

# Lesson 2: The Table Widget and Basic Navigation

Created by Adriaan Klue, last modified by Jacques Marais on Nov 08, 2018

*As a System Administrator I want to view a list of System Administrators.*

- [Lesson Outcomes](#)
- [New & Modified App Files](#)
- [Adding Another View](#)
- [Collection Variables and Selectors](#)
- [Adding a Simple Table Widget](#)
- [More <attributeName> Children per Column](#)
- [Adding a View Button to Act on a Table Row](#)
- [Basic Navigation](#)
- [Lesson Source Code](#)

## Lesson Outcomes

We'll now start working towards managing our "System Admin" users in-app (with the role privileged to do so being itself a "System Admin" since it's highest in the hierarchy). First we want to see who we have in the system already, so we'll display them in a table. We'll then demonstrate interaction with specific rows in the table widget, which in this lesson will be navigating to a view showing the selected user's details.

Other widgets used in this lesson:

- `<button>`
- `<action>`

## New & Modified App Files

```
./web-app/images/people.png
./web-app/lang/en.lang
./web-app/presenters/SystemAdminUserMgmt.mez
./web-app/views/SystemAdminUserDetails.vxml
./web-app/views/SystemAdminUserMgmt.vxml
```

## Adding Another View

This new view also needs to be accessible via the menu, so we add a new view and a new `<menuitem>` precisely as in [Lesson 1](#), except with a different `view label` and `menuitem label`:

SystemAdminUserMgmt.vxml	
1	
2	<code>&lt;?xml version="1.0" encoding="UTF-8"?&gt;</code>
3	<code>&lt;ui xmlns="http://uiprogram.mezzanine.com/View"</code>
4	<code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>
5	<code>xsi:schemaLocation="http://uiprogram.mezzanine.com/View http://uiprogram.mezzanine.com/View.xsd"</code>
6	<code>&lt;view label="view_heading.system_admin_user_manage"</code>
7	<code>&lt;table widget="Table" data-bbox="100 800 600 900"&gt;</code>
8	<code>&lt;menuitem label="menu_item.system_admin_user_manage" view="SystemAdminUserDetails" data-bbox="100 920 300 950"/&gt;</code>

```

9      <userRole>System Admin</userRole>
10    </menuItem>
11  </view>
</ui>

```

#### en.lang

```

1  view_heading.system_admin_user_management = System Admin
2  menu_item.system_admin_user_management = System Admin U

```

## Collection Variables and Selectors

Even though at this point we have only one "System Admin", we'll be working with a collection variable - i.e. an array of entities - to cater for when we add more. Collections are covered more in-depth in [Lesson 8](#), but for now take note of how it is declared and of the `all()` function used to populate this collection from the database.

#### SystemAdminUserMgmt.mez

```

1  unit SystemAdminUserMgmt;
2  SystemAdmin[] systemAdmins;
3  void init() {
4      systemAdmins = SystemAdmin:all();
5  }

```

We refer to `all()` as a "selector", and it is one out of many such selectors you have available in lieu of SQL select queries. These are persistence built-in functions (BIFs) accessible from the object's namespace, e.g. `SystemAdmin`. For the full list of selectors, consult the [DSL Reference](#).

These BIFs can be used to build up a complex selector for reading sets of data from the persistence layer, and there will be examples of this as the tutorial code gets more complex. Here's a short example of the syntax you can expect to see in the near future:

```
activeSystemAdmins = SystemAdmin:and(equals(deleted, false),
```

## Adding a Simple Table Widget

Add a `<table>` widget beneath the `<menuItem>` tags:

#### SystemAdminUserMgmt.vxml

```

1  <table title="table_title.system_admins">
2    <collectionSource variable="systemAdmins"/>
3    <column heading="column_heading.first_name">
4

```

**i** The `<table>` widget's child elements need to be in a certain order to compile successfully, with the `<collectionSource>` first, then the `<column>` elements.

```

5      <attributeName>firstName</attributeName>
6    </column>
7    <column heading="column_heading.last_name">
8      <attributeName>lastName</attributeName>
9    </column>
</table>

```

The first child element is `<collectionSource>`. It contains a data source attribute that is either a `variable` or a `function` (in our case the former, which we declared above) which contains or returns a collection variable. After that we can add any number of columns (with a `heading` attribute, for fetching a heading from the lang file). The `<column>` element gets an `<attributeName>` child element that specifies which field on the object is to be displayed in that column. The above gives you:

System Administrators		Search	+
First name	Last name		
Test	User		
↓		Page 1 of 1	

## More <attributeName> Children per Column

We can compact our table a little by putting more than one `<attributeName>` under a single parent `<column>`:

### SystemAdminUserMgmt.vxml

```

1  <table title="table_title.system_admins">
2    <collectionSource variable="systemAdmins"/>
3    <column heading="column_heading.name">
4      <attributeName>firstName</attributeName>
5      <attributeName>lastName</attributeName>
6    </column>
7  </table>

```

Helium will automatically concatenate the values and separate them with spaces.

## Adding a View Button to Act on a Table Row

The `<table>` widget has an optional `<rowAction>` child element that adds a button to every row in the table. Consider a scenario where we do not want to display all of an object's fields in a summary table, and instead allow the user to click to navigate to a detail page.

### SystemAdminUserMgmt.vxml

```

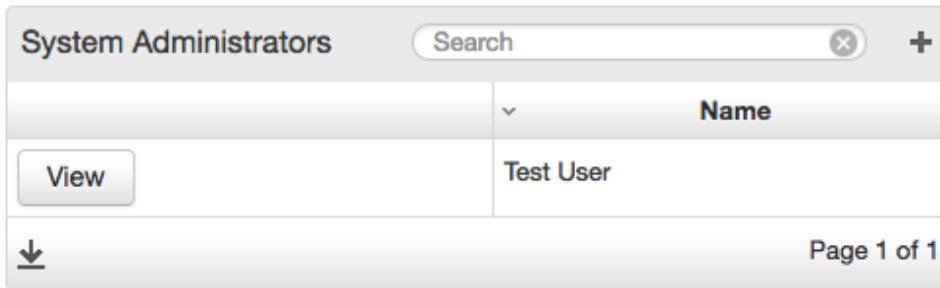
1  <table title="table_title.system_admins">
2    <collectionSource variable="systemAdmins"/>
3

```

```

4      <column heading="column_heading.name">
5          <attributeName>firstName</attributeName>
6          <attributeName>lastName</attributeName>
7      </column>
8      <rowAction label="button.view" action="viewUser">
9          <binding variable="selectedSystemAdmin" />
10     </rowAction>
</table>

```



The `action` attribute binds to a function (`viewUser`) in the presenter unit specified for this view (`SystemAdminUserMgmt`). (It could also have bounded to a function in a different presenter - then it would have been entered as `action="OtherUnit:viewUser"`.) This function can contain any amount of logic you need, but in our case we'll simply use it to navigate to a different view, which we'll get back to in a second.

The `binding` child element specifies the variable to which the selected record binds. It's how we obtain a handle on the user's selection. Clicking on the button does the assignment. Of course, the type of this variable must be of the same type as the collection source used to populate the table:

SystemAdminUserMgmt.mez	
1	SystemAdmin[] systemAdmins;
2	SystemAdmin selectedSystemAdmin;

More table features will be highlighted in the following lessons once we have populated it with more than one object.

## Basic Navigation

As mentioned in the section above, the "View" button is linked to the `viewUser` function, which we'll use to navigate to a new page. In order to do so, `viewUser` needs to return a member value from an enumerated type called `DSL_VIEWS`, which contains the names of all your views. Helium will automatically generate this enum when you run your app. (Although not used in this lesson, returning `null` where you would otherwise return a member of `DSL_VIEWS` keeps you on the same page.)

The first step is to create the user details view, this time without the `<menuItem>` widget because we don't want to navigate to it from the menu, only from this particular table. For this view's presenter we'll just use the same one linked to the view with the table. Simply add some `<info>` widgets as described in [Lesson 1](#). Note we're using the `selectedSystemAdmin` variable specified earlier as the `<rowAction>`'s binding variable to retrieve the selected user's details.

**i** Clicking your browser's back button will not open the previous page in your DSL app. It will instead browse "back" to your location before opening your DSL app. Similarly hitting refresh will refresh for the whole app and not your particular app view.

```

SystemAdminUserDetails.vxml
1  <view label="view_heading.system_admin_user_details" ur
2    <info label="info.first_name">
3      <binding variable="selectedSystemAdmin">
4        <attribute name="firstName"/>
5      </binding>
6    </info>
7    (etc.)

```

Also note that if our current view is bound to a different unit, i.e. not the same unit where `selectedSystemAdmin` was declared, we would have needed to specify the unit as well:

```

SystemAdminUserDetails.vxml
1  <info label="info.first_name">
2    <binding variable="SystemAdminUserMgmt:selectedSyst
3      <attribute name="firstName"/>
4    </binding>
5  </info>

```

If the new view is called `SystemAdminUserDetails.vxml`, we can now write our `viewUser` function in `SystemAdminUserMgmt` as:

```

SystemAdminUserMgmt.mez
1  DSL_VIEWS viewUser() {
2    return DSL_VIEWS.SystemAdminUserDetails;
3  }

```

And clicking the button will load the page.

We would like to be able to return to the view from whence we came, so let's create a "Back" button with the `<button>` widget on the details page and bind it to a function returning `DSL_VIEWS.SystemAdminUserMgmt`:

```

SystemAdminUserDetails.vxml
1  <view label="view_heading.system_admin_user_details" ur
2    <button label="action.back" action="back" />
3    <info label="info.first_name">
4      <binding variable="selectedSystemAdmin">
5        <attribute name="firstName"/>
6      </binding>
7    </info>


```


```

SystemAdminUserMgmt.mez
1
2  DSL_VIEWS back() {
3    return DSL_VIEWS.SystemAdminUserMgmt;

```

3 }

  
System  
Admin  
Home

  
System  
Admin  
Users

## System Admin User Details

Back

First name

Test

Last name

User

E-mail address:

user@domain.com


Mobile number:


278212345678

We can do one better. Helium's `<action>` widget positions itself in a spot more appropriate for navigating back, so we replace the `<button>` widget line with this:

#### SystemAdminUserDetails.vxml

```
<action label="action.back" action="back" />
```

  
System  
Admin  
Home

  
System  
Admin  
Users

## System Admin User Details

First name

Test

Last name

User

E-mail address:

user@domain.com

Mobile number:

278212345678

Note that all button variants also support the confirm dialog box. This is especially useful when a button is used to perform a critical or sensitive operation where the user must be extra sure before executing the logic backing the button. For more details on confirm and how it can be used with the different types of buttons, see the reference documentation [here](#) for more details.

## Lesson Source Code

[Lesson 2.zip](#)

