FAIMS Data, UI and Logic Cook-Book

FAIMS Data, UI and Logic Cook-Book

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

Project Creation

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

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

In addition to that, the project could contain other files such as:

1. properties file
2. validation schema

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

Simple Project

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

Project files

data_schema.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="data_schema.xsl"?>
<dataSchema name="SimpleProject" 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 project files to create a project on the server. Then download the project 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 type="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>
</dataSchema>
```

There are 2 types of elements for the data schema namely relationshipElement and archaeologicalElement. Relationship element defines the structure of the relationship entity to be saved into the database while archaeological element defines the structure of the archaeological entity to be saved into the database.

Relationship Element

Relationship element has attributes called name to define the relationship name to saved the record to and type. There are several elements contained in the relationship element:

1. description : the description of the relationship element
2. parent : the verb that defines the parent relationship to other entity
3. child : the verb that defines the child relationship to other entity
4. property : attribute of the relationship to be saved or loaded to the android app, it contains:

    - bundle
    - lookup : the vocabularies that is related to the attribute

Archaeological Element

Archaeological element has attribute called type to define the archaeological entity name to saved the record to. There are several elements contained in the relationship element:

1. description : the description of the archaeological element
2. property : attribute of the archaeological to be saved or loaded to the android app, it contains:

    - bundle
    - lookup : the vocabularies that is related to the attribute

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 View
- Checkbox Group View
- Radiobox Group View
- List View
- Date Picker
- Time Picker
- Map View
- Picture Slider
- 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 View

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 View

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="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 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 Slider

This example creates a picture slider.

```
...
        <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 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

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 load data from the database automatically into the TabGroup through logic calls. 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 load data from the database automatically into the TabGroup through logic calls. To tell which relationship type the TabGroup belongs to you can use the faims_rel_type attribute to specify the relationship 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">
```

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>

...

 <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.

Saving/Loading Arch Entity

Define a archaelogical entity in the data schema.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="data_schema.xsl"?>
<dataSchema name="SimpleProject" preparer="Your Name">
  <ArchaeologicalElement type="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 archaelogical 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: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);
}

saveEntity() {
  List attributes = createAttributeList();
  attributes.add(createEntityAttribute("name", getFieldValue("tabgroup1/tab1/name"),
null, null, getFieldCertainty("tabgroup1/tab1/name")));
  attributes.add(createEntityAttribute("value",
getFieldAnnotation("tabgroup1/tab1/value"), null,
getFieldValue("tabgroup1/tab1/value"), getFieldCertainty("tabgroup1/tab1/value")));

  String id = saveArchEnt(null, "Simple", null, attributes);

  update();
}

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

clearEntity() {
  newTabGroup("tabgroup1");
}

update();
```

How it works

- The data schema has defined a archaelogical 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 archaelogical 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.

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="SimpleProject" 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);
}

saveRelationship() {
  List attributes = createAttributeList();
  attributes.add(createRelationshipAttribute("name",
getFieldValue("tabgroup1/tab1/name"), null));

  String id = saveRel(null, "AboveBelow", null, attributes);

  update();
}

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

clearRelationship() {
  newTabGroup("tabgroup1");
}

update();
```

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

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

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")
```

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

```
showBusy("loading project", "please wait")
```

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

Fetching Records

UI designer can fetch records by calling one of the four methods fetchArchEnt, fetchRel, fetchOne, fetchAll. Each of these has different purpose of loading the records.

e.g. fetchArchEnt

```
fetchArchEnt("10000112441409729")
```

In the example, the function will return results for arch ent with uuid = 10000112441409729 if the uuid is exist, if not it will return null value.

e.g. fetchRel

```
fetchRel("10000112441409729")
```

In the example, the function will return results for arch ent with relationshipid = 10000112441409729 if the relationshipid is exist, if not it will return null value.

e.g. fetchOne

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

FetchOne is executing the query specified but only return one value from the query even though the query returns more than one result. It is usually useful to select one particular data from the database.

e.g. fetchAll

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

FetchAll is executing the query specified and return all values from the query. It is usually useful to select particular data from the database.

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 that the ui designer can fetch for the user from the database and generate a list of users. 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.

Note that getListItemValue only works if we bind click event to the 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 or not having picture url.

```
...
<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.

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

When a user wants to close the tab or tabgroup, the user can call the api cancelTab or cancelTabGroup. For each api, it has boolean warn in which determine whether a dialog should pop up to notify if there are any changes to the field values, annotation, or certainty.

e.g. cancelTab

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

The cancelTab will cancel and close the tab specified in the path <tabgroupname>/<tabname>. If the boolean value is true and there is any changes to the fields (value, annotation, certainty), the dialog will pop up asking whether the user wants to cancel the tab or not. If the user chooses OK, then the tab will be closed. If the boolean is false, the tab will be closed even there is any change to the field.

Note for cancelTab, the ui designer needs to specify which next tab it should be opened after canceling the tab by using showTab.

e.g. cancelTabGroup and goBack

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

The cancelTabGroup will cancel and close the tab group specified in the path <tabgroupname>. If the boolean value is true and there is any changes to the fields in any tab (value,annotation, certainty), the dialog will pop up asking whether the user wants to cancel the tab group or not. If the user chooses OK, then the goBack api will be called. If the boolean is false, the tab group will be closed even there is any change to the field. GoBack will bring the previous tab group open.

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.

Project Metadata

Project metadata can be shown on the UI by calling the apis getProjectName, getProjectId, getProjectSeason, getProjectDescription, getPermitNo, getPermitHolder, getContactAndAddress, and getParticipants.

e.g. Project metadata example

```
setFieldValue("tabgroup1/tab1/name", getProjectName());
setFieldValue("tabgroup1/tab1/id", getProjectId());
setFieldValue("tabgroup1/tab1/season", getProjectSeason());
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());
```

In the example, the project 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 project metadata on the device.

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

Maps and GIS

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

Rendering map raster layers

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>
...
```

You can render a raster map using the following logic. The reference to the map is relative to the maps folder inside the projects directory.

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

The raster map must be in projection EPSG:3785

Rendering shape vector layers

Rendering shape files can be done by adding vector layers to the map.

```
showShapeLayer("tabgroup1/tab1/map", "Shape Layer", "shape.shp", pointStyleSet,
lineStyleSet, polygonStyleSet);
```

The reference to the shape file is relative to the maps folder inside the projects directory. As for the pointStyleSet, lineStyleSet and polygonStyleSet, these will be described in more detail in the drawing geometry section below.

The shape file must be in projection EPSG:3785

Rendering spatialite vector layers

Rendering spatialite layers can be done by first generating a spatialite database with tables that contain GIS data.

```
showSpatialLayer("tabgroup1/tab1/map", "Spatial Layer", "db.sqlite", "table1",
"pk_id", "pk_label", pointStyleSet, lineStyleSet, polygonStyleSet);
```

A single database can contain multiple tables with gis data. You specify which table, id column, and label column to render as a spatialite layer. The reference to the database file is relative to the maps folder inside the projects directory.

The spatialite tables must contain a geometry column called Geometry.

The spatialite data must be in projection EPSG:3785

Using 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 projection EPSG:4326

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.

Creating canvas layers

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.

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.

Drawing points

To draw a point you must first create a point style set. This describes how the point will look like at various zoom levels.

To create a simple point style with a point style that will be active from zoom level 10 use the following.

```
pointStyle = createPointStyle(10, Color.RED, 0.1f, 0.3f);
```

Here we create a point style with color RED, size 0.1f and picking size 0.3f. The picking size is used when selecting the point on the map.

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 projection EPSG:4326

Drawing lines

To draw a line you must first create a line style set. This describes how the line will look like at various zoom levels.

To create a simple line style with a point style that will be active from zoom level 10 use the following.

```
lineStyle = createLineStyle(10,Color.GREEN, 0.1f, 0.3f, null);
```

Here we create a line style with color GREEN, with 0.1f, picking with 0.3f and null for point styles on the line. The point style in the line set is used to style points used to generate the line but by passing in null we define no points to be rendered.

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 projection EPSG:4326

Drawing polygons

To draw a polygon you must first create a polygon style set. This describes how the polygon will look like at various zoom levels.

To create a simple polygon styl with a point style that will be active from zoom level 10 use the following.

```
polygonStyle = createPolygonStyle(10, Color.BLUE, null);
```

Here we define a polygon style with color BLUE and no line style. The line style is used for the edges of the polygon.

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 projection EPSG:4326

Moving vector geometry

To move a geometry rendered on the map you must first select it and lock the geometry

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

Afterward, add the geometry to the highlight and prepare the geometry to be transformed to the new position.

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

onMapSelect() {
  geomId = getMapGeometrySelected();
  addGeometryHighlight("tabgroup1/tab1/map", currentGeometryId);
  prepareHighlightTransform("tabgroup1/tab1/map");
}
```

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

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 = new ArrayList();
collection.add(getGeometry("tabgroup1/tab1/map", geomId));
...
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);
  }
}
```

Maps and GPS

The following examples will show how to use GPS on maps.

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);
```

Syncing

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

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 occuring. 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);
```

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);
```

Show file browser

To attach files you first need to be able to select a file. You can show a file browser using the following.

```
showFileBrowser("getFilename()");

getFilename() {
 setFieldValue("tabgroup1/tab1/file", getLastSelectedFilename());
}
```

Here showFileBrowser opens a file browser view. Once you have selected a file the callback getFilename is executed. To get the file selected you can use the methods getLastSelectedFilename and getLastSelectedFilepath to refer to the files name and absolute path.

Attach server only files

To attach files for syncing with the server you can use the following.

```
fileId = attachFile(filepath, false);
```

The attachFile method copies the file into the projects files directory in either the server directory or app directory. All files in the server directory only get sent to the server. All files in the app directory get sent to the server and other apps. This example will copy the file into the server directory.

The fileId contains the relative path to the attached file within the projects directory.

Attach server and app files

To attach files for syncing with the server and other apps you can use the following.

```
fileId = attachFile(filepath, true);
```

Now the file will be copied into the app directory and will be synced with the server and other apps.