

FAIMS Data, UI and Logic Cook-Book

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>
  <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>

```

```
</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>
  </group>
</group>

```



```
        </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>
              <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>

```

```
</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 addReln api.

e.g. addReln


```
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, with, 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/tab1/starttrackingtime", "click", "startTrackingGPS(\"time\", 10,
\"saveTimeGPSTrack()\")");
onEvent("controls/tab1/starttrackingdistance", "click",
"startTrackingGPS(\"distance\", 10, \"saveDistanceGPSTrack()\")");
onEvent("controls/tab1/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>-->

    <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>-->

        <!--><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 type 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.


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.