchaeologicalElement name="small">
    <description>
      An small entity
    </description>
    <property type="string" name="entity" isIdentifier="true">
      <bundle>DOI</bundle>
    </property>
    <property type="string" name="name">
      <bundle>DOI</bundle>
    </property>
    <property type="integer" name="value">
      <bundle>DOI</bundle>
    </property>
    <property type="file" name="filename">
      <bundle>DOI</bundle>
    </property>
    <property type="file" name="picture">
      <bundle>DOI</bundle>
    </property>
  </ArchaeologicalElement>
</dataSchema>
```

```
<property type="file" name="video">
  <bundle>DOI</bundle>
</property>
<property type="file" name="audio">
  <bundle>DOI</bundle>
</property>
<property type="timestamp" name="timestamp">
  <bundle>DOI</bundle>
</property>
<property type="dropdown" name="location">
  <bundle>DOI</bundle>
  <lookup>
    <term>Location A</term>
    <term>Location B</term>
    <term>Location C</term>
    <term>Location D</term>
  </lookup>
```



```
</property>
</ArchaeologicalElement>
</dataSchema>
```

There are 2 types of elements for the data schema namely RelationshipElement and ArchaeologicalElement. Relationship element defines the schema of a relationship while archaeological element defines the schema of a archaeological entity.

Relationship Element

A relationship element has both name and type attributes. Type states the nature of the relationship of which there are three which are hierarchy, bidirectional and container. There are also child elements contained in a relationship element:

1. description: describes the relationship element
2. parent: defines the verb for the parent entity
3. child: defines the verb for the child entity
4. property: a data attribute of a relationship; properties have both name and type attributes:
 - bundle: ?
 - lookup: list vocabulary terms for the property

Archaeological Element

A archaeological element a type attribute. There are also child elements contained in a archaeological element:

1. description: the description of the archaeological element
2. property: a data attribute of the archaeological entity; properties have both name and type attributes:
 - bundle: ?
 - lookup: list vocabulary terms for the property

Constructing a UI

This example will teach you how to construct a ui and bind it with logic.

The faims android apps dynamic ui is comprised of tabgroups, tabs and views. The full list of supported views are:

- Text View
- DropDown
- Checkbox Group
- Radiobox Group
- List View
- Date Picker
- Time Picker
- Map View
- Picture Gallery
- Camera Gallery
- Video Gallery
- Files List
- Button

Creating a Text View

The ui_schema.xml is used to define what to render on screen. There are two parts to the ui schema. The first part defines the overall structure of the ui and second part defines the elements on screen to render.

Look at this example ui_schema.xml where each part is labeled as comments.

```

<h:html xmlns="http://www.w3.org/2002/xforms"
  xmlns:h="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jr="http://openrosa.org/javarosa">
  <h:head>
    <h:title>Simple Example</h:title>

    <model>
      <instance>
        <faims id="simple_example">
          <!-- PART 1: Define ui structure -->
          <tabgroup1>
            <tab1>
              <text></text>
            </tab1>
          </tabgroup1>
        </faims>
      </instance>
      <bind nodeset="/faims/tabgroup1/tab1/text" type="string"/>
    </model>
  </h:head>

  <h:body>
    <!-- PART 2: Define ui elements -->
    <group ref="tabgroup1">
      <label></label>
      <group ref="tab1">
        <label>Tab 1</label>
        <input ref="text">
          <label>Text:</label>
        </input>
      </group>
    </group>
  </h:body>
</h:html>

```

In this example part 1 defines a tabgroup called "tabgroup1", a tab inside the tabgroup called "tab1" and a text element inside the tab called "text".



The names for the tabgroup, tab and element can be called anything that is valid xml.

In the second part it defines how the structure is to be rendered.

```

<group ref="tabgroup1">
  <label></label>

```

These lines in the xml document define a group element "tabgroup1" (referenced using the ref attribute) which will be used to rendered it as a tabgroup. The label is currently not used but is needed to ensure a valid ui schema.

```

<group ref="tab1">
  <label>Tab 1</label>

```

These lines in the xml document nested inside the parent group define a group element "tab1" which will be used to rendered it as a tab. The label is used to label the tab in the ui.



The faims android app ui must use tabgroups and tabs in this way or else the app will not recognise the xml as valid. Also be aware that both group elements define label elements which is a requirement

```
<input ref="text">
  <label>Text:</label>
</input>
```

Finally these lines of code define an input element for text. The label is used to label the text element.

Creating a Number Text View

Following from the previous example the example below defines an additional text view "number" that is of decimal format.

```
...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <text></text>
          <number></number>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
  <bind nodeset="/faims/tabgroup1/tab1/text" type="string"/>
  <bind nodeset="/faims/tabgroup1/tab1/number" type="decimal"/>
...

...
  <group ref="tab1">
    <label>Tab 1</label>
    <input ref="text">
      <label>Text:</label>
    </input>
    <input ref="number">
      <label>Number:</label>
    </input>
  </group>
...
```

```
<bind nodeset="/faims/tabgroup1/tab1/number" type="decimal"/>
```

This line in the ui schema lets the renderer know to render this text view as a number only text view. Notice that the nodeset value follows the ordering of the xml elements. This is used to ref number text view uniquely.

Creating a Dropdown

This example creates a drop down.

```

...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <text></text>
          <number></number>
          <itemList></itemList>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
  <bind nodeset="/faims/tabgroup1/tab1/text" type="string"/>
  <bind nodeset="/faims/tabgroup1/tab1/number" type="decimal"/>
...

...
<group ref="tab1">
  <label>Tab 1</label>
  <input ref="text">
    <label>Text:</label>
  </input>
  <input ref="number">
    <label>Number:</label>
  </input>
  <select1 ref="itemList">
    <label>Item List:</label>
    <item>
      <label>Item A</label>
      <value>0</value>
    </item>
    <item>
      <label>Item B</label>
      <value>1</value>
    </item>
    <item>
      <label>Item C</label>
      <value>2</value>
    </item>
    <item>
      <label>Item D</label>
      <value>3</value>
    </item>
  </select1>
</group>
...

```

```

<item>
  <label>Item A</label>
  <value>0</value>
</item>

```

These lines in the ui schema define a single item of the dropdown. The label is the text that shows up in the drop down and value is the value using logic calls (more on that later).



Drop downs or any list views must include at least 1 item in ui schema even if later in ui logic you remove them.

Creating a Checkbox Group

This example creates a checkbox group.

```
...  
    <select ref="itemList">  
      <label>Item List:</label>  
      <item>  
        <label>Item A</label>  
        <value>0</value>  
      </item>  
      <item>  
        <label>Item B</label>  
        <value>1</value>  
      </item>  
      <item>  
        <label>Item C</label>  
        <value>2</value>  
      </item>  
      <item>  
        <label>Item D</label>  
        <value>3</value>  
      </item>  
    </select>  
...
```



Notice that select is used instead of select1.

Creating a RadioButton Group

This example creates a radio button group.

```

...
<select1 ref="itemList" appearance="full">
  <label>Item List:</label>
  <item>
    <label>Item A</label>
    <value>0</value>
  </item>
  <item>
    <label>Item B</label>
    <value>1</value>
  </item>
  <item>
    <label>Item C</label>
    <value>2</value>
  </item>
  <item>
    <label>Item D</label>
    <value>3</value>
  </item>
</select1>
...

```



Notice that the select1 has appearance set to full.


Creating a List View

This example creates a list view.

```

...
<group ref="tab1" faims_scrollable="false">
  <label>Tab 1</label>
...
  <select1 ref="itemList" appearance="compact">
    <label>Item List:</label>
    <item>
      <label>Item A</label>
      <value>0</value>
    </item>
    <item>
      <label>Item B</label>
      <value>1</value>
    </item>
    <item>
      <label>Item C</label>
      <value>2</value>
    </item>
    <item>
      <label>Item D</label>
      <value>3</value>
    </item>
  </select1>
...

```

 Notice that the select1 has appearance set to compact.

```
<group ref="tab1" faims_scrollable="false">
  <label>Tab 1</label>
```


Since list are scrollable views they cannot be placed into normal tabs as tabs themselves are scrollable. Therefore you set tabs to not scroll by using the faims_scrollable attribute to make a tab not scroll.

Creating a Date Picker

This example creates a date picker.

```
...
  <faims id="simple_example">
    <!-- PART 1: Define ui structure -->
    <tabgroup1>
      <tab1>
        <date>
      </tab1>
    </tabgroup1>
  </faims>
</instance>
<bind nodeset="/faims/tabgroup1/tab1/date" type="date"/>
...

...
  <group ref="tab1">
    <label>Tab 1</label>
    <input ref="date">
      <label>Date:</label>
    </input>
  </group>
...
```

 Notice the binding of date to type date.

Creating a Time Picker


This example creates a time picker.

```

...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <time>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
  <bind nodeset="/faims/tabgroup1/tab1/time" type="time"/>
...

...
<group ref="tab1">
  <label>Tab 1</label>
  <input ref="time">
    <label>Time:</label>
  </input>
</group>
...

```

 Notice the binding of time to type time.

Creating a Map View


This example creates a map view.

```

...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <map>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
...

...
<group ref="tab1">
  <label>Tab 1</label>
  <input ref="map" faims_map="true">
    <label>Map:</label>
  </input>
</group>
...

```

 Notice set faims_map to true to make it a map view.

Creating a Picture Gallery

This example creates a picture gallery.

```
...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <pictures>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
...

...
  <group ref="tab1">
    <label>Tab 1</label>
    <select1 ref="picture" type="image">
      <label>Picture:</label>
      <item>
        <label>dummy</label>
        <value>dummy</value>
      </item>
    </select1>
  </group>
...

```

Creating a Camera Gallery

This example creates a camera gallery slider.

```
...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <gallery>
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
...

...
  <group ref="tab1">
    <label>Tab 1</label>
    <select ref="gallery" type="camera">
      <label>Gallery:</label>
      <item>
        <label>dummy</label>
        <value>dummy</value>
      </item>
    </select>
  </group>
...

```



Notice that the camera and is a select not select1 as the picture gallery

Creating a Video Gallery

This example creates a video gallery slider.

```
...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <gallery>
            </tab1>
          </tabgroup1>
        </faims>
      </instance>
    ...

    ...
    <group ref="tab1">
      <label>Tab 1</label>
      <select ref="gallery" type="video">
        <label>Gallery:</label>
        <item>
          <label>dummy</label>
          <value>dummy</value>
        </item>
      </select>
    </group>
    ...
```



Notice that the video and is a select not select1 as the picture gallery

Creating a Files List

This example creates a files list.

```

...
    <select ref="files" type="file">
      <label>Files:</label>
      <item>
        <label>Item A</label>
        <value>0</value>
      </item>
      <item>
        <label>Item B</label>
        <value>1</value>
      </item>
      <item>
        <label>Item C</label>
        <value>2</value>
      </item>
      <item>
        <label>Item D</label>
        <value>3</value>
      </item>
    </select>
...

```



Notice that select is used instead of select1.

Creating a Button

This example creates a button.

```

...
    <faims id="simple_example">
      <!-- PART 1: Define ui structure -->
      <tabgroup1>
        <tab1>
          <button />
        </tab1>
      </tabgroup1>
    </faims>
  </instance>
...

...
  <group ref="tab1">
    <label>Tab 1</label>
    <trigger ref="button">
      <label>Click Me</label>
    </trigger>
  </group>
...

```

Customising the UI

Now that you know how to construct the UI here are some examples on how to customise the ui so you can facilitate automatic loading of data

from the database, hiding tabs, making readonly elements etc

Hiding labels

Labels can be hidden for views by creating an empty label node. You must include an empty label node for the ui schema to be valid.

```
...  
    <input ref="text">  
        <label></label>
```

Tab Scrolling

Tabs are set to scroll by default but to make them stop scrolling then set the `faims_scrollable` attribute in the tab group element to false. e.g.

```
...  
    <group ref="tab1" faims_scrollable="false">  
        <label>Tab 1</label>
```

Hiding Tabs

To hide tabs when they are first shown you can use the `faims_hidden` attribute to make tabs hidden. By default tabs are visible.

```
<group ref="tab1" faims_hidden="true">
```

Making Text Views readonly

Text views can be made readonly by setting `faims_readonly` attribute to true.

```
...  
<input ref="text" faims_read_only="true">  
    <label>Text:</label>  
</input>  
...
```

Binding Tabgroup to Arch Entity

Binding a TabGroup to an arch entity allows you to do automatic loading and saving of the entity defined within a TabGroup via the logic script. To tell which entity type the TabGroup belongs to you can use the `faims_archent_type` attribute to specify the entity type.

```
<group ref="tabgroup1" faims_archent_type="simple">
```



The entity type must match an entity type defined in the data schema. This will be shown in the saving and loading entities example.

Binding Tabgroup to Relationship

Binding a TabGroup to a relationship allows you to do automatic loading and saving of the relationship defined within a TabGroup via the logic script. To tell which relationship type the TabGroup belongs to you can use the `faims_rel_type` attribute to specify the entity type.

```
<group ref="tabgroup1" faims_rel_type="abovebelow">
```



The relationship type must match an relationship type defined in the data schema. This will be shown in the saving and loading relationships example.

Binding Views to Entity/Relationship Attributes

Once a TabGroup is bound to an entity/relationship type you need to specify how the views map to the attributes of the entity/relationship. You can do this by specifying the `faims_attribute_name` and `faims_attribute_type`.

```
<input ref="text" faims_attribute_name="name" faims_attribute_type="freetext">
```



Here text view maps to the name attribute which will store in the attributes freetext. Attributes for entities have 4 values freetext, vocab, measure and certainty while attributes for relationships are freetext, vocab and certainty. More on this in the saving and loading entities/relationships example.

Disabling Certainty Buttons on views

By default all views in TabGroups that are bound to entities or relationships show an certainty button. This button allows views to set certainty values to them. If you want to disable them you can simply set the `faims_certainty` attribute of the view to false.

```
<input ref="text" faims_certainty="false">
```

Disabling Annotation Buttons on views

By default all non-text views in TabGroups that are bound to entities or relationships show an annotation button. This button allows views to set annotation values to them. If you want to disable them you can simply set the `faims_annotation` attribute of the view to false.

```
<input ref="text" faims_annotation="false">
```

Syncing files

For view that are bound to attributes that a file type then you can specify a `faims_sync` attribute to indicate if those files are to sync to the server only or other apps as well. More on this in the saving and loading entities/relationships example.

```
<select ref="files" type="file" faims_attribute_name="files"
faims_attribute_type="freetext" faims_sync="true">
```

Styling

The styling can be defined and applied from the ui schema. It needs to be defined at the top of the tabgroup definition.

```
...
<faims id="simple_example">
  <style>
    <orientation>
      <orientation></orientation>
```

```

    </orientation>
    <even>
      <layout_weight></layout_weight>
    </even>
  </style>
  <tabgroup1>
    <tab1>
      <container1>
        <child1>
          <name></name>
          <value></value>
        </child1>
        <child2>
          <timestamp></timestamp>
          <location></location>
        </child2>
      </container1>
    </tab1>
  </tabgroup1>
...

<group ref="style">
  <label></label>
  <group ref="orientation">
    <label></label>
    <input ref="orientation">
      <label>horizontal</label>
    </input>
  </group>
  <group ref="even">
    <label></label>
    <input ref="layout_weight">
      <label>1</label>
    </input>
  </group>
</group>
<group ref="tabgroup1" faims_archent_type="small">
  <label></label>
  <group ref="tab1" faims_hidden="false">
    <label>Save Entity</label>
    <group ref="container1" faims_style="orientation">
      <label></label>
      <group ref="child1" faims_style="even">
        <label></label>
        <input ref="name" faims_attribute_name="name"
faims_attribute_type="freetext">
          <label>Name:</label>
        </input>
        <input ref="value" faims_attribute_name="value"
faims_attribute_type="measure">
          <label>Value:</label>
        </input>
      </group>
      <group ref="child2" faims_style="even">
        <label></label>
        <input ref="timestamp" faims_attribute_name="timestamp"
faims_attribute_type="freetext" faims_read_only="true" faims_certainty="false">
          <label>Timestamp:</label>
        </input>
        <select ref="location" faims_attribute_name="location"

```

```
faims_attribute_type="vocab">
  <label>Location:</label>
  <item>
    <label>dummy</label>
    <value>dummy</value>
  </item>
```

```

        </select>
    </group>
</group>

```

The example of styling shows that we can define a style for making a container with certain orientation and certain layout weight to divide it even. The styling should be applied to the using the attribute `faims_style` to the group tag.

Using Logic

This section will provide examples on how to the logic file to add interaction between ui logic and data storage.

Automated Saving/Loading Arch Entity

Define a archaeological entity in the data schema.

```

<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="data_schema.xsl"?>
<dataSchema name="SimpleModule" preparer="Your Name">
  <ArchaeologicalElement name="Simple">
    <description>
      An simple entity
    </description>
    <property type="string" name="name" isIdentifier="true">
      <bundle>DOI</bundle>
    </property>
    <property type="real" name="value" isIdentifier="true">
      <bundle>DOI</bundle>
    </property>
  </ArchaeologicalElement>
</dataSchema>

```

This data schema defines a single archaeological entity called "Simple" with two properties name and value.

Now we define a ui schema to save this entity.

```

<h:html xmlns="http://www.w3.org/2002/xforms"
  xmlns:h="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jr="http://openrosa.org/javarosa">
  <h:head>
    <h:title>Simple Example</h:title>

    <model>
      <instance>
        <faims id="simple_example">
          <tabgroup1>
            <tab1>
              <name></name>
              <value></value>
              <save></save>
              <clear></clear>
            </tab1>
            <tab2>
              <entity></entity>
            </tab2>
          </tabgroup1>
        </faims>
      </instance>
    </model>
  </h:head>

```



```

        <load></load>
      </tab2>
    </tabgroup1>
  </faims>
</instance>
<bind nodeset="/faims/tabgroup1/tab1/name" type="string"/>
<bind nodeset="/faims/tabgroup1/tab1/value" type="decimal"/>
</model>
</h:head>

<h:body>
  <group ref="tabgroup1" faims_archent_type="Simple">
    <label></label>
    <group ref="tab1">
      <label>Tab 1</label>
      <input ref="name" faims_attribute_name="name"
faims_attribute_type="freetext">
        <label>Name:</label>
      </input>
      <input ref="value" faims_attribute_name="value"
faims_attribute_type="measure">
        <label>Value:</label>
      </input>
      <trigger ref="save">
        <label>Save</label>
      </trigger>
      <trigger ref="clear">
        <label>Clear</label>
      </trigger>
    </group>
    <group ref="tab2">
      <label>Tab 2</label>
      <select1 ref="entity">
        <label>Entity:</label>
        <item>
          <label>dummy</label>
          <value>dummy</value>
        </item>
      </select1>
      <trigger ref="load">
        <label>Load</label>
      </trigger>
    </group>
  </group>
</h:body>
</group>

```

```
</h:body>
</h:html>
```

Now let use the logic to save and load an arch entity to and from the database.

```
// add click events for buttons
onEvent("tabgroup1/tab1/save", "click", "saveEntity()");
onEvent("tabgroup1/tab1/clear", "click", "clearEntity()");
onEvent("tabgroup1/tab2/load", "click", "loadEntity()");

update() {
    Object entities = fetchAll("select uuid, uuid from archentity where uuid ||
aenttimestamp in ( select uuid || max(aenttimestamp) from archentity group by uuid
having deleted is null);");

    populateDropDown("tabgroup1/tab2/entity", entities);
}

entity_id = null;
saveEntity() {
    callback = "entity_id= getLastSavedRecordId(); update();";
    saveTabGroup("tabgroup1", entity_id, null, null, callback);
}

loadEntity() {
    entity_id = getFieldValue("tabgroup1/tab2/entity");
    showTabGroup("tabgroup1", entity_id);
}

clearEntity() {
    newTabGroup("tabgroup1");
    entity_id = null;
}

update();
```

How it works

- The data schema has defined a archaeological entity with two attributes name and value. These are specified using property elements.
- The ui schema defines a ui with a single tab group and 2 tabs. The first tab contains a name text view and a value text view. These are mapped to the Simple archaeological element using the faims_arch_ent_type attribute and faims_attribute_name & faims_attribute_type attributes.
- The ui logical now uses the api calls provided [here](#) to save the data to the database and load data back from the database.

Automated Saving/Loading Relationships

Define a relationship in the data schema.

```

<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="data_schema.xsl"?>
<dataSchema name="SimpleModule" preparer="Your Name">
  <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="name" isIdentifier="true">
      <bundle>DOI</bundle>
    </property>
  </RelationshipElement>
</dataSchema>

```

This data schema defines a single relationship called "Simple" with one attribute called name.

Now we define a ui schema to save this relationship.

```

<h:html xmlns="http://www.w3.org/2002/xforms"
  xmlns:h="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jr="http://openrosa.org/javarosa">
  <h:head>
    <h:title>Simple Example</h:title>

    <model>
      <instance>
        <faims id="simple_example">
          <tabgroup1>
            <tab1>
              <name></name>
              <save></save>
              <clear></clear>
            </tab1>
            <tab2>
              <relationship></relationship>
              <load></load>
            </tab2>
          </tabgroup1>
        </faims>
      </instance>
      <bind nodeset="/faims/tabgroup1/tab1/name" type="string"/>
    </model>
  </h:head>

  <h:body>
    <group ref="tabgroup1" faims_rel_type="AboveBelow">
      <label></label>
      <group ref="tab1">
        <label>Tab 1</label>
        <input ref="name" faims_attribute_name="name"

```

```
faims_attribute_type="freetext">
  <label>Name:</label>
</input>
<trigger ref="save">
  <label>Save</label>
</trigger>
<trigger ref="clear">
  <label>Clear</label>
</trigger>
</group>
<group ref="tab2">
  <label>Tab 2</label>
  <select1 ref="relationship">
    <label>Relationship:</label>
    <item>
      <label>dummy</label>
      <value>dummy</value>
    </item>
  </select1>
  <trigger ref="load">
    <label>Load</label>
  </trigger>
</group>
</group>
```

```
</h:body>
</h:html>
```

Now let use the logic to save and load relationship to and from the database.

```
// add click events for buttons
onEvent("tabgroup1/tab1/save", "click", "saveRelationship()");
onEvent("tabgroup1/tab1/clear", "click", "clearRelationship()");
onEvent("tabgroup1/tab2/load", "click", "loadRelationship()");

update() {
  Object relationships = fetchAll("select relationshipid, relationshipid from
relationship where uuid || relntimestamp in ( select relationshipid ||
max(relntimestamp) from relationship group by relationshipid having deleted is
null);");

  populateDropDown("tabgroup1/tab2/relationship", relationships);
}

rel_id = null;
saveRelationship() {
  callback = "rel_id = getLastSavedRecordId(); update();";
  saveTabGroup("tabgroup1", rel_id, null, null, callback);
}

loadRelationship() {
  rel_id = getFieldValue("tabgroup1/tab2/relationship");
  showTabGroup("tabgroup1", rel_id);
}

clearRelationship() {
  newTabGroup("tabgroup1");
  rel_id = null;
}

update();
```

Fetching Entities and Relationships

To load entities manually use the following.

e.g. `fetchArchEnt`

```
entity = fetchArchEnt("10000112441409729");
// useful methods on entity
id = entity.getId();
type = entity.getType();
attributes = entity.getAttributes();
geometry_list = entity.getGeometryList();
has_conflict = entity.isForked();
```

In the example, the function will return an archaeological entity with uuid 10000112441409729 if the uuid exists or null.

To load relationships manually use the following.

e.g. `fetchRel`

```
rel = fetchRel("10000112441409730");  
// useful methods on relationship  
id = rel.getId();  
type = rel.getType();  
attributes = rel.getAttributes();  
geometry_list = rel.getGeometryList();  
has_conflict = rel.isForked();
```

In the example, the function will return a relationship with relationshipid 10000112441409730 if the relationshipid exists or null.

To query the database use the following api.

e.g. fetchOne

```
result = fetchOne("select vocabid, vocabname from vocabulary left join attributekey  
using (attributeid) where attributename = 'type'");
```

FetchOne will only return one row from the query even though the query may return more than one result.

e.g. fetchAll

```
results = fetchAll("select vocabid, vocabname from vocabulary left join attributekey  
using (attributeid) where attributename = 'type'");
```

FetchAll will return all values from the query.

e.g. fetchEntityList

```
entities = fetchEntityList("small");
```

This will return a list of entities of a particular type with each row having the first item equal the id of the entity and the second item equal the identifier of the entity. This is useful to populate list and dropdowns etc.

e.g. fetchRelationshipList

```
abovebelow = fetchRelationshipList("abovebelow");
```

This will return a list of relationships of a particular type with each row having the first item equal the id of the relationship and the second item equal the identifier of the relationship. This is useful to populate list and dropdowns etc.

Saving Archaeological Entities and Relationships

To save entities manually use the following.

e.g. saveArchEnt

```
attributes = createAttributeList();
name = "name";
text = "some text";
vocab = null;
measure = null;
certainty = null;
attributes.add(createEntityAttribute(name, text, vocab, measure, certainty));
entity_id = saveArchEnt(null, "small", null, attributes);
```

To save relationships manually use the following.

e.g. saveRel

```
attributes = createAttributeList();
name = "name";
text = "some text";
vocab = null;
certainty = null;
attributes.add(createRelationshipAttribute(name, text, vocab, certainty));
rel_id = saveRel(null, "abovebelow", null, attributes);
```

Deleting Archaeological Entities and Relationships

There are a couple of apis to delete entities and relationships from the database.

e.g. deleteArchEnt

```
deleteArchEnt('10000112441409729');
```

This will delete the archaeological entity with uuid 10000112441409729

e.g. deleteRel

```
deleteRel('10000112441409730');
```

This will delete the relationship with relationshipid 10000112441409730

Deleting Attributes

To delete a particular attribute of an entity or relationship you can use the overloaded createEntityAttribute and createRelationshipAttribute to specify that the attributes to be saved are deleted.

e.g. createEntityAttribute

```
attribute = createEntityAttribute('name', null, null, null, null, true);
```

This will create an attribute for 'name' with deleted set to true. Add this to the attribute list when you save the entity and it will mark the name attribute as deleted.

e.g. createRelationshipAttribute

```
attribute = createRelationshipAttribute('name', null, null, null, true);
```

This will create an attribute for 'name' with deleted set to true. Add this to the attribute list when you save the relationship and it will mark the name attribute as deleted.

Relating Entities

To add an existing entity to an existing relationship use the addReIn api.

e.g. addReIn

```
addReIn('10000112441409729', '10000112441409730', 'Above');
```

This will add entity with uuid 10000112441409729 to relationship with relationshipid 10000112441409730 with the verb 'Above'.

Getting Field Values, Annotations and Certainty

Given the saving and loading examples provided getting field values, annotations and certainty are done by using the api calls getFieldValue, getFieldAnnotation and getFieldCertainty.

e.g. getFieldValue

```
String value = getFieldValue("tabgroup1/tab1/name");

List pairs = getFieldValue("tabgroup1/tab1/pictures");
for (NameValuePair pair : pairs) {
    value = pair.getName();
}
```



For views like checkbox group, files list, camera picture gallery and video picture gallery the returned value is not a string but a list of name value pairs.

e.g. getFieldAnnotation

```
getFieldAnnotation("tabgroup1/tab1/name")
```

e.g. getFieldCertainty

```
getFieldCertainty("tabgroup1/tab1/name")
```

The input string references the view in the ui schema. The path matches <tabgroup>/<tab>/<name>. For more information on what these functions do please refer to the api calls provided [here](#).



Note that since not all fields supports certainty and annotation, all calls to the fields that do not support the certainty and annotation will result in showing a warning dialog.

Setting Field Values, Annotations and Certainty

The value of the fields can also be set by using the logic by calling setFieldValue, setFieldAnnotation, and setCertainty.

e.g. `setFieldValue`

```
setFieldValue("tabgroup1/tab1/name", "value1")

List pairs = new ArrayList();
pairs.add(new NameValuePair("value1", true);
pairs.add(new NameValuePair("value2", false);
setFieldValue("tabgroup1/tab1/pictures", pairs);
```

e.g. `setFieldAnnotation`

```
setFieldAnnotation("tabgroup1/tab1/name", "value1")
```

e.g. `setFieldCertainty`

```
setFieldCertainty("tabgroup1/tab1/name", "0.5")
```



Note that since not all fields supports certainty and annotation, all calls to the fields that do not support the certainty and annotation will result in showing a warning dialog.

Event Handling

Referring to the saving / loading examples provided above adding events to the ui are done using the following.

e.g. `onEvent`

```
onEvent("tabgroup1/tab1/save", "click", "saveEntity()");
```

`OnEvent` supports `delayclick`, `click`, `load`, and `show` events. In the example, it shows that when `"tabgroup1/tab1/save"` is clicked, the `saveEntity` will be called. The `click` event is dispatched when a view is touched, the `load` event is dispatched when a tabgroup is first shown and the `show` event is dispatched when everytime a tabgroup is shown. The `delay click` is a special `click` event that will only be executed every one second.

e.g. `onFocus`

```
onFocus("tabgroup1/tab1/name", "showWarning(\"focus\", \"it is focus\")",
"showWarning(\"blur\", \"it is blur\")");
```

`OnFocus` supports `focus` and `blur` events. The example shows that when the field `"tabgroup1/tab1/name"` is focused, the warning dialog `"focus"` will appear, meanwhile when it is blurred, the warning dialog `"blur"` will appear.



Note that if the focus callback or blur callback is not defined, the callback would not be executed.

Alert, Toast, Busy and Warning

Alert, toast, and warning are useful to show information to the user.

e.g. `showAlert`

```
showAlert("alert", "Do you want to navigate to the next page?", "goToNextPage()",
"stayInCurrentPage()")
```

The example shows that show alert could be useful in the scenario of moving page. The user would see a dialog asking for moving page, when the user press OK, the goToNextPage() function will be executed while pressing Cancel will result in calling the stayInCurrentPage() function. Note that the goToNextPage() and stayInCurrentPage() are not part of api calls and depends on ui designer to create them.

e.g. showToast

```
showToast("Starting GPS")
```

ShowToast is usefull to show a short period toast to the user with certain message. It does not block the user unlike the alert dialog or warning dialog.

e.g. showWarning

```
showWarning("warning","This is a warning")
```

When showWarning is called, a warning dialog will appear and shows the title and message as specified in the function. The user then needs to press OK button to dismiss the dialog.

e.g showBusy

```
dialog = showBusy("loading module", "please wait")
...
dialog.dismiss(); // to close the dialog
```

When showBusy is called, a busy dialog will appear and shows the title and message as specified in the function. The user can only close the dialog by dismissing it in the logic.

Drop Downs, Radio Button Groups, CheckBox Groups, Lists, Picture Gallery

The api populateDropDown, populateRadioGroup, and populate CheckBoxGroup adds the ability to set selection options from the database or from the logic.

e.g. populateDropDown

```
Object entities = fetchAll("select uuid, uuid from archentity where uuid ||
aenttimestamp in ( select uuid || max(aenttimestamp) from archentity group by uuid
having deleted is null);");

populateDropDown("tabgroup1/tab1/entities", entities);
```

e.g. populateRadioGroup

```
Object types = fetchAll("select vocabid, vocabname from vocabulary left join
attributekey using (attributeid) where attributename = 'type';");

populateRadioGroup("tabgroup1/tab1/types", types);
```

e.g. populateCheckBoxGroup

```
Object locations = fetchAll("select vocabid, vocabname from vocabulary left join
attributekey using (attributeid) where attributename = 'location';");

populateCheckBoxGroup("tabgroup1/tab1/locations", locations);
```


e.g. populateList and getListItemValue

```
Object users = fetchAll("select userid,(fname || ' ' || lname) as name from user;");

populateList("user/tab1/userlist",users);

onEvent("user/tab1/userlist","click","showToast(\"getListItemValue()\")")
```

The example shows how to fetch users from the database and populate a list. By binding the click event to the list, when clicking on the list, it will execute the callback which in this case show a toast containing the value of the clicked by calling the getListItemValue.

 getListItemValue returns the last selected item on the clicked list.

To populate a picture gallery from a query use the following.


e.g. populatePictureGallery

```
Object pictures = fetchAll("select vocabid, vocabname, pictureurl from vocabulary left
join attributekey using (attributeid) where attributename = 'picture';");

populatePictureGallery("tabgroup1/tab1/picture", pictures);
```

For this to work you need to add the picture urls to the vocab in the data schema as follows. The picture urls are relative to the modules root directory.

```
...
<property type="dropdown" name="picture" minCardinality="1" maxCardinality="1">
  <bundle>DOI</bundle>
  <lookup>
    <term pictureURL="pictures/cugl69808.jpg">cugl69808.jpg</term>
    <term pictureURL="pictures/cugl69807a.jpg">cugl69807a.jpg</term>
    <term>None</term>
  ...
</property>
...
```


 Note that the collection pictures should contain of the vocabid, vocabname, and picture url in order to work.

To populate a camera or video gallery use the following.

e.g. populateCameraPictureGallery and populateVideoGallery

```
photos = new ArrayList();
photos.add(photoUrl);
populateCameraPictureGallery("tabgroup1/tab1/photos", photos);

videos = new ArrayList();
videos.add(videoUrl);
populateVideoGallery("tabgroup1/tab1/videos", videos);
```

 The photo and video urls need to be absolute path urls. For a better example look at the file attachment examples below.

Showing Tabs & Tab Groups

To show tabs or tab groups use the following apis.

e.g. newTab

```
newTab( "tabgroup1/tab2" )
```

The newTab will open the tab specified in the path <tabgroupname>/<tabname> and clear out the values if it has been answered.

e.g. newTabGroup

```
newTabGroup( "tabgroup1" )
```

The newTabGroup will open the tab group specified in the path <tabgroupname> and clear out the values if it has been answered.

e.g. showTab

```
showTab( "tabgroup1/tab1" )
```

The showTab will open the tab specified in the path <tabgroupname>/<tabname> and reserve the values for each fields if it has been answered.

e.g. showTab with uuid

```
showTab( "tabgroup1/tab1" , "100001242124" )
```

The showTab with uuid will open the tab specified in the path <tabgroupname>/<tabname> and load the values from the records specified by the id to the related fields in the tab. So it would not change the value in other tabs.

e.g. showTabGroup

```
showTabGroup( "tabgroup1" )
```

The showTabGroup will open the tab group specified in the path <tabgroupname> and reserve the values for each fields if it has been answered.

e.g. showTabGroup with uuid

```
showTabGroup( "tabgroup1" , "100001242124" )
```

The showTabGroup with uuid will open the tab group specified in the path <tabgroupname> and load the values from the records specified by the id to the related fields in the tabgroup.

Leaving Tabs and Tab Groups

To close tabs and tabgroups the cancelTab and cancelTabGroup apis are available.

e.g. cancelTab

```
cancelTab( "tabgroup1/tab1" , true )
```

The cancelTab will close the tab specified in the path <tabgroupname>/<tabname>. If the warn argument is set to true and if there are any new changes to the fields (value, annotation, certainty) a dialog will pop up asking whether the user wants to cancel the tab or not. If the user chooses OK then the tab will close else it remains open. If the warn argument is set to false then tab will be closed regardless of any changes in the tab.



Its best practice after calling cancelTab to use showTab to open a new tab for the user

e.g. cancelTabGroup

```
cancelTabGroup("tabgroup1", true)
```

The cancelTabGroup will close the tab group specified in the path <tabgroupname>. It works similar to cancelTab but instead of closing the tab it closes the entire tab group.

Date & Time

There is a special method to show the current time to the ui that can be specified from the logic by calling getCurrentTime. GetCurrentTime returns a string containing current time in the format of 'YYYY-MM-dd hh:mm:ss'

e.g. getCurrentTime

```
time = getCurrentTime();
setFieldValue("tabgroup5/tab3/lastsuccess", time);
```

The example shows that we can set a field with the current time and the field should have faims_read_only attribute set to be true since the current time should not be editable by the user.

Module Metadata

Module metadata can be shown on the UI by calling the following apis.

e.g. Module metadata example

```
setFieldValue("tabgroup1/tab1/name", getModuleName());
setFieldValue("tabgroup1/tab1/id", getModuleId());
setFieldValue("tabgroup1/tab1/season", getModuleSeason());
setFieldValue("tabgroup1/tab1/description", getProjectDescription());
setFieldValue("tabgroup1/tab1/permit_no", getPermitNo());
setFieldValue("tabgroup1/tab1/permit_holder", getPermitHolder());
setFieldValue("tabgroup1/tab1/contact_address", getContactAndAddress());
setFieldValue("tabgroup1/tab1/participants", getParticipants());
setFieldValue("tabgroup1/tab1/permit_issued_by", getPermitIssuedBy());
setFieldValue("tabgroup1/tab1/permit_type", getPermitType());
setFieldValue("tabgroup1/tab1/copyright_holder", getCopyrightHolder());
setFieldValue("tabgroup1/tab1/client_sponsor", getClientSponsor());
setFieldValue("tabgroup1/tab1/land_owner", getLandOwner());
setFieldValue("tabgroup1/tab1/has_sensitive_data", hasSensitiveData());
```

In the example, the module metadata is obtained and shown on the fields. It is important to give the field faims_read_only attribute to be true since we do not want the user to be able to edit the module metadata on the device.

The server will have the ability to edit the module metadata.

Maps and GIS

The following examples will show how to render maps, vectors and draw geometry.

Rendering map

Given the following map view defined in the ui schema.

```

...
<tab1>
  <map></map>
</tab1>
...
<group ref="tab1" faims_scrollable="false">
  <input ref="map" faims_map="true">
    <label>Map:</label>
  </input>
</group>
...

```

Add base map

To add a base raster layer use the following. There can only be a single base layer.

```
showBaseMap("tabgroup1/tab1/map", "raster map" "map.tif");
```



The raster map must be in projection EPSG:3857



map.tif url is relative to the root of the modules folder

Add raster map

To add a raster layer use the following. You can have multiple raster layers.

```
showRasterLayer("tabgroup1/tab1/map", "raster map" "map.tif");
```

Add shape vector layer



This has been deprecated by spatial layer

To add a shape vector layer use the following.

```

ps = createPointStyle(10, Color.BLUE, 0.2f, 0.5f);
ls = createLineStyle(10, Color.GREEN, 0.05f, 0.3f, null);
pos = createPolygonStyle(10, Color.parseColor("#440000FF"), createLineStyle(10,
Color.parseColor("#AA000000"), 0.01f, 0.3f, null));
ts = createTextStyle(10, Color.WHITE, 40, Typeface.SANS_SERIF);
showShapeLayer("tabgroup1/tab1/map", "Shape Layer", "shape.shp", ps, ls, pos, ts);

```



The shape file must be in projection EPSG:3857

Add spatial vector layer

To add a spatial vector layer use the following.

```
ps = createPointStyle(10, Color.BLUE, 0.2f, 0.5f);
ls = createLineStyle(10, Color.GREEN, 0.05f, 0.3f, null);
pos = createPolygonStyle(10, Color.parseColor("#440000FF"), createLineStyle(10,
Color.parseColor("#AA000000"), 0.01f, 0.3f, null));
ts = createTextStyle(10, Color.WHITE, 40, Typeface.SANS_SERIF);
table = // specify table in database
idcolumn = // specify id column name
labelcolumn = // specify label column name
showSpatialLayer("tabgroup1/tab1/map", "Spatial Layer", "spatial.sqlite", table,
idcolumn, labelcolumn, ps, ls, pos, ts);
```



The spatial database must be in projection EPSG:3857

Add database vector layer

To add a database vector layer use the following.

```
ps = createPointStyle(10, Color.BLUE, 0.2f, 0.5f);
ls = createLineStyle(10, Color.GREEN, 0.05f, 0.3f, null);
pos = createPolygonStyle(10, Color.parseColor("#440000FF"), createLineStyle(10,
Color.parseColor("#AA000000"), 0.01f, 0.3f, null));
ts = createTextStyle(10, Color.WHITE, 40, Typeface.SANS_SERIF);
isEntity = // specify where to load entity or relationships
queryName = // specify the name of the sql query
querySql = // specify the query sql to run against the database
showDatabaseLayer("tabgroup1/tab1/map", "Database Layer", isEntity, queryName,
querySql, ps, ls, pos, ts);
```

Add canvas layer

You can create canvas layers to draw geometry onto using the following.

```
layerId = createCanvasLayer("tabgroup1/tab1/map", "Canvas Layer");
```




Keep a reference to the layerId returned by createVectorLayer. This can be used to draw points onto the layer or removing the layer entirely.

Map Controls

To set the map focus point use the following:

```
// australia
lon = 151.23f
lat = -33.58f
setMapFocusPoint("tabgroup1/tab1/map", lon, lat);
```

Value could be double or float.


 The point must be in module projection.

To set the map rotation use the following.

```
setMapRotation("tabgroup1/tab1/map", 90.0f);
```


To set the map tilt use the following

```
setMapTilt("tabgroup1/tab1/map", 90.0f);
```

 The minimum tilt is 30.0f

To set the map zoom use the following.

```
setMapZoom("tabgroup1/tab1/map", 17.0f);
```

 There are 18 levels of zoom defined for the raster map therefore zoom levels above 18 might not rendered as well.

Center map using GPS

To center the map the GPS position use the following.

```
centerOnCurrentPosition("tabgroup1/tab1/map");
```

Locking / Unlocking the map

Locking the map keeps the map at a 2D perspective. This is useful when drawing geometry on to the map. To lock the map use the following.

```
lockMapView("tabgroup1/tab1/map", true);
```

To unlock the map use the following.

```
lockMapView("tabgroup1/tab1/map", false);
```

Adding map event listeners

To be able to draw points on the map or select geometry on the map you can add a map event listener. To add click and select listener to the map use the following.


```
onMapEvent("tabgroup1/tab1/map", "clickCallback()", "selectCallback()");

clickCallback() {
    point = getMapPointClicked();
    ...
}

selectCallback() {
    geomId = getMapGeometrySelected();
    ...
}
```

The `getMapPointClicked` method will hold the last clicked value on the map and the `getMapGeometrySelected` method will hold the last selected geometry on the map.

Styling Vectors

Styling is used when loading vector layers or creating geometry. Use the following to create point, line, polygon and text styles.

e.g. point style

```
minZoom = 10;
color = Color.RED;
size = 0.1f;
pickingSize = 0.3f;
pointStyle = createPointStyle(minZoom, color, size, pickingSize);
```

e.g. line style

```
minZoom = 10;
color = Color.RED;
width = 0.1f;
pickingWidth = 0.3f;
lineStyle = createLineStyle(minZoom, color, width, pickingWidth, pointStyle); // note:
point style can be null
```

e.g. polygon style

```
minZoom = 10;
color = Color.RED;
pointStyle = createPolygonStyle(minZoom, color, lineStyle); // note: line style can be
null
```

e.g. text style


```
minZoom = 10;
color = Color.RED;
fontSize = 40;
font = Typeface.SANS_SERIF;
textStyle = createTextStyle(minZoom, color, fontSize, font);
```

Drawing points

To draw a point on the map you can use the following.

```
lon = 151.23f
lat = -33.58f
point = createPoint(lon, lat);
geomId = drawPoint("tabgroup1/tab1/map", layerId, point, pointStyle);
```

Keep a reference to the geomId so you can later clear the geometry or draw overlays for it.


 The points must be in the modules projection.

Drawing lines

To draw a line on the map you can use the following.

```
points = new ArrayList();
points.add(createPoint(x1, y1));
points.add(createPoint(x2, y2));
points.add(createPoint(x3, y3));
geomId = drawLine("tabgroup1/tab1/map", layerId, points, lineStyle);
```

Keep a reference to the geomId so you can later clear the geometry or draw overlays for it.


 The lines must be in the modules projection.

Drawing polygons

To draw a polygon on the map you can use the following.

```
points = new ArrayList();
points.add(createPoint(x1, y1));
points.add(createPoint(x2, y2));
points.add(createPoint(x3, y3));
drawPolygon("tabgroup1/tab1/map", layerId, points, polygonStyle);
```

Keep a reference to the geomId so you can later clear the geometry or draw overlays for it.

 The polygons must be in the modules projection.

Moving vector geometry

To move a vector on the map you must first highlight it, then call prepareHighlightTransform, move the vector then call doHighlightTransform.

```
onMapEvent("tabgroup1/tab1/map", "onMapClick()", "onMapSelect()");

onMapSelect() {
  geomId = getMapGeometrySelected();
  addGeometryHighlight("tabgroup1/tab1/map", currentGeometryId); // add it to the
highlight list
  prepareHighlightTransform("tabgroup1/tab1/map"); // prepare it for transform
}
```

Now you can move the map normally. This way you can adjust the size, orientation and position of the geometry by dragging the map around. Once you happy with the new position of the geometry you can replace the selected geometry by replacing it using the following.

```
doHighlightTransform("tabgroup1/tab1/map"); // transform the vector to its new
position
removeGeometryHighlight("tabgroup1/tab1/map", geomId) // remove the geometry
highlight, or call clearGeometryHighlights("tabgroup1/tab1/map") to clear all
highlights
```

Clearing Geometry

To clear a single geometry from the map use the following.

```
clearGeometry("tabgroup1/tab1/map", geomId);
```

To clear a list of geometry use the following.

```
clearGeometryList("tabgroup1/tab1/map", geomIdList);
```

Saving GIS to Entities / Relationships

Referring to the save entities / relationships examples above. To save GIS data you can use the following.

```
...
collection = getGeometryHighlights();
...
saveArchEnt(entityId, "simple", collection, attributes);
```

Or if you want to save the entire layer to the entity you can use the following.

```
...
collection = getGeometryList("tabgroup1/tab1/map", layerId);
...
saveArchEnt(entityId, "simple", collection, attributes);
```

Loading GIS from Entities / Relationships

Referring to the load entities / relationships examples above. To load GIS data you can use the following.

```
Object entity = fetchArchEnt(entityId);

geometryList = entity.getGeometryList();

for (Geometry geomId : geometryList) {
    if (geom instanceof Polygon) {
        drawGeometry("tabgroup1/tab1/map", layerId, geomId, polygonStyleSet);
    } else if (geom instanceof Line) {
        drawGeometry("tabgroup1/tab1/map", layerId, geomId, lineStyleSet);
    } else if (geom instanceof Point) {
        drawGeometry("tabgroup1/tab1/map", layerId, geomId, pointStyleSet);
    }
}
```

Add Database Layer Queries

To add database layer queries to load database layers via the layer manager use the following.

```
isEntity = true;
queryName = "All entities";
querySQL =
    "SELECT uuid, max(aenttimestamp) as aenttimestamp\n" +
    " FROM archentity join aenttype using (aenttypeid)\n" +
    " where archentity.deleted is null\n" +
    "   and lower(aenttypename) != lower('gps_track')\n" +
    " group by uuid\n" +
    " having max(aenttimestamp)";
addDatabaseLayerQuery("control/map/map", queryName, querySQL);
```

Add TrackLog layer Queries

To add track log layer queries to load track log layers via the layer manager use the following.

```
addTrackLogLayerQuery("control/map/map", "track log entities",
    "SELECT uuid, max(aenttimestamp) as aenttimestamp\n" +
    " FROM archentity join aenttype using (aenttypeid)\n" +
    " where archentity.deleted is null\n" +
    "   and lower(aenttypename) = lower('gps_track')\n" +
    " group by uuid\n" +
    " having max(aenttimestamp)");
```

Adding Selection Tool Queries

To add database and legacy selection queries use the following.

```
// create a query builder
queryBuilder = createQueryBuilder(
    "select uuid\n" +
    "  from latestNonDeletedArchent\n" +
    " JOIN latestNonDeletedAentValue using (uuid)\n" +
    " join aenttype using (aenttypeid)\n" +
    " LEFT OUTER JOIN vocabulary using (vocabid, attributeid) \n" +
    " where lower(aenttypename) = lower(?) \n" +
    "   group by uuid");

// add a parameter type with default argument
queryBuilder.addParameter("Type", "Structure");

// add the query builder
addSelectQueryBuilder("control/map/map", "Select entity by type", queryBuilder);

// similar process for legacy data
queryBuilder = createLegacyQueryBuilder("Select PK_UID from Geology100_Sydney where
PK_UID = ?");

queryBuilder.addParameter("ID", null);

addLegacySelectQueryBuilder("control/map/map", "Select geometry by id",
"files/data/maps/sydney.sqlite", "Geology100_Sydney", queryBuilder);
```

setLayerVisible

To change the visibility of a map layer use the following

```
setLayerVisible("tabgroup1/tab1/map", true);
```

Convert projection of points

To convert a point from one projection to another use the following.

```
MapPos p = new MapPos(x, y);
MapPos np = convertFromProjToProj("4326", "3875", p);
```

Enable / Disable tools view

To enable or disable the tools bar and layers bar use the following

```
setToolsEnabled(true);
```

Bind events to tools

The create point, line and polygon tools trigger the tool create event when a geometry is created and the Load tool triggers a tool load event when a geometry is selected. Use the following to bind callback to those events.

```
onToolEvent("tabgroup1/tab1/map", "create", "onCreate");
onToolEvent("tabgroup1/tab1/map", "load", "onLoad");

onCreate() {
  id = getMapGeometryCreated(); // geometry id of the vector element
}

onLoad() {
  id = getMapGeometryLoaded(); // uuid or relationshipid of the vector element
  type = getMapGeometryLoadedType(); // either "entity" or "relationship"
}
```

Refresh map

Refreshing the map will cause all layer to re-render themselves. This is useful if you have made changes that could effect the map.

e.g. refreshMap

```
refreshMap("tabgroup1/tab1/map");
```

GPS

The following examples will show how to use GPS.

Get GPS position

To get the current GPS position use the following.

```
location = getGPSPosition();
lon = location.getLongitude();
lat = location.getLatitude();
```

To get the current GPS position using the projection selected by user, use the following

```
location = getGPSPositionProjected();
```

Get GPS accuracy

To get the current GPS estimated accuracy use the following.

```
accuracy = getGPSEstimatedAccuracy();
```

or to specify which type of gps to use i.e. internal or external use the following

```
accuracy = getGPSEstimatedAccuracy("internal");
```

Get GPS heading

To get the current GPS heading use the following.

```
heading = getGPSHeading();
```

or to specify which type of gps to use i.e. internal or external use the following

```
heading = getGPSHeading("internal");
```

Set GPS interval

To configure the delay between GPS updates you can use the following.

```
setGPSUpdateInterval(5);
```

Setup Track Logs

The track log allows the user to track their progression throughout the day. The track log has two modes time and distance. The time mode allows the user to specify the time interval (seconds) between callbacks and the distance mode allows the user to specify the distance threshold (meters) between callbacks.

e.g. startTrackingGPS and stopTrackingGPS

```

onEvent("controls/tabl/starttrackingtime", "click", "startTrackingGPS(\"time\", 10,
\"saveTimeGPSTrack()\");");
onEvent("controls/tabl/starttrackingdistance", "click",
"startTrackingGPS(\"distance\", 10, \"saveDistanceGPSTrack()\");");
onEvent("controls/tabl/stoptracking", "click", "stopTrackingGPS()");

saveTimeGPSTrack() {
    List attributes = createAttributeList();
    attributes.add(createEntityAttribute("gps_type", "time", null, null, null));
    saveGPSTrack(attributes);
}

saveDistanceGPSTrack() {
    List attributes = createAttributeList();
    attributes.add(createEntityAttribute("gps_type", "distance", null, null, null));
    saveGPSTrack(attributes);
}

saveGPSTrack(List attributes) {
    position = getGPSPosition();
    if (position == null) return;

    attributes.add(createEntityAttribute("gps_user", "" + user.getUserId(), null, null,
null));
    attributes.add(createEntityAttribute("gps_timestamp", "" + getCurrentTime(), null,
null, null));
    attributes.add(createEntityAttribute("gps_longitude", "" + position.getLongitude(),
null, null, null));
    attributes.add(createEntityAttribute("gps_latitude", "" + position.getLatitude(),
null, null, null));
    attributes.add(createEntityAttribute("gps_heading", "" + getGPSHeading(), null, null,
null));
    attributes.add(createEntityAttribute("gps_accuracy", "" + getGPSEstimatedAccuracy(),
null, null, null));

    positionProj = getGPSPositionProjected();

    Point p = new Point(new MapPos(positionProj.getLongitude(),
positionProj.getLatitude()), null, (PointStyle) null, null);
    ArrayList l = new ArrayList();
    l.add(p);

    saveArchEnt(null, "gps_track", l, attributes);
}

```

The following example setups a time and distance track log and saves an entity to the database each time the callback is triggered.

Syncing

The following examples will show how to sync the database and files with the server and other apps.

Pull database from server

To pull the entire server database onto the app use the following api.


```
pullDatabaseFromServer("onComplete()");

onComplete() {
  showToast("finished pulling database");
}
```

Push Database to server

To push the entire app database to the server use the following api.

```
pushDatabaseToServer("onComplete()");

onComplete() {
  showToast("finished pushing database");
}
```

Database syncing

To use database syncing you must first enable it using the following code.

```
setSyncEnabled(true);
```

Now the database will be set to sync with the server. An indicator on the top right will let you know when syncing is occurring. Green indicates syncing is working as normal, Orange to indicate syncing is in progress and Red indicates that syncing is not working.

To adjust sync intervals and delays use the following.

```
setSyncMinInterval(10.0f);
setSyncMaxInterval(20.0f);
setSyncDelay(5.0f);
```

The min sync interval lets you configure the time between syncs. The sync delay sets a period of time to delay the sync interval if the previous sync has failed. e.g. if your sync interval is 10 seconds and the sync delay is 5 seconds then if the sync fails then the next sync will occur in 15 seconds and if it fails again the next sync will occur in 20 secs etc. The sync max interval sets the limit for the maximum sync interval. Once a sync completes successfully the sync interval is reset the minimum sync interval.

Stop syncing

To stop syncing use the following.

```
setSyncEnabled(false);
```

Sync Event

To track sync progress in logic you can bind to the sync event.

```
onSyncEvent("onSyncStart()", "onSyncSuccess()", "onSyncFailure()");

onSyncStart() {
  showToast("sync started");
}

onSyncSuccess() {
  showToast("sync success");
}

onSyncFailure() {
  showToast("sync failed");
}
```

File syncing

To use file syncing you must first enable normal syncing and then enable file syncing. Use the following to enable database syncing.

```
setSyncEnabled(true);
setFileSyncEnabled(true);
```

File Attachments

To attach files, photos, videos and audios use the following apis.

Selecting files

To select a file you need to bring up the file chooser popup.

```
showFileBrowser("saveFile()");

saveFile() {
  filename = getLastSelectedFilename();
  filepath = getLastSelectedFilepath();
}
```



The filename contains only the files name where the filepath contains the absolute path to the file.

Taking Photos

To take a photo use the following.

```
openCamera("savePhoto()");

savePhoto() {
  url = getLastPictureFilePath();
}
```



The url is the absolute path to the file.

Recording Video

To record a video use the following.

```
openVideo("saveVideo()");

saveVideo() {
  url = getLastVideoFilePath();
}
```



The url is the absolute path to the file.

Recording Audio

To record audio use the following.

```
recordAudio("saveAudio()");

saveAudio() {
  url = getLastAudioFilePath();
}
```



The url is the absolute path to the file.

Saving / Loading Files, Photos, Videos and Audio

Here is an example to quickly setup saving photos, videos, audios and files.

When saving files to an entity you must save each file to an attribute with type 'file'.

```
// attach files to file list view
onEvent("tabgroup1/tab1/attachFile", "attachFileTo(\"tabgroup1/tab1/files\");");

// attach audios to file list view
onEvent("tabgroup1/tab1/attachAudio", "attachAudioTo(\"tabgroup1/tab1/audios\");");

// attach picture to camera picture gallery
onEvent("tabgroup1/tab1/attachPicture",
"attachPictureTo(\"tabgroup1/tab1/pictures\");");

// attach video to camera picture gallery
onEvent("tabgroup1/tab1/attachVideo", "attachVideoTo(\"tabgroup1/tab1/videos\");");
```

View Attached Files

To view attached files for an entity or relationship used the following api.

e.g. viewArchEntAttachedFiles

```
viewArchEntAttachedFiles("10000112441409729");
```

This will show the attached files for an archaeological entity with uuid 10000112441409729.

e.g. viewRelAttachedFiles

```
viewRelAttachedFiles("10000112441409730");
```

This will show the attached files for an relationship with relationshipid 10000112441409730.

Misc

UI Logic persistence

When android runs low on resources the module activity can be destroyed if its in a suspended state and then restored when its resumed. This will result in a loss of variable state in the logic script. To restore variable state in the logic script use the following method to define the name of a single object that will saved and restored when the activity is destroyed and restored.

```
persistObject('varData');

// varData can be any object
varData = new ArrayList();
// add some data
varData.add(var1);
varData.add(var2);

// Once the script is restored then varData's state will be restored
```

Set User

Use the following api to set the current user of the app.

```
// create a new user with id 1, with first and last names
user = new User("1", "John", "Doe");
setUser(user);
```

Execute

Use the following to execute code.

```
callback = "showToast(\"this is a test\")";
execute(callback);
```

Validation

Adding a validation schema file to the module will allow you to validate your database records on the server and propagate them to the apps.

Here is an example of a validation schema.

```

<ValidationSchema>

  <RelationshipElement name='AboveBelow'>

    <property name='name'>

      <validator type='evaluator' cmd='spell.sh ?'>
        <cmd><![CDATA[spell.sh ?]]></cmd>
        <param type='field' value='freetext' />
      </validator>

      <!--<validator type='evaluator' cmd='spell.sh ?'>
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->

      <!--<validator type='evaluator' cmd='spell.sh ? ?'>
        <param type='field' value='freetext' />
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->

      <validator type='blankchecker'>
        <param type='field' value='freetext' />
      </validator>

      <!--<validator type='blankchecker'>
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->

      <!--<validator type='blankchecker'>
        <param type='field' value='freetext' />
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->

      <validator type='typechecker' datatype='text'>
        <param type='field' value='freetext' />
      </validator>

      <!--<validator type='typechecker' datatype='text'>
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->

      <!--<validator type='typechecker' datatype='text'>
        <param type='field' value='freetext' />
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
      </validator-->
    </property>
  </RelationshipElement>

```

```

    <validator type='querychecker'>
        <query><![CDATA[select length(?) < 20, 'Field value is too
long']]></query>
    <param type='field' value='freetext' />
    </validator>

    <!--<validator type='querychecker'>
        <query><![CDATA[select length(?) < 20, 'Field value is too long']]></query>
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
    </validator>-->

    <!--<validator type='querychecker'>
        <query><![CDATA[select length(?) < 20 AND ? like 'Test', 'Field value is too
long and does not contain Test']]></query>
        <param type='field' value='freetext' />
        <param type='query' value="select freetext from relnvalue join attributekey using
(attributeid) where relationshipid = ? and relnvaluetimestamp = ? and attributename =
'name';" />
    </validator>-->

</property>

<property name='location'>
    <validator type='blankchecker'>
        <param type='field' value='vocab' />
    </validator>
</property>

</RelationshipElement>

<ArchaeologicalElement name='small'>

    <property name='name'>

        <validator type='evaluator' cmd='spell.sh ?'>
            <param type='field' value='freetext' />
        </validator>

        <!--<validator type='evaluator' cmd='spell.sh ?'>
            <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
        </validator>-->

        <!--<validator type='evaluator' cmd='spell.sh ? ?'>
            <param type='field' value='freetext' />
            <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
        </validator>-->

        <validator type='blankchecker'>
            <param type='field' value='freetext' />
        </validator>

        <!--<validator type='blankchecker'>
            <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
        </validator>-->
    </property>
</ArchaeologicalElement>

```

```

<!--><validator type='blankchecker'>
  <param type='field' value='freetext' />
  <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
</validator>-->

<validator type='typechecker' datatype='text'>
  <param type='field' value='freetext' />
</validator>

<!--><validator type='typechecker' datatype='text'>
  <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
</validator>-->

<!--><validator type='typechecker' datatype='text'>
  <param type='field' value='freetext' />
  <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
</validator>-->

<validator type='querychecker'>
  <query><![CDATA[select length(?) < 20, 'Field value is too
long']]></query>
  <param type='field' value='freetext' />
</validator>

<!--><validator type='querychecker' query="select length(?) < 20, 'Field value is
too long'" >
  <query><![CDATA[select length(?) < 20, 'Field value is too long']]></query>
  <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
</validator>-->

<!--><validator type='querychecker' query="select length(?) < 20 AND ? like 'Test',
'Field value is too long and does not contain Test'" >
  <query><![CDATA[select length(?) < 20 AND ? like 'Test', 'Field value is too
long and does not contain Test']]></query>
  <param type='field' value='freetext' />
  <param type='query' value="select freetext from aentvalue join attributekey using
(attributeid) where uuid = ? and valuetimestamp = ? and attributename = 'name';" />
</validator>-->

</property>

<property name='value'>
  <validator type='blankchecker'>
    <param type='field' value='measure' />
    <param type='field' value='certainty' />
  </validator>

  <validator type='typechecker' datatype='real'>
    <param type='field' value='measure' />
  </validator>

  <validator type='typechecker' datatype='real'>
    <param type='field' value='certainty' />
  </validator>

```

```
</property>

<property name='location'>
  <validator type='blankchecker'>
    <param type='field' value='vocab' />
  </validator>
</property>
```



```
</ArchaeologicalElement>

</ValidationSchema>
```

How it works

To specify validation for a relationship or archaeological entity defined in the data schema you simply need add validation on the corresponding elements in the validation schema.

e.g. add validation for the AboveBelow relationship

```
...
<RelationshipElement name='AboveBelow'>
...
```

e.g. add validation for the small archaeological entity

```
...
<ArchaeologicalElement name='small'>
...
```

To add validation to a property of a relationship or archaeological entity do the following.

e.g. add validation to the name property of relationship

```
...
<RelationshipElement name='AboveBelow'>

  <property name='name'>
...

```

e.g. add validation to the name property of archaeological entity

```
...
<ArchaeologicalElement name='small'>

  <property name='name'>
...


```

There are four types of validators you can use. To use an validator you need to specify the type and the params for the validator

e.g


```
...
  <validator type='evaluator' cmd='spell.sh ?'>
    <param type='field' value='freetext' />
  </validator>
...
```

Here the validator is an evaluator which takes a command. The ? in the command is replaced by the param. Params can be type field which takes freetext, certainty, vocab and measure (for archaeological entities only) or queries which run query to return a value. You can also specify multiple params for a single evaluator.

 Examples have been provided above in the commented out sections.


Evaluator

This runs a program against the given params

 Examples have been provided above in the commented out sections.


Blank Checker

This checks if the values are not null, or an empty string against the given params

 Examples have been provided above in the commented out sections.


Type Checker

This checks if the values are integer, real or text against the given params.

 Examples have been provided above in the commented out sections.

Query Checker

This checks if the values are valid when running a query against the given params.


 Examples have been provided above in the commented out sections.

Arch16n translations

Adding Arch16n properties file to the module will allow you to translate reserved terms in view labels, buttons, popup messages and vocabulary.

Below is an example of an arch16n file.

```
entity=Arch16n Entity
name=Arch16n Name
value=Arch16n Value
superA=Arch16n Super A
superB=Arch16n Super B
superC=Arch16n Super C
superD=Arch16n Super D
locationA=Arch16n Location A
locationB=Arch16n Location B
locationC=Arch16n Location C
locationD=Arch16n Location D
typeA=Arch16n Type A
typeB=Arch16n Type B
typeC=Arch16n Type C
typeD=Arch16n Type D
```

 This is a standard java properties file so make sure the left hand side of the equals contains no spaces.

To replace terms in the ui schema use the following.

```
...
<group ref="tabgroup1" faims_archent_type="simple">
  <label>Simple Entity Example</label>
  <group ref="tab1">
    <label>{entity} Tab1</label>
    <input ref="name" faims_attribute_name="name" faims_attribute_type="freetext">
      <label>{name}:</label>
    </input>
  </group>
</group>
...
```

In this example {entity} will be replaced by Arch16n Entity and {name} will be replaced by Arch16n name.

To replace terms in the data schema use the following.

```
...
<property type="checklist" name="type">
  <bundle>DOI</bundle>
  <lookup>
    <term>{typeA}</term>
    <term>{typeB}</term>
    <term>{typeC}</term>
    <term>{typeD}</term>
  </lookup>
</property>
...
```

In this example {typeA} will be replaced by Arch16n Type A.

To replace terms in the ui logic use the following.

```
setFieldValue("tabgroup1/tab1/name", "{typeC}")
```

In this example the name field will be set the value Arch16n Type C.

1.2 Program Logic

FAIMS Mobile Application Program Logic Documentation.

The program logic page is an exhaustive list of every feature offered by our Beanshell logic. As a reference for module developers it provides a quick overview of possible functions that can be invoked in logic. It should be used in conjunction with the Cookbook as well as the Github repositories and sample modules therein, which illustrate the functions invoked.

Program Logic Support

- Persist Functionality
- Tab / Tab Group Functionality
- Dialog Functionality
- Setter / Getter Functionality
- Event Callback Functionality
- User Functionality
- Archaeological Entity / Relationship Functionality
- Navigation Functionality
- GPS Functionality
- Map Functionality
- Sync Functionality
- Static Data Functionality
- File attachment Functionality
- MISC

Persist Functionality

When android runs low on resources the project activity can be destroyed if its in a suspended state and then restored when its resumed. This will result in a loss of variable state in the logic script. To restore variable state in the logic script use the following method to define the name of a single object that will saved and restored when the activity is destroyed and restored. For more information on how to use this function please look at the cookbook.

- **name** the name of the object to restore

persistObject(String name) ;

Tab / Tab Group Functionality

Show the tab group with the following reference and clear all values in the tab group.

- **ref** the reference to the tab group

newTabGroup(String ref) ;

Show the tab with the following reference and clear all values in the tab.

- **ref** the reference to the tab

newTab(String ref) ;

Show the tab group with the following reference. This will retain all the current values in the tab group.

- **ref** the reference to the tab group

showTabGroup(String ref);

Show the tab group with the following reference and load the values of the supplied entity or relationship id into the tab group.

- **ref** the reference of the tab group
- **id** the id of the entity or relationship

showTabGroup(String ref, String id);

Show the tab with the following reference. This will retain all the current values in the tab.

- **ref** the reference to the tab

showTab(String ref) ;

Show the tab with the following reference and load the values of the supplied entity or relationship id into the tab.

- **ref** the reference to the tab
- **id** the id of the entity or relationship

showTab(String ref, String id) ;

Save the tab group with the following reference.

- **ref** the reference of the tab group
- **id** the id of the entity or relationship. Can be null to save new record
- **geometry** the geometry list of the record
- **attributes** additional attributes to save for the record
- **callback** code to execute when saving is finished

saveTabGroup(String ref, String id, List geometry, List attributes, String callback);

Save the tab with the following reference.

- **ref** the reference of the tab group
- **id** the id of the entity or relationship. Can be null to save new record
- **geometry** the geometry list of the record
- **attributes** additional attributes to save for the record
- **callback** code to execute when saving is finished

saveTab(String ref, String id, List geometry, List attributes, String callback);

Return the last saved record id.

return id the id of the last saved record using saveTabGroup or saveTab

getLastSavedRecordId() ;

Close the tab group with the following reference with an option to show a warning dialog if there are changes that haven't been saved.

- **ref** the reference to the tab group
- **warn** set to true to show a warning dialog if there are changes that haven't been saved

cancelTabGroup(String ref, boolean warn);

Close the tab with the following reference with an option to show a warning dialog if there are changes that haven't been saved.

- **ref** the reference to the tab
- **warn** set to true to show a warning dialog if there are changes that haven't been saved

cancelTab(String ref, boolean warn);

Dialog Functionality

Show a toast to the user with the given message, the toast will last for about 1 second.

- **message** the message to be shown to the user

showToast(String message) ;

Show an alert dialog to the user with the given message.

- **title** the title of the dialog
- **message** the message to be shown to the user
- **okCallback** the callback that is executed when Ok button is pressed
- **cancelCallback** the callback that is executed when Cancel button is pressed

showAlert(String title, String message, String okCallback, String cancelCallback);

Show a warning dialog to the user with the given message.

- **title** the title of the dialog
- **message** the message to be shown to the user

showWarning(String title, String message) ;

Show a busy dialog to the user with the given message.

- **title** the title of the dialog
- **message** the message to be shown to the user

showBusy(String title, String message) ;

Setter / Getter Functionality

Set the field with the following reference to the given value. If the field reference is not found a logic error dialog will appear. For more information on how to use this function please look at the cookbook.

- **ref** the reference to the field
- **value** the value to set the field to

setFieldValue(String ref, Object value);

Set the certainty of the field with the following reference to the given value If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **value** the value to set the certainty for the field to

setFieldCertainty(String ref, Object value);

Set the annotation of the field with the following reference to the given value If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **value** the value to set the annotation for the field to

setFieldAnnotation(String ref, Object value);

Get value of the field with the following reference. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field

return the value of the field, could be collection or String

Object getFieldValue(String ref);

Get certainty of the field with the following reference. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field

return the certainty of the field

Object getFieldCertainty(String ref);

Get annotation of the field with the following reference. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field

return the annotation of the field

Object getFieldAnnotation(String ref);

Get the current time of the application

return the current time e.g. 2013-01-20 13:20:01

String getCurrentTime();

Clear a dirty field with the following reference

- **ref** the reference to the field

clearFieldDirty(String ref) ;

Event Callback Functionality

Binding an event to the field with the given reference. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **type** the type of event, one of click, show, or load
- **callback** the callback that is executed when the event is triggered

onEvent(String ref, String type, String callback) ;

Binding a focus/blur event to the field with the given reference. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **focusCallback** the callback that is executed when focus event is triggered
- **blurCallback** the callback that is executed when blur event is triggered

onFocus(String ref, String focusCallback, String blurCallback) ;

User Functionality

Set the current user of the application. Is a requirement for insert records into the database.

- **user** the user of the application

setUser(User user);

Archaeological Entity / Relationship Functionality

Insert a new or update an existing archaeological entity record and return the id of the saved record.

- **entityId** the id of the entity to be saved, set to null to save a new record
- **entityType** the type of the entity to be saved, must be one specified in the data schema
- **geometry** the list of geometries to be associated with the entity
- **attributes** the list of attributes to be associated with the entity

return the id of the saved entity

saveArchEnt(String entityId, String entityType, List geometry, List attributes) ;

Insert a new or update an existing relationship record and return the id of the saved record.

- **relationshipId** the id of the relationship to be saved, set to null to save a new record
- **relationshipType** the type of the relationship to be saved, must be one specified in the data schema
- **geometry** the list of geometries to be associated with the relationship
- **attributes** the list of attributes to be associated with the relationship

return the id of the saved relationship

saveRel(String relationshipId, String relationshipType, List geometry, List attributes) ;

Deletes the specified archaeological entity.

- **entityId** the id of the entity to be deleted

return boolean value true if deleted, false if not

deleteArchEnt(String entityId);

Deletes the specified relationship.

- **relationshipId** the id of the relationship to be deleted

return boolean value true if deleted, false if not

deleteRel(String relationshipId);

Add an existing archaeological entity to an existing relationship and the verb of the relation.

- **entityId** the id of the entity
- **relationshipId** the id of the relationship
- **verb** the relation verb that is defined in the data schema for the specified relationship

addReln(String entityId, String relationshipId, String verb) ;

Create an attribute list.

return new attribute list

createAttributeList() ;

Create an entity attribute to be added to the attribute list.

- **name** the name of the attribute
- **text** the text of the entity attribute, could be null
- **vocab** the vocab id of the entity attribute (obtained from the database), could be null
- **measure** the measure of the entity attribute, could be null
- **certainty** the certainty of the entity attribute, could be null

return an entity attribute

createEntityAttribute(String name, String text, String vocab, String measure, String certainty);

Create an entity attribute to be added to the attribute list.

- **name** the name of the attribute
- **text** the text of the entity attribute, could be null
- **vocab** the vocab id of the entity attribute (obtained from the database), could be null
- **measure** the measure of the entity attribute, could be null
- **certainty** the certainty of the entity attribute, could be null
- **isDeleted** set to true to delete the attribute

return an entity attribute

createEntityAttribute(String name, String text, String vocab, String measure, String certainty, boolean isDeleted) ;

Create a relationship attribute to be added to the attribute list.

- **name** the name of the attribute
- **text** the text of the relationship attribute, could be null
- **vocab** the vocab id of the relationship attribute (obtained from the database), could be null
- **certainty** the certainty of the relationship attribute, could be null

return a relationship attribute

createRelationshipAttribute(String name, String text, String vocab, String certainty);

Create a relationship attribute to be added to the attribute list.

- **name** the name of the attribute
- **text** the text of the relationship attribute, could be null
- **vocab** the vocab id of the relationship attribute (obtained from the database), could be null
- **certainty** the certainty of the relationship attribute, could be null
- **isDeleted** set to true to delete the attribute

return a relationship attribute

createRelationshipAttribute(String name, String text, String vocab, String certainty, boolean isDeleted) ;

Populate dropdown field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the dropdown with

populateDropDown(String ref, Collection values);

Populate dropdown field hierarchical vocab terms of the given attribute. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **attributeName** the name of the attribute

populateHierarchicalDropDown(String ref, String attributeName);

Populate radio group field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the radio group with

populateRadioGroup(String ref, Collection values) ;

Populate checkbox group field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the checkbox group with

populateCheckBoxGroup(String ref, Collection values) ;

Populate list with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the list with

populateList(String ref, Collection values);

Used in conjunction with the click event on a list. This returns last selected value in the list.

return the last selected value in the list

String getListItemValue() ;

Populate picture gallery field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the picture gallery with

populatePictureGallery(String ref, Collection values);

Populate hierarchical picture gallery field with the vocabulary in the given attribute name. If the field reference is not found a logic error dialog will

appear.

- **ref** the reference to the field
- **attributeName** the name of the attribute with the vocabulary

populateHierarchicalPictureGallery(String ref, String attributeName);

Populate camera picture gallery field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the camera picture gallery with

populateCameraPictureGallery(String ref, Collection values);

Populate video gallery field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the video gallery with

populateVideoGallery(String ref, Collection values);

Populate audio list field with the given collection of values. If the field reference is not found a logic error dialog will appear.

- **ref** the reference to the field
- **values** the collection of values to populate the audio list with

populateAudioList(String ref, Collection values);

Fetch the archaeological entity record with the specified id.

- **id** the id of the entity

return an entity or null

Object fetchArchEnt(String id);

Fetch the relationship record with the specified id.

- **id** the id of the relationship

return a relationship or null

Object fetchRel(String id);

Fetch the result of a query. This will return only a single row.

- **query** the query to be run against the database

return Collection of String

Object fetchOne(String query);

Fetch the results of a query. This will return a collection of rows.

- **query** the query to be run against the database

return Collection of Collection of String

Collection fetchAll(String query);

Fetch a list of entities. Each row in the collection is a list with the first item being the id of the entity and the second the identifier of the entity.

- **type** the type of arch entity to filter

return Collection of Collection of String

Collection fetchEntityList(String type);

Fetch a list of relationships. Each row in the collection is a list with the first item being the id of the relationship and the second the identifier of the relationship.

- **type** the type of relationship to filter

return Collection of Collection of String

Collection fetchRelationshipList(String type);

Navigation Functionality

Provide functionality to go back as if the user press the hardware back button.

goBack();

GPS Functionality

Set the GPS update interval to determine how often the GPS should update. Default value is 10 seconds.

- **seconds** the length of the interval in seconds

setGPSUpdateInterval(int seconds);

Start using the internal GPS to update the location.

startInternalGPS();

Start using the external GPS to update the location. A dialog will be presented to the user to choose which bluetooth device to be used as external GPS.

startExternalGPS();

Get the GPS position as longitude and latitude from external GPS or internal GPS.

return GPSPosition if starting GPS or null if no GPS started or position found

Object getGPSPosition();

Get the GPS position as longitude and latitude in the project projection from external GPS or internal GPS.

return projected GPSPosition if starting GPS or null if no GPS started or position found

Object getGPSPositionProjected() ;

Get the GPS accuracy from external GPS or internal GPS.

return accuracy of the gps or null

Object getGPSEstimatedAccuracy();

Get the GPS heading from external GPS or internal GPS.

return heading of the gps or null

Object getGPSHeading();

Get the GPS position as longitude and latitude from selected GPS.

- **gps** the type of gps used, one of "internal" or "external"

return GPSPosition if starting GPS or null if no GPS started or position found

Object getGPSPosition(String gps);

Get the GPS accuracy from selected GPS.

- **gps** the type of gps used, one of "internal" or "external"

return accuracy of the selected gps or null

Object getGPSEstimatedAccuracy(String gps);

Get the GPS heading from selected GPS.

- **gps** the type of gps used, one of "internal" or "external"

return heading of the selected gps or null

Object getGPSHeading(String gps);

Start GPS track log with as either time based or distance based.

- **type** either "distance" or "time"
- **value** the value of the tracking, if type is "distance", the value will be in meter, if type is "time", the value will be in seconds
- **callback** code to execute when interval limit is reached

startTrackingGPS(String type, int value, String callback);

Stop the GPS track log.

stopTrackingGPS();

Map Functionality

Bind map click and select events to the map with the following reference.

- **ref** the reference to the map view
- **clickCallback** the callback that is executed when the map view is clicked
- **selectCallback** the callback that is executed when a element is clicked

onMapEvent(String ref, String clickCallback, selectCallback) ;

Used in conjunction with the map click event. This returns the last point clicked on the map.

return the clicked map point

getMapPointClicked() ;

Used in conjunction with the map select event. This returns the last element selected on the map.

return the selected geometry

getMapGeometrySelected() ;

Add a raster map layer to the map view with the following reference and make it the base layer. You can only have a single base layer.

- **ref** the reference to the map view
- **layerName** the layer name for the raster map
- **filename** the filename of the map view, will show error if the file does not exist

showBaseMap(String ref, String layerName, String filename) ;

Add a raster map layer to the map view with the following reference. You can have multiple raster layers.

- **ref** the reference to the map view
- **layerName** the layer name for the raster map
- **filename** the filename of the map view, will show error if the file does not exist

showRasterMap(String ref, String layerName, String filename) ;

Add a shape map layer to the map view with the following reference.

- **ref** the reference to the map view
- **layerName** the layer name for the shape layer
- **filename** the file path of the shape map, will show error if the file does not exist
- **pointStyle** the styling to points appearing in the layer
- **lineStyle** the styling to lines appearing in the layer
- **polygonStyle** the styling to polygons appearing in the layer

showShapeLayer(String ref, String layerName, String filename, GeometryStyle pointStyle, GeometryStyle lineStyle, GeometryStyle polygonStyle) ;

Add a vector map layer to the map view with the following reference.

- **ref** the reference to the map view
- **layerName** the layer name for the shape layer
- **filename** the file path of the shape map, will show error if the file does not exist
- **tableName** the table name to be loaded from the database
- **idColumn** the id column from the table to be loaded from the database
- **labelColumn** the label column from the table to be loaded from the database to be shown as the label
- **pointStyle** the styling to points appearing in the layer
- **lineStyle** the styling to lines appearing in the layer
- **polygonStyle** the styling to polygons appearing in the layer
- **textStyle** the styling to all text labels appearing in the layer

showSpatialLayer(String ref, String layerName, String filename, String tableName, String idColumn, String labelColumn, GeometryStyle pointStyle, GeometryStyle lineStyle, GeometryStyle polygonStyle, GeometryTextStyle textStyle) ;

Add a database map layer to the map view with the following reference.

- **ref** the reference to the map view
- **layerName** the layer name for the shape layer
- **isEntity** a boolean value to determine whether it is entity or relationship
- **queryName** the query name for the executed query
- **querySql** the sql to be executed for the layer
- **pointStyle** the styling to points appearing in the layer
- **lineStyle** the styling to lines appearing in the layer
- **polygonStyle** the styling to polygons appearing in the layer
- **textStyle** the styling to all text labels appearing in the layer

showDatabaseLayer(String ref, String layerName, boolean isEntity, String queryName, String querySql, GeometryStyle pointStyle, GeometryStyle lineStyle, GeometryStyle polygonStyle, GeometryTextStyle textStyle) ;

Add a canvas layer to the map view with the given reference.

- **ref** the reference to the map view
- **layerName** the name of the canvas layer

createCanvasLayer(String ref, String layerName) ;

Set the focus point of the map view using longitude and latitude specified in the projects projection.

- **ref** the reference to the map view
- **longitude** the longitude of the focus point
- **latitude** the latitude of the focus point

setMapFocusPoint(String ref, float longitude, float latitude) ;

Set the focus point of the map view using longitude and latitude specified in the projects projection.

- **ref** the reference to the map view
- **longitude** the longitude of the focus point
- **latitude** the latitude of the focus point

setMapFocusPoint(String ref, double longitude, double latitude) ;

Set the rotation of the map with the given rotation.

- **ref** the reference to the map view
- **rotation** the rotation of the map view in degrees

setMapRotation(String ref, float rotation) ;

Set the zoom level of the map with the given level.

- **ref** the reference to the map view
- **zoom** the zoom level of the map view

setMapZoom(String ref, float zoom) ;

Set the tilt of the map with the given tilt.

- **ref** the reference to the map view
- **tilt** the tilt value default is 90 degrees for 2d view, minimum is 30 degrees

setMapTilt(String ref, float tilt) ;

Remove a layer from the map view with the given reference.

- **ref** the reference to the map view
- **layerId** the layerid to be removed, if not found, a logic error will appear.

removeLayer(String ref, int layerId) ;

Center the map based on the current GPS position if there is GPS position.

- **ref** the reference to the map view

centerOnCurrentPosition(String ref);

Change the visibility of the specified layer in the map view with the given reference.

- **ref** the reference to the map view
- **layerId** the layerid to set visibility, will show logic error dialog if not found
- **visible** the boolean to set whether it is visible or not

setLayerVisible(String ref, int layerId, boolean visible) ;

Change the showAlways attribute of the specified gdal layer in the map view with the given reference. If showAlways set to true, the GDAL layer will always be loaded without doing calculation to determine which tile should be shown at certain zoom level.

- **ref** the reference to the map view
- **layerName** the layerName to set showAlways options, will show logic error dialog if not a gdal layer
- **showAlways** the boolean to set whether it is doing calculation or not

setGdalLayerShowAlways(String ref, String layerName, boolean showAlways);

Draw a point on the map view with the given reference by specifying the point and style.

- **ref** the reference to the map view
- **layerId** the layerId to draw the point to, will show logic error dialog if not found
- **point** the map view point to be drawn
- **style** the style that will be applied to the point

drawPoint(String ref, int layerId, MapPos point, GeometryStyle style) ;

Draw a line on the map view with the given reference by specifying the points and style.

- **ref** the reference to the map view
- **layerId** the layerId to draw the point to, will show logic error dialog if not found
- **points** the map view points to be drawn as a line
- **style** the style that will be applied to the line

drawLine(String ref, int layerId, List points, GeometryStyle style) ;

Draw a polygon on the map view with the given reference by specifying the points and style.

- **ref** the reference to the map view
- **layerId** the layerId to draw the point to, will show logic error dialog if not found
- **points** the map view points to be drawn as polygon
- **style** the style that will be applied to the polygon

drawPolygon(String ref, int layerId, List points, GeometryStyle style) ;

Clear a geometry from the map view with the given reference by specifying the geometry id.

- **ref** the reference to the map view
- **geomId** the id of the geometry to be cleared, will show logic error dialog if not found

clearGeometry(String ref, int geomId) ;

Clear a list of geometries from the map view with the given reference by specifying the list of geometry.

- **ref** the reference to the map view
- **geomList** the list of id of the geometries to be cleared, will show logic error dialog if not found

clearGeometryList(String ref, List geomList) ;

Create a point from longitude and latitude.

- **lon** the longitude of the point in String format
- **lat** the latitude of the point in String format

return MapPos the new point object for the map view

createPoint(String lon, String lat) ;

Create a point from longitude and latitude.

- **lon** the longitude of the point in float format
- **lat** the latitude of the point in float format

return MapPos the new point object for the map view

createPoint(float lon, float lat) ;

Create a point from longitude and latitude.

- **lon** the longitude of the point in double format
- **lat** the latitude of the point in double format

return MapPos the new point object for the map view

createPoint(double lon, double lat) ;

Get all geometries in a specified layer from the map view with the given reference.

- **ref** the reference to the map view
- **layerId** the layerId to get the geometries from, will show logic error dialog if not found

return list of the geometries or null

getGeometryList(String ref, int layerId) ;

Get a geometry from the map view with the given reference by specifying the geometry id.

- **ref** the reference to the map view
- **geomId** the id of the geometry, will show logic error dialog if not found

return list of the geometries or null

getGeometry(String ref, int geomId) ;

Get the layer name of the associated geometry on a canvas layer. @geomId the id of the geometry

- **ref** the reference to the map view

return the layer name of the canvas layer the geometry is on

getGeometryLayerName(String ref, int geomId) ;

Draw geometry on the map view with the given reference by specifying the geometry and style.

- **ref** the reference to the map view
- **layerId** the layerId to draw the geometry to, will show logic error dialog if not found
- **geom** the geometry to be drawn
- **style** the style that will be applied to the geometry

return the id of the created geometry

drawGeometry(String ref, int layerId, Geometry geom, GeometryStyle style) ;

Create a point style with the given minZoom, color, size, and pickSize.

- **minZoom** the minimum level of zoom for the point to show up
- **color** the color of the point
- **size** the size of the point, range from 0.0 - 1.0
- **pickSize** the picking size of the point, used for selecting the point, range from 0.0 - 1.0

return the new point style

createPointStyle(int minZoom, int color, float size, float pickSize) ;

Create a line style with the given minZoom, color, width, pickWidth, and pointStyle.

- **minZoom** the minimum level of zoom for the line to show up
- **color** the color of the line
- **width** the width of the line, range from 0.0 - 1.0
- **pickWidth** the picking width of the line, used for selecting the line, range from 0.0 - 1.0
- **pointStyle** the styling of the point for the line

return the new line style

createLineStyle(int minZoom, int color, float width, float pickWidth, GeometryStyle pointStyle) ;

Create a polygon style with the given minZoom, color, and lineStyle.

- **minZoom** the minimum level of zoom for the polygon to show up
- **color** the color of the polygon
- **lineStyle** the styling of the line for the polygon

return the new polygon style

createPolygonStyle(int minZoom, int color, GeometryStyle lineStyle) ;

Create a text style with the given minZoom, color, size, and font.

- **minZoom** the minimum level of zoom for the text label to show up
- **color** the color of the text label
- **size** the size of the text label
- **font** the font of the text label

return the new text style

createTextStyle(int minZoom, int color, int size, android.graphics.Typeface font) ;

This sets the tilt of the map to be locked at 90 degrees to give a 2D view of the map.

- **ref** the reference to the map view
- **lock** set to true to lock the map view

lockMapView(String ref, boolean lock) ;

Add the specified geometry to the list of highlighted geometry on the map with the given reference.

- **ref** the reference to the map view
- **geomId** the id of the geometry to be highlighted

addGeometryHighlight(String ref, int geomId) ;

Remove the specified geometry from the list of highlighted geometry on the map with the given reference.

- **ref** the reference to the map view
- **geomId** the id of the geometry to be removed from highlighted list

removeGeometryHighlight(String ref, int geomId) ;

Call this method to prepare the geometry in the highlisted list to be transformed to their new position.

- **ref** the reference to the map view

prepareHighlightTransform(String ref) ;

Call this method after calling prepareHighlightTransform once your ready to transform the geometry to their new position.

- **ref** the reference to the map view

doHighlightTransform(String ref) ;

Clear the highlighted geometry list for the map view with the given reference.

- **ref** the reference to the map view

clearGeometryHighlights(String ref) ;

Get the highlighted geometry list for the map view with the given reference.

- **ref** the reference to the map view

getGeometryHighlights(String ref) ;

Add a database layer query to the map with the given reference which will then be used when loading database layers via the layer manager.

- **ref** the reference to the map view
- **name** the name of the query added
- **sql** the query to be executed

addDatabaseLayerQuery(String ref, String name, String sql) ;

Add a track log layer query to the map with the given reference which will then be used when loading track log layers via the layer manager.

- **ref** the reference to the map view
- **name** the name of the query added
- **sql** the query to be executed

addTrackLogLayerQuery(String ref, String name, String sql) ;

Add a query builder to the map view which will then be used by the database selection tool via the tools bar.

- **ref** the reference to the map view
- **name** the name of the query added
- **builder** the query builder to be executed when selected

addSelectQueryBuilder(String ref, String name, QueryBuilder builder) ;

Create a query builder for database selection tool by providing the sql query and adding parameters.

- **sql** the query to be executed when selected

createQueryBuilder(String sql) ;

Add a legacy query builder to the map view which will then be used by the legacy selection tool via the tools bar.

- **ref** the reference to the map view
- **name** the name of the query added
- **dbPath** the path of the database file
- **tableName** the name of the table for the database executed against
- **builder** the query builder to be executed when selected

addLegacySelectQueryBuilder(String ref, String name, String dbPath, String tableName, QueryBuilder builder) ;

Create a legacy query builder for legacy selection tool by providing the sql query and adding parameters.

- **sql** the query to be executed when selected

createLegacyQueryBuilder(String sql) ;

Convert a map position from one projection to another projection.

- **fromSrid** the projection to convert
- **toSrid** the projection to be converted to
- **p** the map position to be converted

convertFromProjToProj(String fromSrid, String toSrid, MapPos p) ;

Enable or disable map tools for the map view.

- **ref** the reference to the map view
- **enabled** true or false to enable or disable map tools

setToolsEnabled(String ref, boolean enabled) ;

Add tool specific events (create or load). The create event is called when the create point, line or polygon tools generate their geometry. The load event is called when the load data tool is used to select geometry. @callback the callback code to execute

- **ref** the reference to the map view
- **type** the event type (create or load)

onToolEvent(String ref, String type, String callback) ;

This is used in conjunction with the tool create event. This returns the last created geometry id.

getMapGeometryCreated() ;

This is used in conjunction with the tool load event. Thi will return the last selected geometry id.

getMapGeometryLoaded() ;

This is used in conjunction with the tool load event. This will return the lsat selected geometry type (either entity or relationship).

getMapGeometryLoadedType() ;

Refresh all the layers of the map.

- **reference** to the map view

refreshMap(String ref) ;

Sync Functionality

Push the full database from the app to the server and execute the callback when finished.

- **callback** the callback that will be executed when the operation finished

pushDatabaseToServer(String callback) ;

Pull the full database from the server to the app and execute the callback when finished.

- **callback** the callback that will be executed when the operation finished

pullDatabaseFromServer(String callback) ;

Enable or disable syncing the database from the app to the server.

- **value** boolean value to set the sync enabled or not

setSyncEnabled(boolean value) ;

Bind to the sync start, success and failure events.

- **startCallback** the callback that will be executed when sync is started
- **successCallback** the callback that will be executed when sync is finish
- **failureCallback** the callback that will be executed when sync is failing

onSyncEvent(String startCallback, String successCallback, String failureCallback) ;

Set the minimum interval for the sync to happen.

- **value** the minimum interval for the sync to happen in seconds

setSyncMinInterval(float value) ;

Set the maximum interval for the sync to happen.

- **value** the maximum interval for the sync to happen in seconds

setSyncMaxInterval(float value) ;

Set the delay interval to add to the current sync interval when a sync failure happens.

- **value** the delay interval for each sync in seconds.

setSyncDelay(float value) ;

Set whether files should also be synced with the database.

- **value** boolean value to set the file sync enabled or not

setFileSyncEnabled(boolean enabled) ;

Static Data Functionality

Get the static data for current project name.

return project name

String getProjectName();

Get the static data for current project projection.

return project projection

String getProjectSrid();

Get the static data for current project id.

return project id

String getProjectId();

Get the static data for current project season.

return project season

String getProjectSeason();

Get the static data for current project description.

return project description

String getProjectDescription();

Get the static data for current permit number.

return permit number

String getPermitNo();

Get the static data for current permit holder.

return permit holder

String getPermitHolder();

Get the static data for current contact and address.

return contact and address

String getContactAndAddress();

Get the static data for current participants.

return participants

String getParticipants();

Get the static data for permit issued by.

return permit issued by

public String getPermitIssuedBy() ;

Get the static data for permit type.

return permit type

public String getPermitType() ;

Get the static data for copyright holder.

return copyright holder

public String getCopyrightHolder() ;

Get the static data for client/sponsor.

return client/sponsor

public String getClientSponsor() ;

Get the static data for land owner.

return land owner

public String getLandOwner() ;

Get the static data for whether containing sensitive data or not.

return true or false

public String hasSensitiveData() ;

File attachment Functionality

Show the file browser and execute the callback once a file is selected.

- **callback** the callback to be executed once the operation finished

showFileBrowser(String callback) ;

This is used in conjunction with `showFileBrowser`. This returns the filename of the last selected file.

return name of last selected file

`getLastSelectedFilename()` ;

This is used in conjunction with `showFileBrowser`. This returns the filepath of the last selected file.

return path of last selected file

`getLastSelectedFilepath()` ;

Copy a file into the projects server or app folders. If the sync is set to true then the file is copied to the app folder. If the sync is set to false then the file is copied to the server folder. Files in the app folder are sync to the server and other apps. Files in the server folder are only synced to the server.

- **filepath** the path of the file to be synced
- **sync** the boolean value to tell whether the file should be copied to the app/server folder

`attachFile(String filePath, boolean sync)` ;

Copy a file into the projects server or app folders. If the sync is set to true then the file is copied to the app folder. If the sync is set to false then the file is copied to the server folder. Files in the app folder are sync to the server and other apps. Files in the server folder are only synced to the server.

- **filepath** the path of the file to be synced
- **sync** the boolean value to tell whether the file should be copied to the app/server folder
- **dir** the directory to be added to the app/server folder, can be null

`attachFile(String filePath, boolean sync, String dir)` ;

Copy a file into the projects server or app folders. If the sync is set to true then the file is copied to the app folder. If the sync is set to false then the file is copied to the server folder. Files in the app folder are sync to the server and other apps. Files in the server folder are only synced to the server.

- **filepath** the path of the file to be synced
- **sync** the boolean value to tell whether the file should be copied to the app/server folder
- **dir** the directory to be added to the app/server folder, can be null
- **callback** code to execute when file is finished copying into the directory

`attachFile(String filePath, boolean sync, String dir, String callback)` ;

List all attached files of an archaeological entity by providing the id of the entity.

- **id** the id of the entity

`viewArchEntAttachedFiles(String id)`;

List all attached files of a relationship by providing the id of the relationship.

- **id** the id of the relationship

`viewRelAttachedFiles(String id)`;

Open the camera and then execute the callback after finish.

- **callback** the callback executed after operation is finish

`openCamera(String callback)`;

Open the video recorder and then execute the callback after finish.

- **callback** the callback executed after operation is finish

openVideo(String callback);

Open the audio recorder and then execute the callback after finish.

- **callback** the callback executed after operation is finish

recordAudio(String callback);

This is used in conjunction with openCamera. This returns the file path of the last picture taken.

return the path of the last taken picture

String getLastPictureFilePath();

This is used in conjunction with openVideo. This returns the file path of the last video taken.

return the path of the last recorded video

String getLastVideoFilePath();

This is used in conjunction with recordVideo. This returns the file path of the last audio taken.

return the path of the last recorded audio

String getLastAudioFilePath();

return returns true if files are currently being attached

isAttachingFiles();

return returns the full path to attached file

getAttachedFilePath(String file) ;

return returns the file path relative to the projects folder of file

stripAttachedFilePath(String file) ;

Add file to checkbox group.

- **ref** reference to the checkbox group
- **file** the filepath of the file to add

addFile(String ref, String file) ;

Add picture to gallery.

- **ref** reference to the gallery
- **file** the filepath of the picture to add

addPicture(String ref, String file) ;

Add file to checkbox group.

- **ref** reference to the checkbox group
- **file** the filepath of the video to add

addVideo(String ref, String file) ;

MISC

Executes a given string of code.

- **code** the string of code to execute

execute(String code);

Helper method to be used with a button click event to attach files to a view

- **ref** the view reference to attach the files to (must be of type file)

attachFileTo(String ref) ;

Helper method to be used with a button click event to attach pictures to a view

- **ref** the view reference to attach the pictures to (must be of type camera)

attachPictureTo(String ref) ;

Helper method to be used with a button click event to attach videos to a view

- **ref** the view reference to attach the videos to (must be of type video)

attachVideoTo(String ref) ;

Helper method to be used with a button click event to attach audios to a view

- **ref** the view reference to attach the videos to (must be of type file)

attachAudioTo(String ref) ;

Helper method to test if a value is null or empty.

- **value** a string value

isNull(String value) ;

Helper method to test if a value is null or empty.

- **value** a list value.

isNull(List value) ;

1.3 Known Issues

FAIMS Mobile Application Known Issues.

The Known Issues page is a list of issues found during regression testing that it was not feasible to fix at the time. This document serves to notify prospective users of the software (provided AS IS without any warranty) about the problems that we have found and are aware of.

Known Issues and Limitations

This page was requested by Brian to detail known limitations of the hardware and software associated with FAIMS. (This includes all known bugs within the FAIMS system as of 29/10/2013)

Tablet Software:

- Android 4.0.3 on the Samsung 10" does not work with the external GPS connectivity (supported Android versions are only those from 4.1.X upwards)
- Samsung note 10.1 - the native video recorder does not record any video when using the default application, requiring the user to download an external video recording application
- Depending on tablet, file selection for attachment may be restricted to the files contained on the SD card only

Web App:

- Deletion of a user does not remove the user from the projects they are currently part of
- Deletion of a single value of a multivalued attribute shows up in the history as all deleted
- If all entities/relationships are deleted, there is no option to view deleted entities
- Archive process can not always handle large files depending on the processing power of the server (for example: on the test server any file >10gb in size failed to archive)
- Syncing devices while the server is locked (ie. when archiving a project, creating a project etc.) will sometimes slow the server down till it no longer loads anything. Server needs to be locked for a long period of time for it to be a problem
- Currently multivalued attributes can not be added or deleted on the server. Multivalued attributes also allow duplicate records to be created if the records are edited to be the same

Using web app on a tablet browser:

- Use the OI File Manager to attach files to the server when using the Android device (download from Android Apps Store). Other file managers may not function properly
- Certain version of the Chrome browser will not download files from the web app correctly from an Android device (download unsuccessful error). Use the Firefox browser to download files instead

Data schema:

- Two attributes with the same name are treated as the same therefore only the first attribute defined will be used

Map UI/performance:

- Large map file sizes/large numbers of geometry etc. causes significant performance issues depending on the speed of the tablet. This can often make the map UI slow and inconsistent for practical use. This is worst when highlighting and/or labels are displayed
- Selection of Experimental Fast Raster Loading in the layer config menu of raster layers may result in non-rendering of raster edges
- Rotating the screen orientation while using the Nexus 4 will sometimes cause the phone to crash. Use only in locked screen mode
- Wrong projection for the area will not allow user to save the created geometry on that area
- Layer manager visual bug: activating re-order in layer manager, dragging the top layer to the middle position and then toggling any other layer's visibility will cause the layer highlighted to remain visible as a 'sticky' overlay on all screens
- Tracklog features are always select-able (point select and polygon select tools) even if no tracklog layer exists on the map
- Tracklog features load into Entity layers, resulting in double-plotting if Tracklog layer is also added to map
- Hidden layers will always be selected if they match the criteria despite not being visible

2 Heurist

Heurist comes with its own documentation (not reproduced here) that can be found at:

<http://heuristscholar.org/help/>.

FAIMS is providing FAIMS-specific feature documentation at: <http://wiki.fedarch.org/> and look for the HEURIST space.

Building FAIMS modules from scratch with Heurist: a how-to guide

By Ian Johnson 19 Feb 2014

- Overall Workflow
- Designing your database (Heurist)
 - Defining entities
- Creating a new Database
 - Using the FAIMS template
 - Equivalences
 - Record type definition
 - Fields
 - Map layers
 - Importing definitions from the FAIMS server
 - Modifying database definitions
 - Record type groups
 - Linking entities
- Creating the FAIMS Module
 - Primary and secondary entity types
 - Additional tab groups
- Creating a Module on the FAIMS server
- Adding users
- Adding map data
- Collecting data, synching tablets
- Transferring data from FAIMS to Heurist
- Manipulating data in Heurist
- Publishing data from Heurist to tDAR

This document aims to step you through the process of creating a new FAIMS Project from scratch (with reference to more detailed instruction pages where appropriate). Please consult [Getting started with FAIMS - an overview](#) for an overview of how all the components fit together.

Overall Workflow

The diagram below illustrates a typical workflow, starting with the creation of a Heurist database (top left) and design of your data model and forms in Heurist, export of the data model and user interface schema and logic to the FAIMS server, where it is turned into a FAIMS database synchronised to all the tablets in the field project. Data is then collected and synched on a daily basis (or more frequently - synching can be continuous if a wireless network is available). After fieldwork is complete (or even on a daily basis), the database can be exported back to Heurist for further recoding, analysis, web publication and archiving in tDAR and other repositories (Open Context is on the schedule for 2014).

The process of designing a database, developing a FAIMS tablet app and then reimporting data to Heurist and exporting it to tDAR, and some of the design decisions behind this process, are illustrated in the following presentation prepared for the 2013 Australian Archaeological Association Conference: [Doing Data Structures Right: from Heurist Entities to FAIMS Forms and tDAR Deposit](#)

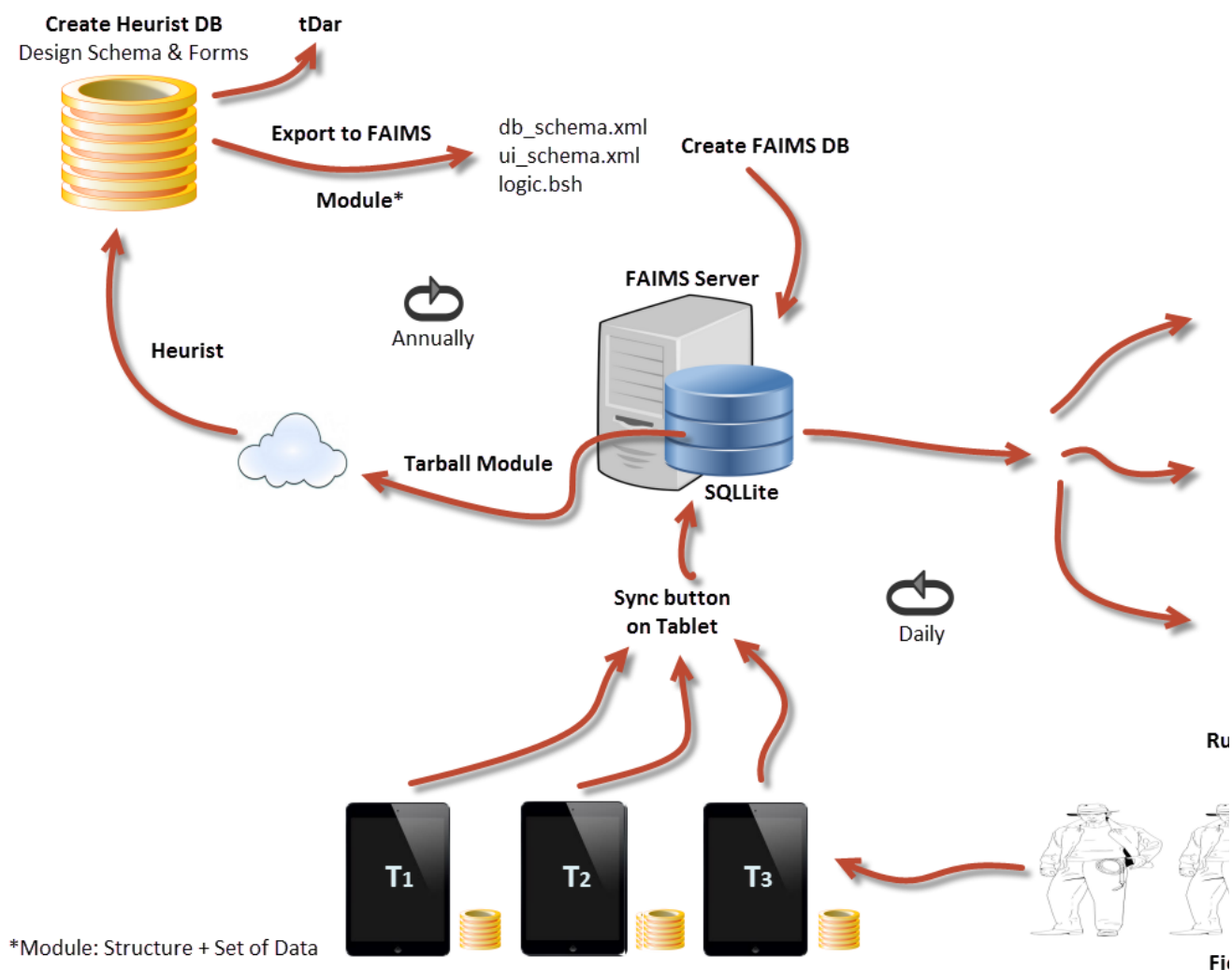


Illustration: Vincent Sheehan

Designing your database (Heurist)

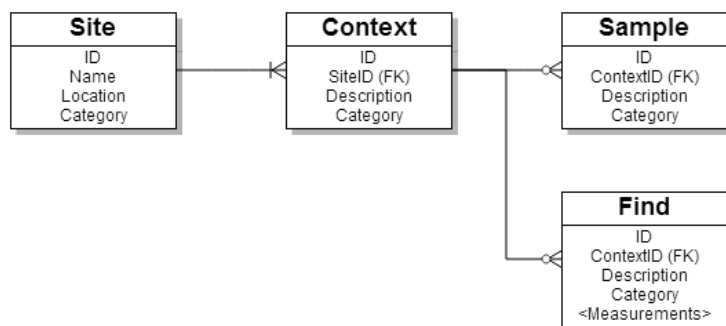
Heurist is a sophisticated generic browser-accessible database, designed by the author and developed at the University of Sydney, with a broad range of functions for database design, manipulation and output. Only a small subset of these capabilities are needed to set up a FAIMS Project / Module. Full instructions on using Heurist are available at <http://heuristscholar.org/help>. Additional information, examples and support are available on the Heurist Network site - <http://HeuristNetwork.org> (under development March 2014).

Defining entities



The first thing you need to do is to define the entities - record types in database parlance - and attributes/fields which you wish to record. Heurist allows the user to create a new database instantly, to import templates for record types, fields and terms from existing databases, and to modify the field definitions through a relatively simple web interface. It then writes out the XML/Beanshell files needed to configure the FAIMS server and tablet app.

Simple hierarchical structure

Let's start with a simple, excavation database:



Archaeological site and survey databases often present this sort of hierarchical structure, with Parents and Children (aka Master and Detail records). In this case,

- *Contexts* (children) belong to *Sites* (parent). *Contexts* cannot belong to more than one *Site*
- *Finds* and *Samples* (children) belong to *Contexts* (parent). *Finds* and *Samples* cannot belong to more than one *Context*
- *Finds* and *Samples* implicitly belong to *Sites* through their connection via *Contexts*
- Every *Site* has at least one *Context* (one to 1-or-many )
- *Contexts* may or may not contain *Samples* and *Finds* (one to 0-or-many )

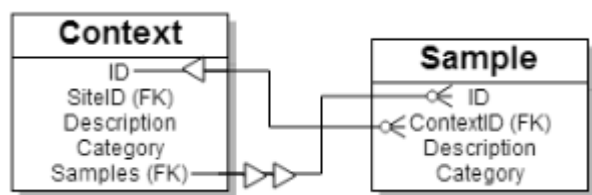
This situation is most easily modelled by including a reference (SiteID) to the *Site* in the *Context* record and a reference (ContextID) to a *Context* in *Samples* and *Finds* from that *Context* (these are known as 'foreign keys' - FK - in database parlance). In Heurist these are simply non-repeatable (single value) Record Pointer fields - 'Site' in the *Contexts* record and 'Context' in the *Samples* and *Finds* record, constrained to *Sites* and *Contexts* respectively. You can find an example of this structure in the FAIMS Community Server under the Simple Excavation tab <to be added>

Important note: in setting up your recording system, you need to think about entry points (Tab Groups in FAIMS parlance). Do you need, for example, to provide direct access to creating a site record? For many excavation projects this will be a rare occurrence, often at the start of the project. You are therefore more likely to start by recording (in this example) a *Context* and relating it back to an existing *Site* (and if the *Site* doesn't exist, you will create it from the first *Context*). You may also enter via a *Find* or a *Sample* and relate it back to an existing *Context* (with the option of creating a parent *Context* and maybe even a parent *Site*). So only *Contexts*, *Samples* and *Finds* should appear as Tab groups.

For workflow convenience you may want to trigger a child record eg. *Sample*, from the parent (*Context* in this case). This can be done by including a repeatable Record Pointer field in *Context* constrained to *Samples*. Clicking on this field will allow the user to create child records of the specified type with a pointer from the parent to the child (*Context* to *Sample* in this example). The child records created from this field should include a Record Pointer field back to the parent record (a Required field, constrained to the parent record type). Note that, at this time (Feb 2014), this field is not populated automatically - you will need to add the parent Record Pointer field and make it Required when defining the database, and the user will have to select the appropriate parent when adding a child record. In a future update (mid 2014) we will automate this process and remove the creation of parent-to-child pointers (see below).

A case of redundancy

Note that this methodology creates undesirable redundancy - a parent can point to a child, but the child can point to a different parent.



Bidirectional Pointers / Foreign Keys (Parent >> Children and Child > Parent)

In a future update (mid 2014) Heurist will allow the creation of child records from a parent record with automatic population of the parent record pointer in the child and without creating child pointers in the parent. The child records will also be creatable independently, with linking back to a record which becomes the parent.

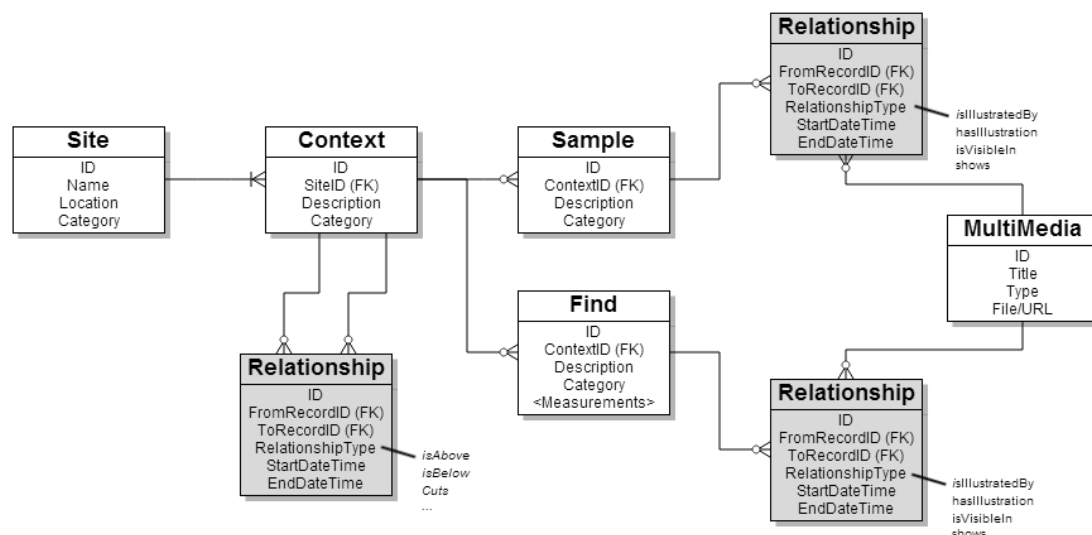
This structure can easily be extended with additional levels of the hierarchy, eg. site areas, buildings, trenches, grid squares, phases etc. following exactly the same pattern. Entities can also belong to multiple parents, although this is rare and generally indicates the need for Relationships (see below).

Adding Relationships

Typically we will want to connect different *Contexts* together with stratigraphic relationships. We may also wish to indicate that particular finds were extracted from a sample bag, that photographs illustrate various Finds, Contexts, Features and so forth.

One way of doing this would be to create pointer fields everywhere that a relationship is required. So artefacts could have a pointer field to all photographs in which they are illustrated. They would then need a pointer field to all artefacts of which they are a part (refitting). But how do we cope with the relationships between Contexts, which may be Above, Below, Cut, ... etc. Should we create 20 or 30 different pointer fields to cover all the alternatives, most of which will be unused? Obviously, the answer is No.

Relationships allow connections to be established between any two types of entity, but also allow the type of connection to be recorded. So the stratigraphic example ends up being a set of relationships (Stratigraphic relationships) where each relationship can have a different relationship type and an entity can have several relationships, with different entities and different types of entities. Photographs, finds, features can all be connected with relationship types such as *illustrates* or *isIllustratedIn*.



When to use: The general rule to follow is to use a Pointer field for simple unequivocal connections such as parent-child/master-detail/whole-part relationships which exist intrinsically between entities, and to use Relationships for record connections which are less standardised ie. have lots of different options (such as stratigraphic relationships or family relationships by birth) or have a time-limited component (such as museum loans or personal relationships by marriage or association) or otherwise require additional information (such as assignments of connections which require interpretation and explanation).

Pointers

Pointer fields in Heurist will be rendered in the FAIMS app as selectable lists of target entities along with a Create button(s) for the appropriate target entity type(s).

Relationship Markers

Relationship markers are a special field type in Heurist which allow relationships to be embedded directly in the data entry form as though they were fields. Each relationship marker can

define the target entity type(s) and the allowable list of relationship types - several relationship markers with different target entity types and different allowable relationship types can co-exist in one entity form. Relationship makers will be rendered as relationships in the FAIMS app, with a Create Relationship button and list of existing relationships.

Adding Master-Detail for analysis

Archaeological entities such as features, samples or artefacts, are generally described through a series of fields or attributes such as type or category, raw material(s), weight, length and other measurements, colour, description, markings and so forth. Some artefacts have components which themselves will have sets of attributes. These will generally need to be picked out as a new entity eg. Artefact Component, and linked as a child/detail record from the parent/master.

This technique can also be used to provide different sets of attributes according to the type of feature or artefact, by providing, for instance, entities such as Wall, Well, Hearth, Artefact, Bone, Manuport, Dating Sample, Soil Sample, each of which has an appropriate set of descriptors. Additional entities can be nested within these for more specialised analysis, eg. Stone Point Attributes or Retouch/Use. We suggest the use of a convention such as «A» after the entity name to indicate that it is a recording of attributes rather than a separate physical entity.

Note: this technique will work all the better once the master-detail changes have been made to Heurist (mid 2014). Examples will be added once we have this developed.

Creating a new Database

To set up a database in Heurist, simply navigate to the URL of your Heurist installation on the Internet (eg. the FAIMS server, a personal NeCTAR Research Cloud server or university server) or on a local server (normally something like 127.0.0.0/heurist), register with Heurist (if a new user) and then create a new database from the menu page. If you are logged into a Heurist database, you can create a new database from Designer

View > Essentials > New Database (switch between views through the USER VIEW / DESIGNER VIEW buttons at top left).

Heurist Designer View (left) and User View (right)

The following powerpoint developed for the 2013 Australian Archaeological Association conference FAIMS workshop illustrates the process of setting up and configuring a Heurist database: [AAA 2013 FAIMS Heurist workshop](#)

Using the FAIMS template

First, select the standard FAIMS template *, if available. If the FAIMS template is not listed, use the standard (default) template. Provide a terse but informative name for the database (eg. USyd_Zagora_Project rather than MyDatabase ...) and click Create Database. Note that a shortened version of your user name is prepopulated in the database name, and it is a good idea to leave this so that the system administrator can identify the databases belonging to each user.

* should be available March 2014, and will be updated as more FAIMS databases get set up and we acquire new template record types

Create New Database

☒ **Standard database**
Gives an uncluttered database with essential record and field types
Recommended for general use

☐ **Extended database**
A database structure with extra record types and fields to support tool such as XSL transforms
The additional structure elements can be imported later from the H3ToolSupport database

New database creation takes 10 - 20 seconds. New databases are created on the current server.
You will become the owner and administrator of the new database.
The database will be created with the prefix "hdb_" (all databases created by this installation of the software will have the same prefix).

Enter a name for the new database:

hdb_

Test

_FAIMS

CREATE DATABASE

The user name prefix is editable, and may be blank, but we suggest using a consistent prefix for personal databases
so that all your personal databases appear together in the list of databases

Creation of a new database takes 10 - 30 seconds. Then login to the new database with the same credentials you used when registering (ie. your normal Heurist user name and password - all databases you create will carry over the same user name and password).

Create New Database








New database 'Test_FAIMS' created successfully

Admin username: johnson
Admin password: <same as account currently logged in to>

You may wish to bookmark the database home page (search page): http://heuristscholar.org/h3-ao/?db=Test_FAIMS.

Go to Administration page, to configure your new database

In Designer View > Essentials > Record types / fields, you will see something like the following pre-defined record (entity) types (the list may differ):

| Basic record types | | | |
|--|---|---------------------------|--|
| Try 'Import structure' in menu on left | | User defined record types | System Internals +/- |
| Common generic record types which will be useful in many databases | | | |
| Page: 1 Show 100 per page | | | |
| Code | Icon | Name | Description |
| 2 |  | Web site / page | A web site URL, typically a specific page (may be the home page of a website or specific pages or documents of interest) |
| 3 |  | Notes | A simple record type for taking notes |
| 4 |  | Organisation | Organisations (companies, universities, granting bodies, museums, libraries etc.) |
| 5 |  | Digital media item | Digital media files - typically image, sound, video - uploaded to the database or external reference |
| 6 |  | Aggregation | A record which describes a static or dynamic collection of records and their filtering and layout, or acts as a root to which other records point, or both |
| 8 |  | Interpretive annotation | Metadata about a date, spatial extent or other interpretation of information |
| 10 |  | Person | A standard record for a person, may be expanded with additional information as required. |

These entity types are those which crop up in many projects. You are free to delete those you do not need, to add new ones and to modify existing ones at any time, without losing existing data - see further discussion below.

Equivalences

| FAIMS | Heurist | |
|------------------|---------------------|--|
| ArchEnt (Entity) | Record type | An identifiable entity. In FAIMS these must be 'objective' physical objects; a site is regarded as a subjective construct. In Heurist, they can be any type of physical or conceptual entity, including relationships. |
| Relationship | Relationship record | A relationship between entities. All Heurist relationships are instantiated as a relationship record in the main entities table which connects two entities with a typed and time-stamped relationship and, optionally, other fields. In FAIMS, relationships exist in a separate relationships table (ArchRel) with similar structure to the ArchEnt table, and may connect entities or act as conceptual containers for entities - sites are therefore conceived of as relationships. |
| Attribute | Field | In Heurist, fields have a tightly defined semantic role as text, numeric, dates, term lists or pointers with explicit definitions eg. of valid terms or target record types. In FAIMS, attributes are generic and acquire semantic meaning through programmed control of the data that can be inserted into them. |
| | Vocabulary | In Heurist, a vocabulary is a flat or hierarchical tree of terms, generally relating to some specific characteristic such as country, raw material, condition, retouch type, colour or texture. A vocabulary can be reused by several fields in several record types. In FAIMS, terms are simply attached to a particular field and have no hierarchical structure. |
| Vocabulary | Term | Specific nominal scale measures identifying a category or characteristic |
| | ConceptID | In Heurist, a Concept ID or Concept Code is a per-type globally unique identifier of the form 10-83 where the first number indicates the Heurist database in which the record type, field type or term was created and the second number is the internal code within that database (which may no longer exist). Global uniqueness is ensured by allocating database identifiers sequentially through the registration of databases with the Heurist master index. Concept IDs can be used to inherit definitions and create compatible report formats, XML output and XSL transforms across databases, and for mapping equivalent concepts between databases. FAIMS has no equivalent concept; all codes are local to a specific database. |

Record type definition

The first step in designing your database is to break your problem down into separate entities. For example, you might have the following entities in a typical project, which will be represented as record types in Heurist:

- Sites
- Trenches
- Grid squares
- Contexts
- Features
- Sample bags
- Individual artefacts

You may create the record types you require in one of two ways:

- **Import record types from another Heurist database:**
Use the Essentials > Import Structure menu entry in Heurist Designer View. See later for a more detailed explanation. The FAIMS Community Server (#10) is a likely source for suitable record types, which will be populated in the course of 2014.
- **Create record types from scratch:**
Use the Essentials > Manage record types/fields menu entry in Heurist Designer View. This function can also be used to adapt record types, fields and terms imported from another database to your specific needs.

Fields

Within each of the record types you should define and set the parameters of the fields/attributes to be recorded. These include:

- Field name - the prompt which will appear left of the field in data entry
- Help text - the text which will appear underneath the field when Help is on, or when the Help button is clicked, or on rollover (on a computer)
- Type of field - text, memo, numeric, date/time, terms list (categories), geographic (point, line, polygon), file (local or remote), record pointer (free or constrained to a specific target record type), relationship marker (see later)
- Requirement - is the field required
- Repeatability - is the field single value or multi-value/repeatable
- Default value - a value which is entered automatically in the field for any new record
- List of terms for term fields (can be specified as a vocabulary, or as individual terms selected from the tree)

When Heurist exports a module to FAIMS, it creates specifications for each record type selected for export:

- Record types selected directly in the export interface are represented as **tab group** in the FAIMS tablet interface.
- Any record types not selected but which are referenced by the selected record types (eg. through constrained record pointers or relationship markers) are also output, and are represented as data entry forms which can be called from the appropriate location within the record types which reference them.

Heurist fields within each record type are translated to fields in the FAIMS tablet app. The field name/prompt is used as the label for the field in the FAIMS app, and the Help text is used to populate the Help button for the field in the app.

Simple fields

- **Headings:** Headings or separators within a record type are used to define tabs within the tab group for that record type/entity. Any fields appearing before the first heading will be placed on a tab labelled *General* (omitted if there are no fields before the first heading). The title of the heading is used as the name of the tab (shortened for display).
- **Text, Memo:** simple text fields which can accept any character data. Text is single line. Memo fields are multi-line text areas.
- **Numeric:** simple numeric fields which can accept any numeric value, integer or decimal. No range or precision constraints are currently implemented.
- **Date/time:** a date field with calendar popup
- **Terms list:**
 - If the field is a single value required field with less than 3 values, the field is represented as a radio button for each of the terms
 - If the field is a single value field with more than 3 values, the field is represented as a dropdown list populated with the set of terms defined for the field
 - If the field is a multivalue (repeatable) field, the set of terms is represented by a set of checkboxes, some or all of which may be checked
 - At this time, images illustrating terms are not supported by Heurist, but this facility will probably be added in 2014 and any such images will be added to the term (vocabulary) definitions in FAIMS

Special fields

- **File (multimedia):** Heurist file fields can contain any sort of file either as a local (uploaded) file or a file accessible through a remote URL. Heurist determines the file type dynamically based on its mime type. FAIMS distinguishes specific file types for specific fields and defines methods of capture, eg. capture of an image or video from the built-in camera. To support this functionality, name File fields as Photo,

Photograph, Image or Picture to obtain a photo capture field in FAIMS, as Video or Movie for a video field, as Audio for an audio field (the field names simply need to contain one of these words). <check for other keywords> If no keyword is provided, FAIMS will provide a simple file selection field.

- **Geographic object:** FAIMS stores a geometry for all ArchEnt records whether or not a geometry field is defined. However the existence of a Geographic Object field in the Heurist record definition will determine whether the form for that entity type contains a Capture Location button <to check> See also the section on maps below for the method of enabling a Map tabgroup and providing background map layers.
- **Record pointer:** Heurist's record pointer fields are powerful method for connecting records together. They are particularly suitable where there is a fixed relationship between two entities. For example, an artefact will always belong to its context of excavation with an implied relationship type of *wasFoundIn* or *isPartOf*. Record pointer fields allow you to constrain the target record types, so that you can avoid the user making an artefact part of an excavator or a photograph (while the artefact could be in the photograph, that is a different relationship along the lines of *isIllustratedBy*). Record pointer fields become browse lists of appropriate ArchEnts in the interface (also allowing the creation of new ArchEnts of the defined type). Note that, for the sake of tablet interface usability, we recommend constraining pointer fields to one and only one target record type - multiple record types will entail manual customisation of the FAIMS schema and logic files. To handle multiple record types we recommend creating a pointer field for each target record type.
- **Relationship marker:** Heurist's relationship marker fields are not really fields at all. They are markers in the data entry form for the creation of relationships. They serve to constrain the type of relationship which can be created (that is, the pulldown list of relationship types) as well as the types of entity with which a relationship can be created. As with pointer fields, we recommend constraining relationship markers to a single target record type, and creating more than one if necessary, to avoid the need for manual customisation of the FAIMS schema and logic files.

Additional characteristics

- At this time, Heurist does not support the addition of Certainty and Annotation values to individual data values. We expect to add this in a future update, and when this is done the certainty and annotation buttons will be enabled when these metadata are enabled within Heurist. In the meantime, both buttons are disabled by default, but the code is output in the schema file so that they can simply be enabled by editing the file and changing False to True on the required fields.
- Default values of a field for new records will be supplied where set in Heurist <check>
- Repeatable (multivalued) fields are supported where available
- Required fields generate logic to require entry of at least one value
- Field widths in Heurist are ignored in FAIMS which does not have a field width concept <check>
- Fields which are used in the construction of record titles in Heurist (set through Manage record /field types, click the Edit icon and set Title Mask) are flagged as *isIdentifier* fields in FAIMS, and this contribute to a constructed human-not-too-unfriendly title of the record.

Important note:

We strongly recommend breaking your recording scheme down into distinct entities wherever possible, and making extensive use of record pointers and relationship markers in Heurist to link these entities together. Lower level entities (eg. individual artefacts or contexts) will generally include a pointer field to the entities which contain them, allowing the parent entities to be chosen from a list of parent entities (and creation of a new one where required).

Apart from the obvious entities - Project, Site, Transect, Survey Unit, Grab Sample, Sample Bag, Individual Find, Trench, Context, Building, Feature, Wall, Pit, Level, Spit etc. - consider creating Master-Detail relationships such as groups of attributes recorded together. For example, where there are numerous similar measurements to be taken on an artefact, consider a detail entity such as *Measurements* with (at least the first three of) the following fields:

- *Parent:* record pointer constrained to artefact; points to the artefact being measured, single, required
- *Measure:* terms list; the measurement being recorded in this record, single, required
- *Value:* numeric; the value of this measurement, single, required
- *Units:* terms list; the units in which the measurement is made, single, required, default 'cms'
- *Notes:* memo text; notes about this specific measurement eg. if it was inaccurate due to breakage or concretion, single, optional

Note: at the current time (Feb 2014) Master-Detail records are not exclusive, that is it is possible to link multiple masters to a detail. This should be avoided through usage protocol, pending implementation of an exclusive relationship.

Map layers

Support for map layers will be developed in 2014.

In order to include map layers on the map tab, you need to load the maps in Heurist as the following record types, available from the FAIMS Community Server database (#10) through Designer View > Essentials > Import Structure:








- Tiled Map Layer (concept code 2-11)
- KML File (concept code 3-1014)
- Other map layer types will be developed later

The export dialogue (see section on exporting a module, below) determines whether the FAIMS app will be configured with a map tab group as the main point of entry to data recording.














Importing definitions from the FAIMS server

You may wish to acquire specific recording systems from the FAIMS community server using Designer View > Essentials > Import Structure, and

selecting the FAIMS Community Server (#10). This is a Heurist database we are populating with additional recording forms to suit different approaches and legislative contexts (Penny Crook). Select the database by clicking on the database icon.

| ID | Browse | Database Name | Description |
|----|---|------------------------|--|
| 1 |  | H3MasterIndex | Heurist Master Index of Registered Heurist Datab |
| 2 |  | H3CoreDefinitions | Core Structure: definitions incorporated into all ne |
| 3 |  | H3ReferenceSet | Reference Database: a curated set of generally |
| 4 |  | H3ToolSupport | Structures required to support annotation, tiled in |
| 8 |  | Arts_eResearch | Arts eResearch task tracking (bugs, issues, feat |
| 9 |  | H3HistorySchemas | History schemas: record types developed across |
| 10 |  | FAIMS_Community_Server | FAIMS: Federated Archaeological Information Ma |

Click on the down arrow icon next to the record types you wish to import (all required fields and terms associated with those record types will also be imported - you can see the list of fields by clicking on the database title). You can sort the list of available record types by name or by group by clicking on the appropriate column heading:

| Import | Group | Record type |
|---|----------------|--------------------------------|
|  | Core functions | Annotation |
|  | Core functions | Heurist Layout |
|  | Core functions | Tiled Image |
|  | Core functions | Transform |
|  | Zagora | Zagora - Archival record |
|  | Zagora | Zagora - Archival record se... |
|  | Zagora | Zagora - Bucket |
|  | Zagora | Zagora - House |
|  | Zagora | Zagora - Plan |
|  | Zagora | Zagora - Sample |
|  | Zagora | Zagora - Sketch plan |
|  | Zagora | Zagora - Unit |
|  | Survey | Stratigraphic Unit2 |

Import successful

Note: If a record type references another record type through a record pointer or relationship marker field, that record type will also be downloaded.

Modifying database definitions

You can create new record types or modify any of the record types in your database - those created at startup, downloaded or created from scratch - to suit your needs, using standard Heurist methods. For a full discussion please visit the Heurist help file at <http://HeuristScholar.org/help> and consult the sections under **Define Structure**. The record addition/modification functions are accessible through **Designer View > Essentials > Manage record / field types**:

Record Types

User defined record types | Common record types | More specialised functions | System internals | Imported | +/-

These are ones you create

Create a new record type

Page: 1 Show 100 per page

DEFINE NEW RECORD TYPE / FIELDS

Filter by name:

| Code | Icon | Name | Description | Edit | Show | Fields | Group |
|------|------|-----------|-------------------------|------|-------------------------------------|--------|--------------|
| 22 | | Role | Role | | <input checked="" type="checkbox"/> | | User defined |
| 24 | | Play | A Play attributed to VM | | <input checked="" type="checkbox"/> | | User defined |
| 25 | | Character | Character | | <input checked="" type="checkbox"/> | | User defined |
| 26 | | Speech | Speech | | <input checked="" type="checkbox"/> | | User defined |
| 27 | | Venue | Venue | | <input checked="" type="checkbox"/> | | User defined |
| 28 | | Location | Location | | <input checked="" type="checkbox"/> | | User defined |

The powerpoint developed for the 2013 Australian Archaeological Association conference FAIMS workshop AAA 2013 FAIMS Heurist workshop is a useful illustration of the process.

Record type groups

It is a very good idea to use record type groups within Heurist to organise your record types. All lists of record types in the Heurist interface eg. addition of a new record, selection of records for output, are organised according to the order of the tabs and record types within them. We suggest no more than 10 record types per tab. Tabs can be reordered by dragging.

Record Types / Fields

Basic record types | Survey | Excavation | FAMS Excavation Module | System Internals | +/-

This is the basic Excavation module supported by the FAIMS project for recording stratigraphic contexts. It can be customised to suit.

Page: 1 Show 100 per page

DEFINE NEW RECORD TYPE / FIELDS

Filter by name:

Show active only

| Code | Icon | Name | Description | Edit | Show | Fields | Group | Status |
|------|------|----------------|---|------|-------------------------------------|--------|------------|-------------------------------------|
| 17 | | Context | The physical result of any single action, whether it leaves a positive or negative record within the stratigraphic sequence. (MoLAS 1994) | | <input checked="" type="checkbox"/> | | FAMS Excav | <input checked="" type="checkbox"/> |
| 18 | | Artefact Group | The total group or assemblage of objects recovered from a single Context. These are sometimes referred to as 'Bulk Finds'. | | <input checked="" type="checkbox"/> | | FAMS Excav | <input checked="" type="checkbox"/> |
| 19 | | Sample | Deposit samples, or samples of building elements, recovered from this Context for the purpose of further analysis, or ongoing reference. | | <input checked="" type="checkbox"/> | | FAMS Excav | <input checked="" type="checkbox"/> |

Page: 1 Show 100 per page

DEFINE NEW RECORD TYPE / FIELDS

Top level tab groups

| Map | YourEntity1 | YourEntity2 | Control |
|-----|-------------|-------------|---------|
| | | | |

The FAIMS interface will normally include a couple of standard tab-groups (**Control** and **Map**) along with a tab group for each primary entity (record type) that you wish to record.

Primary entities are those which you need to be able to add directly and therefore need a tab group of their own (tabs within the tab group for an entity type will be determined by the sections defined by headers inserted in the Heurist record structure). They are selected at time of generation of the FAIMS module out of Heurist.

Secondary entities are those which will be added as part of the entry of a primary entity; for example, features listed within a context. Secondary entities are implied by the use of a pointer field or relationship marker field within a primary entity - they do not need to be individually specified. Make sure you include pointer fields and/or relationship marker fields (with defined target record types) within your primary entities to ensure that

the secondary entities are included (Heurist will also include any entities which point back to any of the selected primary entities and give them their own tab group unless they are already included in all record types to which they point).

Linking entities

A particular feature of Heurist is the ability to link together records without the complexity one is familiar with in relational systems. Relationships are immediately available between ANY record types, without any further work, but these can also be constrained through a constraints menu so that only particular relationship types are allowed between particular pairs of record types, and this also allows constraint of the number of relationships allowed eg. one can only have two parents or a specimen bag can only belong to one context.

Although this feature is available as overarching rules for the database as a whole, we recommend using Heurist's Relationship Marker fields to define the relationships and constraints between entities for FAIMS, as these field types embed connections directly in the records so that they appear contextualised in the data entry form:

Pointer fields

A field which connects to another record in the database. The type of a pointer can be constrained so you can only select a record of a particular type (or types). Typical uses might include specifying the excavator(s) of a context - necessarily Persons - or the previously mentioned sample bag belonging to a context - necessarily a Context.

Relationship marker fields

Normally relationships exist at the level of connecting records as a whole. But how do you prompt users to make a connection? The Relationship Marker field allows a relationship to be embedded as a field directly in the data entry form - it does not actually contain any data, it simply says 'show this type of relationship at this point in the form'. Relationship markers may be constrained to specific record types and a limited set of relationship types appropriate to that point in the form.

When to use a pointer and when a relationship?

The simple rule is, if you simply need to identify a fixed type of relationship, such as an incontrovertible whole-part or a specific function such as Excavator, use a Pointer field. If you want greater richness, such as specifying an open-ended list of roles, eg. for a film Director, Producer, Gaffer, Actor, Cinematographer, etc. and to enrich those roles with temporal limits, annotation and so forth, then use a Relationship Marker field.

Creating the FAIMS Module

A Module is a specific instance of a Project, that is the current state of the database structure, content and user interface.

Having created all the record types, fields and terms you want to use, select **Designer View > FAIMS & HuNI > Create Module (No data)**

Primary and secondary entity types

First, you will be asked to select the primary entity types to be included in your tablet app. These will appear in the **All Entities** tab as a pulldown list at the start of the app (see below). They are your main entry points to adding or browsing data.

Please select ONLY entity (record) types which you want to be represented as top level tab groups.

Any entity which is referenced by a pointer or relationship marker within these entity types (eg. a site within a project, a context within a site, an artefact, sample or feature within a context) is indicated by >> and will be included automatically in the appropriate locations. The Form column shows whether a data entry form will be included in the app.

TabGrp Form Record type >> record types referenced

Basic record types

- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | <input type="checkbox"/> | Aggregation >> Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | Digital media item >> Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | Duplication of Interpretive annotation and some optehr >> Person, Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | Duplication of test3 |
| <input type="checkbox"/> | <input type="checkbox"/> | Interpretive annotation and some optehr >> Person, Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | Organisation >> Digital media item |
| <input type="checkbox"/> | <input type="checkbox"/> | test >> Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | test3 |
| <input type="checkbox"/> | <input type="checkbox"/> | Web site / page |
| <input type="checkbox"/> | <input type="checkbox"/> | Notes >> Person, Organisation, Aggregation |
| <input type="checkbox"/> | <input type="checkbox"/> | Person >> Digital media item |

Bibliographic - common

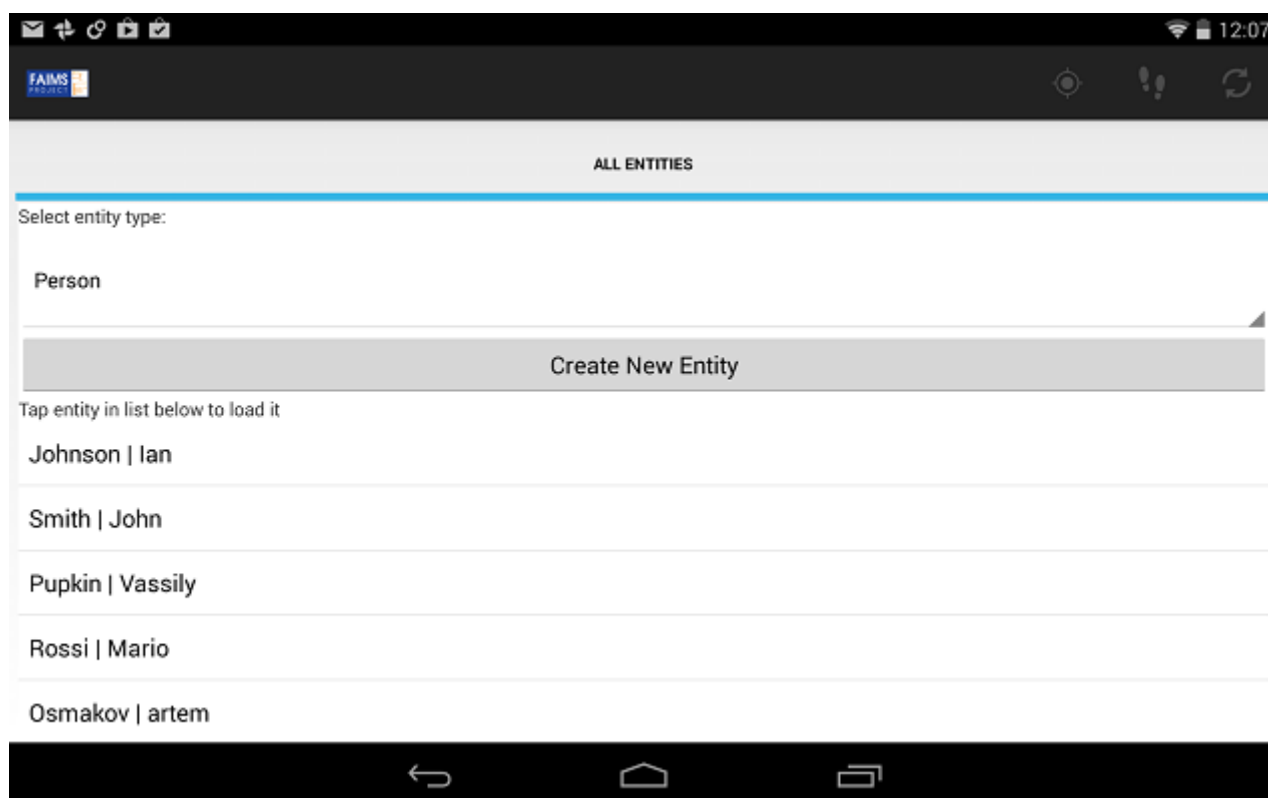
- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | <input type="checkbox"/> | Book >> Person, Organisation, Aggregation, Conference, Publication series, Person |
|--------------------------|--------------------------|---|

Additional data types

- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | <input type="checkbox"/> | Page scan >> Page scan, Page scan, XML Document |
|--------------------------|--------------------------|---|

When you select primary entities you will see that other entities are selected as forms to be created (greyed checkboxes in the second column); these will pop up in appropriate places in the interface - defined by Heurist pointer and relationship marker fields - where these secondary entities are to be entered/linked to the entities which reference them.

The first screen of the tablet interface created by the Heurist export gives you the **All Entities** tab group containing a pulldown list of entity types. Selecting one of these will display existing entities of that type, along with a Create New Entity button which will create a new entity of that type.



Additional tab groups

Next you will be asked to choose the standard GPS controls and the map layers to be included in the tablet app. Either of these tabs can be omitted simply by unchecking the checkbox on the left of the title. The Control tab section determines the GPS and tracklog management functions available to the user. The Map tab section will list any map-renderable datasets found in the Heurist database (geotiffs, tiled images, KML, shapefiles). The Map tab will still be available if there are no suitable map layers in the Heurist database, as it is assumed you may wish to load GIS data separately during preparation of the module on the FAIMS synchronisation server.

Select additional top level tab groups for your app:

☐ **Control tab**

- ☒ Start/stop synching (always on if not checked)
- ☒ Start Internal GPS (on from start if not checked)
- ☐ Connect to External GPS (leave unchecked if no external GPS)
- ☒ Switch tracklog on/off (tracklog unavailable if not checked)

☐ **Map tab**

- ☐ Ordnance survey 1:25K [tiled]
- ☐ Claire's sketch map [tiled]
- ☐ New road alignment [KML]

After making these selections, press **Continue** to write out the FAIMS structure files.

This creates a zip file and provides a link to download it to your laptop/desktop. The file contains the four files required by the FAIMS synchronisation server to create a new FAIMS project. It may also contain supporting files - vocabulary images, map layers and so forth.

- data_schema.xml
- ui_schema.xml
- ui_logic.bsh
- a16n

Note: The FAIMS app draws on the W3C XForms standard, Android native functionality and the Open Data Kit (ODK) data collection toolkit, supported by a custom Java Beanshell script to implement the logic of the interface.

Creating a Module on the FAIMS server

Having obtained the project module files, and edited the UI logic file (if necessary - this will only be required for specific customisation), we can load them into the FAIMS web server to create the project database which can be synched to the tablet app.

Log onto the FAIMS server and click the **Create Module** button:

FAIMS [Users](#) [Modules](#) Ian Johnson ▾

Modules

[Home](#) / [Modules](#)

[Create Module](#) [Upload Module](#)

| No. | Name | Created |
|-----|--|--------------------|
| 1. | Deluxe Excavation Module Alpha | 29/11/2013 04:37PM |
| 2. | Excavation Deluxe Variation 1 | 03/12/2013 06:07PM |
| 3. | Hierarchical Dropdown and Picture Gallery Demo | 29/11/2013 02:27PM |

Browse to the database schema, UI schema, UI logic and A16n files created by Heurist, and click Upload. If all goes well they will be verified and loaded.

Congratulations, you have a FAIMS database.

Adding users

You may now wish to add users before proceeding to the tablet installation step. FAIMS users - defined in the FAIMS Server - are distinct from users defined in the Heurist database.

First select the module then select **Edit Users** to add users to the module:

Historical Architecture

[Home](#) / [Modules](#) / [Historical Architecture](#)

Module Actions

[Edit Module](#) [Edit Vocabulary](#) [Edit Users](#) [Upload Files](#) [Download Module](#)

Users must be added to the FAIMS server first (**Users** at the top of the page) before they can be selected for addition to the module.

Adding map data

Although Heurist may have created a Map tab and copied GIS data files to the ZIP file you downloaded, you will need to transfer and transform these files for use with the FAIMS app.

See [Preparation of Layers outside the Mobile App](#) for instructions on importing map data into the FAIMS server.

Transferring the Module to the tablets

Please consult the FAIMS documentation

See [App install Guide](#) for instructions on installing the app on your tablet(s) and downloading modules.

Collecting data, synching tablets

Please consult the FAIMS documentation

Synchronisation of tablets with the FAIMS server is covered in [Enabling Android Synchronisation](#) - after synchronisation the FAIMS server and all synchronised tablets contain identical FAIMS databases (attached files, however, are not duplicated).

Transferring data from FAIMS to Heurist

At the present time the FAIMS server has relatively little search, visualisation or publishing capability. This may change of course, but it is not a high priority for the FAIMS team given limited resources and much to do on the app development and project implementation.

First, use the Export Tarball function from the FAIMS server to export the complete database + structure to a 'tarball' (a form of Unix zipfile*).

** Brian, turn in your grave!*

Secondly, create a new Heurist database to contain the data (see instructions earlier in this document). This is recommended rather than import into an existing database, as this function is an ingest rather than a synchronisation.

Thirdly, go to Designer View > FAIMS > Import Module (w.data). This function will read the tarball and create any record types, fields and terms required by the Heurist database, then import the data from the tarball. The tarball can be accessed in one of two ways:

- If the Heurist instance is running on a different physical or virtual server from the FAIMS server (upper option below), you need to download the tarball to your desktop or laptop and then upload it to Heurist. This limits the maximum size of tarball to 30MBytes (may be configured by your sysadmin for larger files, up to 300M), enough for a small project but not for a substantial project with significant attached file data.
- If the Heurist instance is running on the same physical or virtual server as the FAIMS server (lower option below), Heurist can simply navigate directly to the file. There is no limit on the size of tarball which can be loaded using this method.

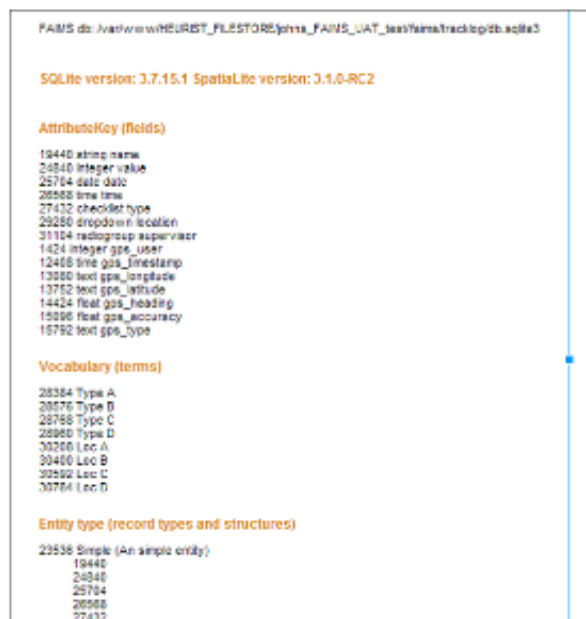
FAIMS sync

☐ Upload FAIMS db or project tar: No file chosen

☒ Or specify server path to FAIMS database:

Check to verify structure (no data import): ☐

As Heurist imports the structure and data it will give a running account of what it is doing:



Once completed, you are ready to use Heurist with the data. In case of failure to complete, please consult the FAIMS team (preferably including a copy of the tarball if not too big to email).

Important Note: We recommend trialling the import of your data from the FAIMS server back to Heurist immediately after setting up the project on the FAIMS server, and again from time to time as the data is collected. This will avoid any nasty surprises and unwanted delays when you actually need to import your data with a post-fieldwork deadline ...

Notes on field values

When importing data from FAIMS to Heurist, there is a potential incompatibility of data model at the level of key-value pairs, which requires careful resolution.

FAIMS allows each key-value pair (AEntVal) to have four separate values simultaneously, and is fairly flexible about what gets recorded owing to a lack of typing and referential integrity (for example, the measure field can contain text and the vocabulary field can contain a code which is not defined as a vocabulary). This can cause some problems in Heurist where the value is unequivocally typed and, for terms and pointers, referentially controlled.

Each FAIMS key-value pair in fact contains four values, and they can all four be filled in one key-value pair:

- vocabulary: code for a term
- measure: a numeric value
- freetext: a text value (can be used as the value itself or as annotation of other values)
- certainty: rating from 0 to 1 (not a value in its own right but an annotation of the other values)

Heurist supports a larger variety of value types, but they cannot co-exist; each key-value pair (recDetail) can have only one value, of the following types:

- term: code for a term
- number: a numeric value
- freetext: a text value (single or multiple lines)
- date: compound date + time, including uncertainty
- pointer: numeric ID of another record
- file: Reference to a locally uploaded or remote file
- geospatial: a geospatial object (FAIMS stores directly in the AEnt record)

In order to resolve this difference we have used the following rules, in priority order, when importing multiple value types into a Heurist value:

1. vocabulary (if present): numeric value is used to set term, other values ignored
2. measure (if present): value is set as numeric, other values ignored
3. freetext (if present): value is set as text
4. certainty value is ignored




































In the next iteration of Heurist (database format version 1.2.0, mid 2014) we plan to add Certainty (decimal 3,2) and Annotation (varchar 250) values to Heurist key-value pairs. We will follow the same priority of allocation, but any additional values in measure or freetext will be

concatenated into our Annotation value, while certainty will be stored separately.

Databases designed in Heurist and then exported to FAIMS will inherently avoid the complication of multiple values in one key-value pair. Where it is wished to associate, for example, a measure with a vocabulary (typically as a means of qualifying a measure with the unit of measurement or of defining multiple related measurements in a single field eg. width at different positions along an object), we prefer to see either repetition of the measure field for each term or, if there are a large number of possible terms, breakout of the data to a separate master-detail entity combining a term list field and a measure field, along with, potentially, other supporting information. The combination of measurement with term in one field is a clever data structure, but one which is likely to create greater complexity of analysis or data exchange at a later stage; it is over normalised.

To find out the content of the ingested database:

- Click the **[Go To User View]** button at top left to return to User View
- Click on the **Database Summary** link at the top of the navigation (lefthand) panel (if there is no Navigation panel, use the Navigation menu or open the panel with the left-arrow icon below the [Go To Designer View] button). This will display the contents of the database and allow navigation to a set of results for each entity type:

| ID | | Record type | | Count |
|----|---|----------------------------|---|-------|
| 1 |  | Record relationship |  | 20 |
| 3 |  | Notes |  | 1 |
| 4 |  | Organisation |  | 8 |
| 5 |  | Digital media item |  | 11 |
| 8 |  | Interpretive annotation |  | 2 |
| 10 |  | Person |  | 23 |
| 23 |  | Book |  | 3 |
| 25 |  | Journal |  | 5 |
| 26 |  | Journal article |  | 12 |
| 28 |  | Journal volume |  | 11 |
| 32 |  | Publication series |  | 2 |
| 33 |  | Publisher |  | 2 |
| 34 |  | Place |  | 27 |
| 38 |  | KML |  | 3 |
| 41 |  | Event |  | 2 |
| 45 |  | Site - field/nat/arch/hist |  | 66 |
| 55 | | Metal Artifact |  | 4 |
| 62 | 62 | Findspot |  | 2 |
| 63 | 63 | Citation |  | 5 |

Manipulating data in Heurist

Once in Heurist, we have access to all Heurist's functions, including advanced searching, data validation and recoding, reporting, mapping and timelines, search and summarisation (simple descriptive stats, notably crosstabulation of 1 to 3 variables), and web publication.

As this is a topic beyond the scope of these instructions, and the capabilities of Heurist continue to evolve, please consult the Heurist help file at <http://HeuristScholar.org/help> for further information.

Publishing data from Heurist to tDAR

We have also developed, as part of the FAIMS project, export direct to the tDAR archaeological repository. FAIMS is working with tDAR to develop an extended capability to run as an Australian archaeological data repository - Heurist will connect to either the US or Australian version (since they share the same Open Source code) and upload data for all selected record types to tables in tDAR, including metadata and associated files. The tDAR API has some weaknesses in terms of uploading information which links 'coding sheets' to fields or columns in their tables, so at the present time some manual editing is required in tDAR after upload to complete the process.

First, you need to obtain a login to the FAIMS tDAR system (or the US system - the software is compatible).

Secondly, login to tDAR via the web interface and create a project, entering the requested project description metadata. Record the project ID allocated by tDAR.

Thirdly, select **Designer View > FAIMS > Export to tDAR repository** and enter the project ID, host address and login information that you have been given, and choose the record types that you wish to export to the repository (your Heurist database will include additional record types which are either for internal usage or have been downloaded but not used, and you want only to upload those record types which are relevant to the repository):


The screenshot shows the 'Export to FAIMS tDAR Repository' form in the 'DESIGNER VIEW' of the 'TDAR_ExportTest' application. The form includes the following fields and options:

- FAIMS project ID:** A text input field containing '7019'.
- Host:** A text input field containing '115.146.85.232:8080'.
- Username:** A text input field containing 'johnson'.
- Password:** A password input field with masked characters.
- Record types to include in export:** A dropdown menu labeled 'SELECT RECORD TYPES'.
- Below the dropdown:** The text 'HARAS Artefact, Site, Collection unit, Specimen Catalog'.
- Create only. Do not upload/register:** A checkbox that is checked.
- Buttons:** A 'START TDAR EXPORT' button at the bottom.

The upload will take some time. Once finished you should be able to find your project by searching for recent projects and view the project details by clicking on the project:

The screenshot shows the FAIMS Project web interface. The top navigation bar includes links for 'About', 'Using FAIMS', and 'Why FAIMS?', along with 'BROWSE' and 'UPLOAD' buttons and a search bar. The main content area displays the details for a project titled 'Heurist UAT SAR' (FAIMS ID: 7010). The project summary states 'Heurist UAT prep by SAR'. The 'Cite this Record' section provides the citation: 'Heurist UAT SAR. (FAIMS ID: 7010)'. A section titled 'There are 8 Resources within this Project' lists resources categorized by type: Coding Sheets (3), Datasets (3), Documents (1), and Images (1). The 'Coding Sheets' section lists three items: 'Coding sheet for Heurist field: Line (2013)', 'Coding sheet for Heurist field: State (2013)', and 'Heurist relation types coding sheet (2013)'. The 'Documents' section lists one item: '9FU118 (2013)'. On the right side, a 'Basic Information' box displays the 'FAIMS ID' as '7010'.

You can then navigate to individual tables corresponding with record types/entities in the Heurist/FAIMS databases:


[Test](#)

[About](#)
[Using FAIMS](#)
[Why FAIMS?](#)
[BROWSE](#)
[UPLOAD](#)

[Search](#)
[Browse](#)
[Explore](#)
[SIGN UP](#)
[LOG IN](#)

Site

Part of the [Heurist UAT SAR](#) project

Year: 2013

Summary

Heurist db TDAR_ExportTest. Rectype#23. A site in the MARTA project

Data Set Structure

☐ Measurement Column
 ☐ Count Column
 ☐ Coded Column
 ☐ Mapping Column
 ☐ Integration Column (has Ontology)

Table Information: 23

| Column Name | Data Type | Type | Category | Coding Sheet | Ontology |
|-----------------|-----------|---------------|---------------|--------------|----------|
| Heurist ID | BIGINT | Uncoded Value | uncategorized | none | none |
| Site identifier | VARCHAR | Uncoded Value | uncategorized | none | none |
| Locality name | VARCHAR | Uncoded Value | uncategorized | none | none |
| Thumbnail image | VARCHAR | Uncoded Value | uncategorized | none | none |
| Digital media > | VARCHAR | Uncoded Value | uncategorized | none | none |
| Short summary | VARCHAR | Uncoded Value | uncategorized | none | none |

Downloads

1

23.csv (1.80kb)

Basic Information

FAIMS ID
7016

Note: Owing to limitations in the tDAR API at this time, it is not possible to automate the creation of the coding sheets for categorised fields (that is, term list fields in Heurist, vocabulary attributes in FAIMS) in the tDAR repository. It will therefore be necessary to edit the coding sheets generated for the project and enter the data through the tDAR web interface. If the tDAR API is extended, the Heurist export functionality will be extended to automate the process.

3 FAIMS Repository

tDAR comes with its own documentation (not reproduced here) that can be found at: <http://dev.tdar.org>. No FAIMS Specific documentation exists at the time of the issue of this Software Development Collection.

The source code for the FAIMS repository lives as a branch within the tDAR source code BitBucket repository: <https://bitbucket.org/tdar/tdar.src>

The supporting scripts (puppet, kettle etc) live in their own sub-directories of the tDAR support code BitBucket repository: <https://bitbucket.org/tdar/tdar.support>

3.1 Developer Instructions

The FAIMS Repository is a scaled up version of the AHAD repository that used to be hosted at La Trobe University, and that was powered by the tDAR project. In effect, an existing body of working software has had some new features added to it. Thus the user documentation is effectively the tDAR user documentation. There are:

- [*Help and tutorials*](#)
- [*Policies and procedures*](#) (The FAIMS ones are different, but these describe quite well what is baked into the code)
- [*Guides to contribution*](#)
- [*The tDAR wiki*](#)

The instructions on how to check out and build the source code are all found on the tDAR developer wiki: <https://dev.tdar.org/confluence/display/DEV/Developer+Documentation>. In addition, design documentation, etc.. can be found there.

The source code is kept in a BitBucket repository: <https://bitbucket.org/tdar/tdar.src>

Note that the currently deployed FAIMS repository code is the [*stable-faims*](#) branch.

There is a separate BitBucket repository for supporting scripts: <https://bitbucket.org/tdar/tdar.support>

This second repository is also of interest to a FAIMS repository developer as it is where one will find the scripts for:

- [*Kettle*](#)
- [*Puppet*](#)

The detailed instructions on how to spin a repository instance up on the NeCTAR cloud are kept as several markdown files in the support repository along side the puppet scripts. Change the scripts: update the markdown!

They are:

- [*How to deploy from your development machine*](#)
- [*An overview of the recovery process*](#)
- [*How to get the data from the object store and onto a machine*](#)

- *How to get the data out of an encrypted file and onto the data block storage device*
- *How to recover from the loss of a VM and to get back to a running instance from the block storage itself*

3.2 Mobile Import

Page 1 of 6

Author: Martin Paulo

The Kettle tarball import

Introduction

The FAIMS tarball import is run by scripts outside of the repository. The reason for choosing this approach is several fold:

- It was originally planned that the mobile import would be done by the tDAR external web API
- The contents of the tarball were likely to change during the mobile development, whereas the tDAR repository has its architecture already baked in, so any internal changes would be very slow to happen. External scripts decouple the two and allow FAIMS to react rapidly without having to affect tDAR
- tDAR is an established code base in use with an existing development team. External scripts mean that FAIMS won't have to negotiate changes with the tDAR team.
- tDAR is being architected to pass long running jobs to a message queue. When the message queue is implemented, FAIMS could plug the working scripts directly into it.
- External scripts allow people to alter the format of the tarball, and the import scripts without having to understand the inner workings of tDAR.

The scripts are run by Kettle: an open source ETL tool. Kettle was chosen as:

- It provides a library of adaptors and interfaces that mean FAIMS can make rapid changes to the import without having to code everything from first principles.
- It allows users to drop down to the command line, so that if there are portions Kettle can't do, they can be scripted.
- The library of adaptors mean that when the tDAR message queue is implemented, FAIMS can plug Kettle into it. Kettle can even be bundled directly into tDAR, if needed.
- It is an "Extract, Transform & Load tool". Which describes pretty much what FAIMS is doing with the tarball.
- It is cross platform: so that people with totally different development environments can use it.
- It accesses databases via JDBC: and allows additional JDBC drivers to be used. Hence if FAIMS can find cross platform JDBC drivers appropriate to the Spatialite version being used, some of the complexity of deployment will be reduced.
- It is open source, reasonably well documented, and has a commercial company behind it.
- It supports the creation of Access databases. These allow the relationships between the tables to be expressed – and understood by tDAR.

Kettle can be found here: <http://kettle.pentaho.com/>

The version used for these scripts is 4.4.0 stable.

The work flow is as follows:

- There is an archive page in tDAR that allows a user to upload a tarball. The user is offered the option of unpacking the contents of the tarball into the repository by means of a checkbox. If selected the tarball will at a slightly later time be unpacked and its components imported into the parent project of the archive by means of the external scripts. If the user doesn't select this checkbox, it will still be available when they edit the archive metadata in



3.3 Backups

Backing up the FAIMS repository

Introduction

The FAIMS repository has several artefacts that need to be backed up: a set of personal file stores, a shared file store, and a set of databases.

The databases hold both data and the state of the system, the shared file store is the actual repository for any uploaded artefact.

Given the number of artefacts that need to be backed up, and the environment in which the repository is run, the backup strategy chosen is a layered one.

The Environment

To understand the strategy, the environment in which the repository runs has to be understood.

The repository runs on a virtual machine hosted in the cloud. That virtual machine has transient storage attached to it. Transient storage is simply storage that only endures while the machine is alive – if the machine dies, everything on the storage is lost.

This transient storage is relatively fast to access, but anything written to it will be lost when the machine is taken down. The transient storage is also fixed in size, and that size will not meet the long term requirements of the repository file store.

The cloud environment also offers what is effectively network attached storage: storage that is durable, backed up, and extensible on request. This storage is termed 'block storage'. Because it is accessed via a network it is distinctly slower to access than the transient storage. However, applications do access it as though it was a conventional disk drive.

In addition to the block storage the cloud also offers what is termed an 'object store'. The object store is a fault tolerant storage device that keeps multiple copies of any file written to it (3). The object store will try to monitor the uploaded archive and repair any of the copies that may become corrupted. Applications can only access it through a special interface. It is publicly accessible if need be.

The strategy

It was decided that all of the databases, which get lots of small read and writes, should be located on the transient storage, and that the file stores, which get far fewer, but larger reads and writes, should be placed on block storage.

Once a day a script runs that makes backup copies of all of the databases.

- It makes a daily backup that is rotated on week boundaries (i.e., only the last 7 daily backups are kept).



- Every Saturday, a weekly backup is created. These weekly backups are rotated on a 5 week cycle.
- On the 1st of every month a monthly backup is created. These monthly backups are not rotated: they simply accumulate over the lifetime of the server.

All of these backups are written to the same block storage that is used for the file stores. The shared file store can be considered to be additive, in that artefacts are never removed from it.

At a slightly later time another script runs. This one attaches another block storage device, then copies everything on the primary block storage device to the backup block storage device, in the form of an encrypted and compressed archive. These encrypted archives are currently rotated on a 7 day boundary.

In addition, this encrypted archive is copied to the object store, from where a copy of the encrypted archive can be downloaded for offsite backup. The intent is that these offsite backups will be done on a weekly/monthly basis once the repository goes live.

The number of backups made to the backup block storage, the object store, and offsite are all parameterised, and thus can be adjusted appropriately as the repository grows.

Each backup contains a greater history of the changing database. So we have backups within backups.

This backup strategy means that if the repository is lost, no more than 24 hours worth of data and state should be lost. This is the trade off between speed and safety that has been made.

3.4 Command Line Import Tool

The tDAR Command Line Tool

Introduction

The command line tool is a tool that allows data to be imported into the repository via the command line, thus allowing the import to be automated.

The tool works by making restful web calls to a running instance of the repository. It expects to be configured via a set of options passed in via the command line.

It expects to be given, as part of these options, one or more file or directory names. This list of files and directories will be iterated across (and recursively down, in the case of subdirectories), and any file found with the '.xml' extension will be deemed to be a record, and any other file found will be deemed to be an attachment.

There are some interesting rules in play here:

- Any directory with more than one record found, will only have the records processed, and any attachment in the directory will be ignored.
- Any directory with only one record found, will have the record and all attachments in the directory processed.
- Any directory with just attachments will have the attachments ignored.

The xml files are expected to conform to the schema supported by the server being targeted. This schema can be found at the URL path [/schema/current](#) of the server being targeted.

In the interests of safety, the command line tool will assume, unless instructed otherwise, that it is being run against a default test instance of the repository, found at <http://alpha.tdar.org>.

Because the command line tool is making use of restful http calls there is no need for it to be run on the server that is being targeted.

For the more technically minded the tool is provided by the Java `org.tdar.utils.CommandLineAPITool.java` file found in the tDAR code base. A reading of its code will show developers how to create equivalent tools in their language of choice should they wish to write to the restful http interface directly.

The command line options

These are the current command line options, without some of which the tool will not run.

- `help` – list the command line options described below.
- `http` – run using the http protocol (the default is https).
- `log` – log messages at the Info level, and copy them to the console.
- `username <argument>` – The name of the user account that command line tool is to be run with. This must be an already registered user within the running instance of tDAR/FAIMS being targetted. Required, unless running against the default alpha system.
- `password <argument>` – The password associated with the user name. Required, unless running against the default alpha system.

- `Host <argument>` – The hostname of the repository that the command line import is targeting. If it is not visible by DNS then this must be an IP address. In the interests of safety this defaults to `alpha.tdar.org`.
- `file <argument>` – The name of a file or directory that is to be ingested. Can occur 1 to many times.
- `config <argument>` – An optional file that allows one to specify the command line arguments that actually take arguments in a file, rather than on the command line. Anything entered on the command line will override the values read from the file.
- `projectId <argument>` – The id of the project in the repository to which the uploaded artefacts are to be added. This project id will replace any project referenced in the xml resource files.
- `accountid <argument>` – The id of the billing account associated with the user that is to be used for this transaction.
- `sleep <argument>` – The time to wait, in milliseconds, between the import of each record. This value currently defaults to zero.
- `logFile <argument>` – The file which tracks the import of each record. Essentially just a list of the records that were successfully processed. This file will be appended to by the application on each run. Don't be confused: there is a separate log file in which the application logs its activities.
- `fileAccessRestriction <choice>` – The access restriction that is to be applied to all the attachment files ingested during this run of the tool. The choice can currently be one of `[PUBLIC | EMBARGOED | CONFIDENTIAL]`. This value currently defaults to `PUBLIC`

Return codes

The command line tool has the following return codes:

- `-1` : there was a problem encountered with the parsing of the arguments
- `0` : the run proceeded without issue
- any number `> 0` : the number of files that the tool was not able to import successfully.

Running the tool

There are currently two ways to run the command line tool:

- From within the deployed Java web application directory
- From within the project source code

From within the deployed Java web application directory:

```
java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool
```

From within the project source code:

```
mvn exec:java -P apitool
```

If no arguments are provided the application simply lists the help and exits:

```
martin:classes admin$ java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool
args are: []
usage: TDAR/FAIMS cli api tool
```



```

-accountid <accountid>      TDAR/FAIMS the users
                             billing account id to
                             use
-config <config>            optional configuration
                             file
-file <file>                the file(s) or
                             directories to process
-fileAccessRestriction <fileAccessRestriction> the access restriction
                             to be applied - one of
                             [PUBLIC | EMBARGOED |
                             CONFIDENTIAL]
-help                       print this message
-host <host>               override default
                             hostname of
                             alpha.tdar.org
-logFile <logFile>         TDAR/FAIMS logFile
-password <password>      TDAR/FAIMS password
-projectid <projectid>    TDAR/FAIMS project id.
                             to associate w/ resource
-sleep <sleep>            TDAR/FAIMS timeToSleep
-username <username>      TDAR/FAIMS username
-----
Visit https://dev.tdar.org/confluence/display/TDAR/CommandLine+API+Tool for documentation on how
to use the TDAR/FAIMS commandline API Tool
-----

```

The config file

The config file is simply a standard Java properties file. Any of the command line arguments that actually take an argument can be placed in this file. E.g.:

```

username=benben
password=ben
file=/Users/admin/Documents/CLTest/exampleOne/
host=localhost:8080

```

There is one caveat to be aware of: the file property can have multiple values, but they must be comma separated. E.g.:

```

file=/Users/admin/Documents/CLTest/exampleOne/,/Users/admin/Documents/CLTest/exampleTwo/

```

Logging

The tool makes use of the log4j library to record what it is doing.

If launched with the -log command line option **any existing configuration will be ignored**, the logger will be set to the INFO level, and all output will be directed to the console.

More on the log4j library and configuring it can be found here:

<http://logging.apache.org/log4j/1.2/manual.html>

Restful web calls

If restful web calls are made instead of using the command line tool, then the repository returns an XML fragment of the form:

```

<apiResult>
  <status>created</status>
  <recordId>989</recordId>
  <message>created:989</message>

```

```
</apiResult>
```

In this fragment all of the elements are optional.

The record id element is the database id of the record affected.

The message is simply extra text: it's useful in debugging problems.

The status will be one of:

- success
- created
- gone
- updated
- notfound
- unauthorized
- badrequest
- unknownerror
- notallowed

Examples

Example 1: Create a project via the command line

Title & description are mandatory fields.

project.xml (contained in the directory /Users/admin/Documents/CLTest/exampleOne/):

```
<?xml version="1.0" encoding="utf-8"?>
<tdar:project xmlns:tdar="http://www.tdar.org/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://core.tdar.org/schema/current">
  <tdar:description>This is a test project imported from the command line.</tdar:description>
  <tdar:title>Command line test project</tdar:title>
</tdar:project>
```

Command line:

```
java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool -http -username benben
-password ben -file /Users/admin/Documents/CLTest/exampleOne/ -host localhost:8080
```

Example 2: Create a project using a config file

project.xml (contained in the directory /Users/admin/Documents/CLTest/exampleOne/):

```
<?xml version="1.0" encoding="utf-8"?>
<tdar:project xmlns:tdar="http://www.tdar.org/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://core.tdar.org/schema/current">
  <tdar:description>This is a test project imported from the command line.</tdar:description>
  <tdar:title>Command line test project</tdar:title>
</tdar:project>
```

input.properties (contained in the same directory as the command line tool):

```
username=benben
password=ben
file=/Users/admin/Documents/CLTest/exampleOne/
host=localhost:8080
```

Command line:

```
java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool -http -config input.properties
```

Example 3: Overriding a value in a config file

project.xml (contained in the directory /Users/admin/Documents/CLTest/exampleOne/):

```
<?xml version="1.0" encoding="utf-8"?>
<tdar:project xmlns:tdar="http://www.tdar.org/namespace"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://core.tdar.org/schema/current">
  <tdar:description>This is a test project imported from the command line.</tdar:description>
  <tdar:title>Command line test project</tdar:title>
</tdar:project>
```

input.properties (contained in the same directory as the command line tool):

```
username=benben
password=ben
file=/Users/admin/Documents/CLTest/exampleOne/
host=localhost:8080
```

Command line:

```
java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool -http -config input.properties
-host=115.146.92.150:8080
```

In this case the host is an entirely different machine.

Example 4: Uploading multiple files from the command line

input.properties (contained in the same directory as the command line tool):

```
username=benben
password=ben
file=/Users/admin/Documents/CLTest/exampleFour/One,/Users/admin/Documents/CLTest/exampleFour/Two
host=localhost:8080
projectid=6838
```

Note the multiple file entries separated by a comma.

Were this to be run from the command line, it would be:

```
java -cp ../ROOT/WEB-INF/lib/*:. org.tdar.utils.CommandLineAPITool -http -username benben
-password ben -file /Users/admin/Documents/CLTest/exampleFour/Two -file
/Users/admin/Documents/CLTest/exampleFour/One -host localhost:8080
```

In the first directory there is an image “melbourne.jpg”, and the file melbourne.xml.

In the second directory there is an image “sorrento.jpg”, and the file sorrento.xml.

The contents of melbourne.xml are:

```
<?xml version="1.0" encoding="utf-8"?>
<tdar:image xmlns:tdar="http://www.tdar.org/namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://localhost:8180/schema/current schema.xsd">
  <tdar:description>A transport nightmare</tdar:description>
  <tdar:resourceType>IMAGE</tdar:resourceType>
  <tdar:title>Melbourne</tdar:title>
  <tdar:date>2012</tdar:date>
  <tdar:dateNormalized>2012</tdar:dateNormalized>
  <tdar:externalReference>false</tdar:externalReference>
  <tdar:inheritingCollectionInformation>false</tdar:inheritingCollectionInformation>
  <tdar:inheritingCulturalInformation>false</tdar:inheritingCulturalInformation>
  <tdar:inheritingIdentifierInformation>false</tdar:inheritingIdentifierInformation>
  <tdar:inheritingInvestigationInformation>false</tdar:inheritingInvestigationInformation>
  <tdar:inheritingMaterialInformation>false</tdar:inheritingMaterialInformation>
  <tdar:inheritingNoteInformation>false</tdar:inheritingNoteInformation>
  <tdar:inheritingOtherInformation>false</tdar:inheritingOtherInformation>
  <tdar:inheritingSiteInformation>false</tdar:inheritingSiteInformation>
```

```

<tdar:inheritingSpatialInformation>false</tdar:inheritingSpatialInformation>
<tdar:inheritingTemporalInformation>false</tdar:inheritingTemporalInformation>
<tdar:relatedDatasetData/>
</tdar:image>

```

The contents of the file sorrento.xl are:

```

<?xml version="1.0" encoding="utf-8"?>
<tdar:image xmlns:tdar="http://www.tdar.org/namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://localhost:8180/schema/current schema.xsd">
  <tdar:description>A dream to visit</tdar:description>
  <tdar:resourceType>IMAGE</tdar:resourceType>
  <tdar:title>Sorrento</tdar:title>
  <tdar:date>2012</tdar:date>
  <tdar:dateNormalized>2012</tdar:dateNormalized>
  <tdar:externalReference>false</tdar:externalReference>
  <tdar:inheritingCollectionInformation>false</tdar:inheritingCollectionInformation>
  <tdar:inheritingCulturalInformation>false</tdar:inheritingCulturalInformation>
  <tdar:inheritingIdentifierInformation>false</tdar:inheritingIdentifierInformation>
  <tdar:inheritingInvestigationInformation>false</tdar:inheritingInvestigationInformation>
  <tdar:inheritingMaterialInformation>false</tdar:inheritingMaterialInformation>
  <tdar:inheritingNoteInformation>false</tdar:inheritingNoteInformation>
  <tdar:inheritingOtherInformation>false</tdar:inheritingOtherInformation>
  <tdar:inheritingSiteInformation>false</tdar:inheritingSiteInformation>
  <tdar:inheritingSpatialInformation>false</tdar:inheritingSpatialInformation>
  <tdar:inheritingTemporalInformation>false</tdar:inheritingTemporalInformation>
  <tdar:relatedDatasetData/>
</tdar:image>

```

3.5 Storage



FAIMS Repository Storage Land Scape ('Use Case')

Revision Chart

| Version | Primary Author(s) | Description of Version | Date Completed |
|-----------------------------|-------------------|--|--------------------------------|
| 0.1 - Draft Internal | Martin Paulo | Initial draft for internal review | 7 th January, 2013 |
| 0.2 – Draft External | Martin Paulo | Draft for external review | 8 th January, 2013 |
| 0.3 – Draft External | Martin Paulo | Draft for external review, adds Lucene indexes | 9 th January, 2013 |
| 1.0 – Final | Martin Paulo | For release | 15 th January, 2013 |

Introduction

The FAIMS repository will be build by scaling up the existing AHAD repository hosted at La Trobe University. The AHAD repository is in turn powered by tDAR, an open source digital archive and repository.

In effect, an existing large body of working Java software will have new features added to it. By this means the repository development is intended to leverage the cooperative nature of open source software, giving a low risk approach to delivering a repository that will allow Australian archaeological researchers to deposit and access data.

tDAR, A Very High Level View

tDAR is first and foremost a repository. In that role, users upload digital artifacts into the repository. These artifacts are stored in their original format, and also, if need be according to their type, copied into an associated preservation format.

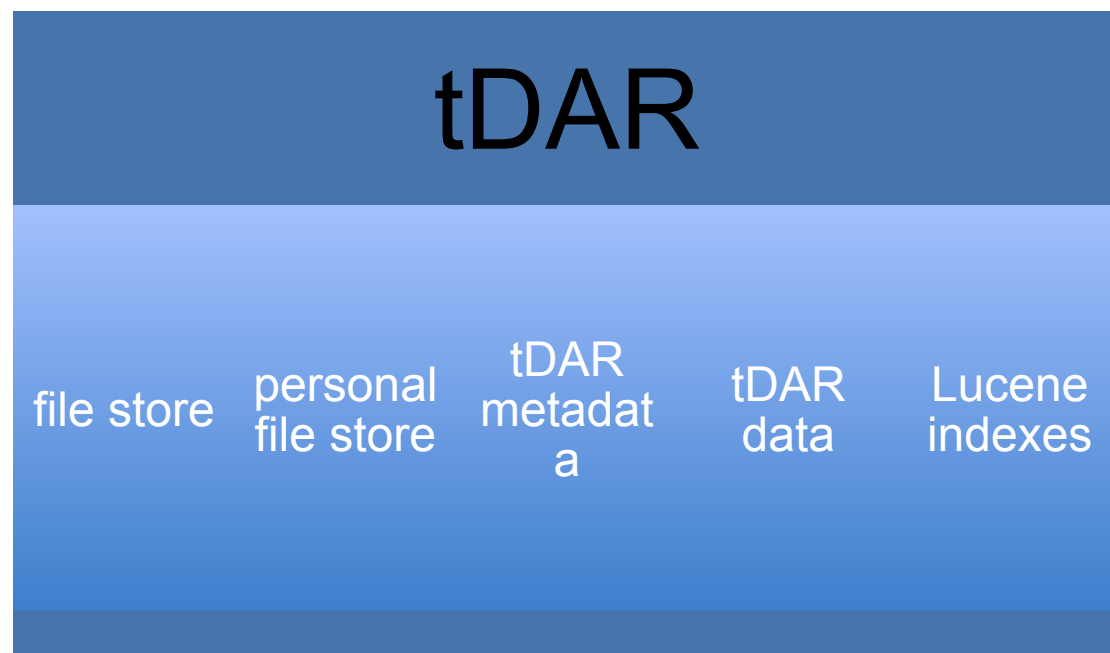
In the process of uploading digital artifacts, metadata is entered by the user and associated with the digital artifacts. This metadata facilitates the organization of, the discovery of, and the definition of relationships between the uploaded artifacts.

Certain types of digital artifacts, containing tabular data, such as Excel spreadsheets and MS-Access databases, also have their contents extracted and copied into a 'read only' relational database. tDAR provides toolsets that allow users to easily share this extracted data with other users, and also to integrate the extracted data to form new data sets offering greater insight and knowledge.

tDAR makes use of Lucene to index the digital artifacts on ingest.

tDAR Storage, The Main Parts

They say a picture is worth a thousand words. So:



File Store

The file store is the repository where the uploaded digital artifacts are stored. It is implemented by means of a [PairTree](#): a file system hierarchy.

Personal File Store

The personal file store acts as user specific digital artifact holding pen: one in which files uploaded asynchronously are assembled before being worked with, or where digital artifacts generated by the user in working with datasets are stored. It is implemented by means of [BagIt](#): a file system based assembly designed to facilitate the exchange of digital content.

tDAR Metadata

tDAR Metadata is a relational PostgreSQL database in which the metadata entered by the user is stored.

tDAR Data

tDAR Data is a relational PostgreSQL database into which uploaded tabular data is copied. This data is not editable by the user: it there to simply to facilitate the sharing and integration of tabular data. The ultimate source of truth will always be the original uploaded file.

Lucene Indexes

Lucene expects to find its indexes on local storage. It also expects to have only one Lucene instance writing updates to the indexes.

Thoughts

As written, tDAR expects to find a file system available to store uploaded files on. tDAR manages the files within that file system and does not expect to find any other system, including other copies of tDAR, updating or modifying files on that file system. This is especially true of the Lucene index files.

This makes moving tDAR to the cloud a somewhat interesting exercise.

Ephemeral storage is simply not a viable option for the repository. A repository with a file system that can be arbitrarily reset to its initial state is not a repository.

Similarly, the two databases require non-ephemeral storage.

At the very simplest, persistent block storage that can be mounted and written to is required.

But even with mountable block storage available, the number of instances of tDAR that can be run, as written, is limited to a single instance. Thus moving tDAR to the NeCTAR cloud will result in an application that is less reliable and less performant than if tDAR were installed on dedicated hardware. This is not a desired outcome!

Ideally the PairTree and BagIt file stores should be replaced with implementations that make use of the available redundant object storage. However, this will involve some significant amount of effort, as the underlying internal semantics of the application will be changed.

This change will also impact the original tDAR development team, as it will mean making changes to an important and stable part of their application!

Perhaps a better approach would be to create a virtual file system that maps 'files' to objects in the object store. This could possibly be done by means of an implementation of the [FUSE](#) package, for example, [CloudFuse](#).

Regardless of approach, the sharing of the Lucene indexes amongst instances is going to be an interesting challenge to tackle.

Conclusion

As it currently stands tDAR is not a good fit for the NeCTAR cloud. The availability of persistent block storage is a necessary prerequisite before this migration can be considered. Only once persistent block storage becomes available can the migration can be started.

That migration has two possible paths: one is via a rewrite of some core components, the other is by means of presenting the cloud components through a façade that preserves the semantics expected by tDAR. Further work is required to establish the optimum path.

3.6 Authentication



FAIMS Repository Authorization and Authentication Land Scape ('Use Case')

Revision Chart

| Version | Primary Author(s) | Description of Version | Date Completed |
|-----------------------------|-------------------|-----------------------------------|---------------------------------|
| 0.1 - Draft Internal | Martin Paulo | Initial draft for internal review | 12 th December, 2012 |
| 0.2 – Draft External | Martin Paulo | Draft for external review | 12 th December, 2012 |
| 1.0 – Final | Martin Paulo | For release | 18 th December, 2012 |

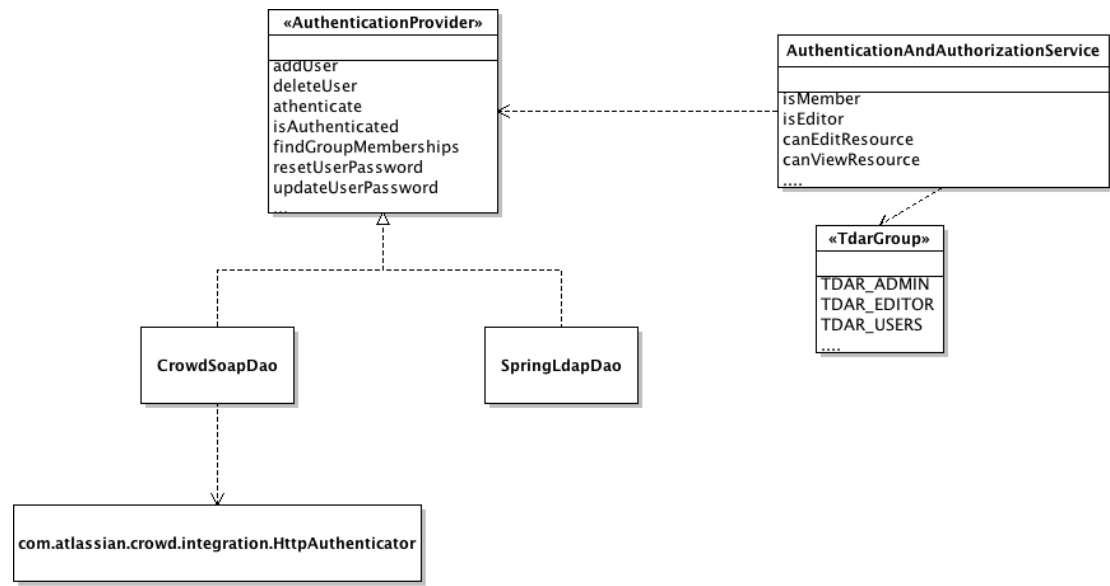
Introduction

The FAIMS repository will be build by scaling up the existing AHAD repository hosted at La Trobe University. The AHAD repository is in turn powered by tDAR, an open source digital archive and repository.

In effect, an existing large body of working Java software will have new features added to it. By this means the repository development is intended to leverage the cooperative nature of open source software, giving a low risk approach to delivering a repository that will allow Australian archaeological researchers to deposit and access data.

The Existing Code

The code in tDAR/AHAD is shown in the following UML class diagram:



tDAR makes use of the CrowdSoapDao class to perform authentication and authorization.

AHAD makes use of the SpringLdapDao class to perform authentication and authorization.

The diagrams shows that the AuthenticationProvider interface expects that in addition to authentication, implementers will be able to add and remove users, find their group memberships, manipulate their passwords, and know the current users authentication state.

These are methods used by the AuthenticationAndAuthorization service in performing its duties. This class also makes use of the TdarGroup enum to control access to resources. Thus it is important that the groups returned by the AuthenticationProvider can be mapped to this enum.

The CrowdSoapDoa delegates calls to the Atlassian HttpAuthenticator class, a part of the Atlassian Crowd application.

The SpringLdapDao simply wraps the popular Spring framework's LDAP implementation.

Atlassian Crowd

Crowd is a single sign-on and user identification tool for web applications. It has a pluggable authentication architecture, and as such supports Active Directory, LDAP, Open ID, and Crowd itself out of the box. It allows any combination of these authentication mechanisms to be used.

Crowd is a commercial product. However, if licenses are purchased, the purchasers get the right to perpetual use of the copy of Crowd purchased, and a copy of the source code. There are discounts for academic licenses, and further more, free licenses are available for open source projects and not-for-profit organizations.

NORDUnet have created a Shibboleth plugin for Crowd (<https://portal.nordu.net/display/NORDUwiki/Crowd+Shibboleth+Module>), thus extending its authentication services to users of Shibboleth. This plug in has been in production use for over a year now, and is available as open source under the NORDUnet IPR Policy (<https://portal.nordu.net/display/NORDUwiki/NORDUnet+IPR+Policy>).

This plugin is the Atlassian recommended path for those wishing to use Shibboleth in conjunction with Crowd.

Proposed Path

From a coding point of view, the simplest path forward would be see if the NORDUnet Shibboleth plugin works with the AAF. If so, then to acquire a license for Crowd that fulfills NeCTAR's requirements and move on.

If this path is not followed, then not only would we have to develop our own AAF authenticator, we would have to ensure that it implemented the expected semantics of the existing code.

Of interest is that these semantics expect group assignment to be done internally by the authentication provider, so (for example) administrators should be members of a remotely managed 'tdar-administrators' group. This does not quite fit with the AAF's model, so some sort of bridging application equivalent to Crowd would probably have to be developed if Crowd was an unacceptable solution.

The proposed path of adopting Crowd, if acceptable and workable, would have the added benefit of allowing all of the FAIMS projects disparate web components to use one central authentication and authorization mechanism for all users.

3.7 CAA 2013 Talk

AHAD

In this talk I intend to answer three questions that you might not know you were going to ask:

- What is AHAD?
- How did it come into being?
- And what does its the future hold?

As to what is AHAD? Well, it's an acronym.

It represents the “Australian Historical Archaeology Database”. That expansion gives clues as to AHAD's intent and substance.

From a technical view point, AHAD is simply a repository of digital archaeological data. A central location where the digital records of archaeological investigations are stored, organised and maintained.

A central location means that AHAD can offer a powerful search facility across the data under its control. So data is easy to find: which means that uploaded data can be shared amongst users, if so desired.

And not only shared: the data from different investigations can be integrated, to form new assemblies of data for download and further analysis. So AHAD can be a wonderful teaching tool.

Now, as a repository, AHAD is more than just a database.

This is an important distinction to make.

The technical term to understand is “bit rot”.

Digital media has a finite life. CD's that were bought in the late 80's are now anodising and those early digital artefact's are now becoming unplayable. And they were made out of aluminium. Hard drive's and backup tapes are, really, just rust. Data stored on them has a far short life span than those early CD's.

The only way to keep digital media from going off is to keep on making new copies of it.

So AHAD, as a repository, keeps all uploaded artefacts, in their original formats, versioned, in a file store. Those original artefacts are regularly tested to check their integrity.

By masquerading as a web site, AHAD allows people to access and use it by means of off the shelf everyday software with no installation required.

Being in the archaeological domain, AHAD uses the vocabulary of archaeologists in its user interface, so it is a comfortable tool for archaeologists to use.

So AHAD sounds wonderful. But how did it come into being?

Well, in the beginning was tDAR. Yes. tDAR is another acronym. tDAR represents “the Digital Archaeological Record”.

tDAR is simply a repository of archaeological data. A central location where the digital records of archaeological investigations are stored, organised and maintained.

Now, that should sound rather familiar. Rather AHAD like?

In fact, tDAR and AHAD share a common code base.

In 2010 the Faculty of Humanities and Social Sciences and the eResearch Office at La Trobe University, in partnership with the Victorian eResearch Strategic Initiative (VeRSI), received funding from the Australian National Data Service (ANDS) to build a national online database for historical archaeological catalogue data and associated stratigraphic and historical records.

Rather than developing new software from scratch, the team tasked with that development wisely researched existing software. And found tDAR, an open source application developed and in every day use under the aegis of the United States based Digital Antiquity organisation.

tDAR offered a lot of the features desired by AHAD, thus the team reached out to Digital Antiquity. And so it was that AHAD became powered by tDAR and now runs, hosted by La Trobe university.

But what of the future?

Much like that earlier AHAD team, the FAIMS team has chosen to reuse existing software where possible.

So in searching for a repository solution they were drawn to AHAD.

So going forward the AHAD/tDAR open source code base will be gaining new features: the ability to ingest data from mobile devices, a home in the cloud. Vocabularies and security features more in line with Australian archaeological needs. And a FAIMS skin.

Now I've said “open source” a number of times: there's a point I'd like you all to consider: what good are perfectly preserved digital artefacts in a repository if you don't have the source code or application required to work with those artefacts?

While your thinking about that I'll tell you that a copy of the repositories source code is actually placed on the server running the repository.

Powered, by tDAR, AHAD's aim was to catalyse collaboration and comparative study by facilitating the production and dissemination of compatible archaeological datasets here in Australia.

Now, thanks the FAIMS project and the choice of open source software, AHAD will be growing new features to support mobile devices, and moving to a new home in cloud.

3.8 Internal VerSI talk

From tDAR to FAIMS

In the beginning, there was tDAR.
Then tDAR begat AHAD.
And AHAD in turn, begat FAIMS.

Some terminology might be in order:
tDAR represents “the Digital Archaeological Record”
AHAD represents the “Australian Historic Archaeological Database”
FAIMS represents the “Federated Archaeological Information Management System”.

The common theme in these terms is “Archaeology”.

You see, tDAR is a repository for data that lies within the Archaeological domain.

It is an application that was started in 2006, in the United States, with the express objective of tackling the problem of how to synthesise systematically collected data recorded using different coding conventions, across multiple data sets and sites.

See, in Archaeology, practitioners have tended to record their finds in almost spreadsheet like formats, but to abbreviate the entries using unique codes and ontologies.

So if you want to consolidate data from one dig with data from another dig, you need to be able to map the different codes and ontologies as well.

And this is the problem that tDAR was developed to address.

From a technical view point, tDAR is simply a repository of archaeological data. A central location where the digital records of archaeological investigations are stored, organised and maintained.

A central location means that tDAR can offer a powerful search facility across the data under its control. So data is easy to find: which means that uploaded data can be shared amongst users, if so desired.

And not only shared: given the original goals, the data from different investigations can be compared by means of mapping the codings and ontologies, and then, in some cases, integrated, to form new assemblies of data for export and analysis.

To effectively support this kind of exploration tDAR has a security model that will allow confidential materials to remain protected. And tools to allow attribution.

As a repository, tDAR is more than just a database. The uploaded artefacts are kept, in their original formats, versioned, in a file store. Those original artefacts are regularly checked to ensure that their integrity is maintained.

A peek under the hood might be helpful in understanding tDAR in a bit more depth.

tDAR is simply two databases and a file store masquerading as a web site.

The file store contains versioned copies of the uploaded artefacts. This is the repository.

The first database contains data about the uploaded artefacts. This is the metadata database.

The second database contains an unpacked copy of any tabular (csv, excel, access, etc) data that has been uploaded: this allows that tabular data to be presented in the web view, and to have the users able to run reports and generate combined views between the contents of different tabular data sets.

The metadata database does contain some information about the data in the second database. But the contents of the second database are generated by ingesting the original database, and thus are dependent on the data that is uploaded.

There are only two prime sources of truth: the metadata describing the uploaded artefacts, and the file store containing those uploaded artefacts.

By masquerading as a web site, tDAR allows people to access and use it by means of off the shelf everyday software with no installation required.

In 2010 La Trobe University, in partnership with the VeRSI, received funding from the ANDS to build a national online database for historical archaeological data.

Rather than developing new software from scratch, the team tasked with that development researched existing software. And found tDAR.

tDAR offered a lot of the features desired by La Trobe, so thus the team reached out to the creators of tDAR.

And so it was that AHAD came into being and now runs, hosted by La Trobe university.

But we are moving into a new world. Most of us have mobile phones in our pockets that in reality are networked computers. Computers way more powerful than the desktop machines of a few years ago.

To help bring that new world to Australian archaeology the FAIMS project was funded by NeCTAR.

The goal of FAIMS is to assemble a comprehensive information system for archaeology that will serve in this new world of mobile devices and distributed computing. Data will be born using digital mobile devices, processed in on site databases, and then posted to an online repository hosted in the NeCTAR cloud.

Much like the earlier AHAD team, the FAIMS team has chosen to reuse existing software where possible in their goal of producing a suite of compatible tools. So in searching for a repository solution they were drawn to AHAD.

That's what I'm working on at the moment: customising the tDAR code base to be used as a basis

for the FAIMS repository.

And so it was that tDAR begat AHAD that begat FAIMs...

Questions?