

2.x core Patient Fragment step by step tutorial

The UI Framework code is being transitioned to a module. Documentation will move to [UI Framework](#).

(Prerequisite: [UI Framework Step By Step Tutorial](#))

This guide will take you through the process of writing a Patient fragment, while following all [best practices](#).

A "Patient fragment" is (obviously) a page fragment that displays data about a patient. While such fragments are typically displayed on a patient dashboard, our best practices will also allow these fragments to be used on pages that aggregate data for multiple patients.

Fragment need to render their data to html on first page load (unless that is very expensive), and also be able to redraw themselves via AJAX, either self-initiated, or when signalled by another fragment on the page.

The example we'll use for this guide is the "Patient Identifiers fragment". We will display a table of PatientIdentifiers, and allow the user to add another (via a short popup form), or delete one (with confirmation) as long as it isn't the last one. This code is already checked into SVN, so you can't literally do these steps yourself, but you could (and should!) use this as a template for building new patient fragments.

Step 1: The controller (quick first pass)

To begin with, we need to write a controller that fetches the patient for us. By default, we want to use the patient from the shared model of the page this fragment is included on, but that should be overridable by a "patient" or "patientId" attribute in the fragment configuration, and if there is not patient available from either of those places, we should throw an exception. All that logic is encapsulated in the `FragmentUtil.getPatient` method.

Since we're calling our fragment "patientIdentifiers", by convention the controller should be at `org.openmrs.ui2.webapp.fragment.controller.PatientIdentifiersFragmentController`, and it needs a `controller()` method.

controller, first pass

```
/**
 * Controller for the PatientIdentifiers fragment.
 * Displays a table of PatientIdentifiers, and allow the user to add another (via a
 short popup form),
 * or delete one (with confirmation) as long as it isn't the last one.
 */
public class PatientIdentifiersFragmentController {

    /**
     * Controller method when the fragment is included on a page
     */
    public void controller(PageModel sharedPageModel, FragmentConfiguration config,
FragmentModel model) {
        model.addAttribute("patient", FragmentUtil.getPatient(sharedPageModel, config));
    }
}
```

You'll often need to fetch a bit more patient data than this, but since identifiers are directly on the patient object, a one-line function is sufficient.

Step 2: The view (quick first pass)

In our first pass at writing the view, we'll just display this data in a table. By convention this file should be at `/webapp/src/main/webapp/WEB-INF/fragments/patientIdentifiers.gsp`.

view, first pass

```
<%
  def id = config.id ? : ui.randomId("patientIdentifiers")
%>

<div id="${ id }">
  ${ ui.includeFragment("widgets/table", [
    columns: [
      [ property: "identifierType", heading:
ui.message("PatientIdentifier.type") ],
      [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
      [ property: "location", heading:
ui.message("PatientIdentifier.location") ]
    ],
    rows: patient.activeIdentifiers,
    ifNoRowsMessage: ui.message("general.none")
  ] ) }
</div>
```

It would be easy to write the html for the table ourselves, and this would be fine for a quick first pass, but in later steps we're going to want some more sophisticated behavior that we can get for free from the standard OpenMRS "table" widget, so we use that instead.

Note:

- for localization, we always use the `ui.message()` function instead of hardcoding user-visible messages in our views.
- for security, we always display user-entered data with appropriate escaping
 - in this case we use that's the `userEntered` setting on the table column, but if we were displaying data ourselves we'd use `ui.escapeHtml()` or `ui.escapeJs()`
- for UI consistency, we should always display standard OpenMRS classes with the `ui.format()` function. (That's taken care of here by the table widget.)

Step 3: Including our patient fragment in standard pages

In the [UI Framework tutorial](#), you created demonstration page to hold your fragments. Since we are now building real functionality, we want to include our fragment in the real user interface. Since the 2.x application is configurable and customizable, there is no single "patient dashboard" as in OpenMRS 1.x. Adding our fragment to the user interface actually means publishing it in the reference application's library of patient fragments, which are exposed as Extensions. To do this we need to add one line to the `org.openmrs.ui2.webapp.extension.CoreExtensionFactory` class:

Publishing our fragment

```
...
// PatientFragmentExtensions
ret.put("patientFragment.patientIdentifiers", new
PatientFragmentExtension("patientIdentifiers.fragment.title",
    "patientIdentifiers.fragment.description", "patientIdentifiers", null));
...
```

We are adding this extension to a Map. The key ("patientFragment.patientIdentifiers") is just a unique name for our extension point, and the value is a (strongly-typed) `PatientFragmentExtension`. Its constructor arguments are message codes for the label and description, the name of the fragment the extension represents, and (null in this case) fragment configuration attributes.

Once you've made this addition, you will need to rebuild and restart the web application. (This is the only time in the tutorial that you'll need to do

this.) Once you've rebuilt, and restarted jetty, you can visit a patient's page, and you will see an "Identifiers" tab.

(On the rare chance that you have manually configured the extension point for patient page tabs, you'll need to manually enable your new fragment.)

Step 4: initial ajax-ification of our widget

Typically anywhere that patient fragments are used, many of them are used together, and they typically all will allow small bits of data entry. As such, we want these widgets to be able to refresh themselves via ajax if other fragments notify them that a specific piece of patient data has changed. We'll start by implementing the ajax refresh, and deal with the message passing a bit later.

The first thing we need to do is write a fragment controller action (a server-side method in our fragment controller) that lets the client fetch the list of identifiers as JSON.

add a fragment action for fetching patient's active identifiers

```
/**
 * Controller for the PatientIdentifiers fragment.
 * Displays a table of PatientIdentifiers, and allow the user to add another (via a
 short popup form),
 * or delete one (with confirmation) as long as it isn't the last one.
 */
public class PatientIdentifiersFragmentController {

    /**
     * Controller method when the fragment is included on a page
     */
    public void controller(PageModel sharedPageModel, FragmentConfiguration config,
FragmentModel model) {
        model.addAttribute("patient", FragmentUtil.getPatient(sharedPageModel, config));
    }

    /**
     * Fragment Action for fetching list of active patient identifiers
     */
    public Object getActiveIdentifiers(UiUtils ui,
                                     @RequestParam("patientId") Patient patient) {
        return SimpleObject.fromCollection(patient.getActiveIdentifiers(), ui,
            "patientIdentifierId", "identifierType", "identifier", "location");
    }
}
```

Then we can write a short javascript function that will call this method, and refresh the table. (For rapid development, we're going to write this function directly in the fragment view, but we'll structure it so it can be pulled out into a shared javascript file at a future point.)

refresh function in the view

```
<%
  def id = config.id ?: ui.randomId("patientIdentifiers")
%>
<script>
  function refreshPatientIdentifierTable(divId, patientId) {
    jq('#' + divId + '_table > tbody').empty();
    jq.getJSON('${ ui.actionLink("getActiveIdentifiers", [returnFormat: "json"])
}', { patientId: patientId },
    function(data) {
      publish(divId + "_table.show-data", data);
    });
  }
</script>

<div id="${ id }">
  <a href="javascript:refreshPatientIdentifierTable('${ id }', ${ patient.patientId
})">test the refresh</a>

  ${ ui.includeFragment("widgets/table", [
    id: id + "_table",
    columns: [
      [ property: "identifierType", heading:
ui.message("PatientIdentifier.type") ],
      [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
      [ property: "location", heading:
ui.message("PatientIdentifier.location") ]
    ],
    rows: patient.activeIdentifiers,
    ifNoRowsMessage: ui.message("general.none")
  ] ) }
</div>
```

The "table" widget is capable of redrawing itself with new data (using the column definitions it was originally loaded with), so all we have to do is make a standard jQuery getJSON call to our fragment action, passing it the required patientId parameter, and tell the table to redraw itself with the returned data. (We do this by publishing a message we know it's listening for. TODO full documentation of the table fragment.)

In order to test that our code is working, we've also added a temporary button that says "test the refresh". If we reload the page and press the button, we'll see it work.

As an aside, note that here, and everywhere else in this fragment, we need to be careful to use the value of "patient.patientId", and not just a plain "patientId". If our fragment is included on a patient page, there will also be a "patient" model attribute, which will be identical to the patientId property of the "patient" model attribute. But in other contexts this won't be the case. So to keep our fragment flexible, remember to only refer to patient (which we explicitly set in the controller).

Step 5: listening for messages

In reality, we aren't going to have a "refresh" button. Rather we want our fragment to refresh automatically when told that the specific patient data it displays has been updated. (For a list of standardized messages people have already defined, see [the style guide](#).)

In our patientIdentifiers fragment, we want to redraw ourselves on three different messages:

- (OUR-FRAGMENT-ID).refresh (may be called by generic page decoration)
 - "patient/(id).changed" (the generic message that something unspecified about a patient has changed)
 - "patient/(id)/identifiers.changed" (the specific message that the patient's identifiers have changed)
- The latter two messages promise to include a "patient" javascript object as their message content, which always has a "patientId"

property, and may have an "activeIdentifiers" property, but the generic refresh message won't provide us anything.

We add a bit of javascript to listen for these messages:

Refresh on certain messages

```
<%
  def id = config.id ?: ui.randomId("patientIdentifiers")
%>
<script>
  function refreshPatientIdentifierTable(divId, patientId) {
    jq('#' + divId + '_table > tbody').empty();
    jq.getJSON('${ ui.actionLink("getActiveIdentifiers", [returnFormat: "json"])
}', { patientId: patientId },
    function(data) {
      publish(divId + "_table.show-data", data);
    });
  }

  function refreshPatientIdentifiers${ id }(message, data) {
    if (data && data.activeIdentifiers)
      publish("${ id }_table.show-data", data.activeIdentifiers);
    else
      refreshPatientIdentifierTable('${ id }', ${ patient.patientId });
  }

  subscribe('${ id }.refresh', refreshPatientIdentifiers${ id });
  subscribe('patient/${ patient.patientId }.changed', refreshPatientIdentifiers${ id
});
  subscribe('patient/${ patient.patientId }/identifiers.changed',
refreshPatientIdentifiers${ id });
</script>

<div id="${ id }">
  <a href="javascript:publish('${ id }.refresh')">div id refresh</a>
  <a href="javascript:patientChanged(${ patient.patientId })">patient changed</a>
  <a href="javascript:patientChanged(${ patient.patientId },
'identifiers')">identifiers changed</a>

  ${ ui.includeFragment("widgets/table", [
    id: id + "_table",
    columns: [
      [ property: "identifierType", heading:
ui.message("PatientIdentifier.type") ],
      [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
      [ property: "location", heading:
ui.message("PatientIdentifier.location") ]
    ],
    rows: patient.activeIdentifiers,
    ifNoRowsMessage: ui.message("general.none")
  ] ) }
</div>
```

We've added three "subscribe" commands, all of which call a new function, that's specific to this instance of our fragment. Note that we've intentionally put as much functionality as we can in the first function (refreshPatientIdentifierTable), which can eventually be refactored out into a cacheable javascript file, while leaving the minimum that must be included directly with the fragment's html in the second function

(refreshPatientIdentifiers + id).

We've also changed our temporary test link to test all three cases. The first publishes a refresh message, the next two call a utility function defined in openmrs.js which publishes a patient changed message with the correct payload.) So at this point if you reload the page, and click on the three links, you will see the fragment update correctly.

Step 6: Letting users add an identifier

Now that we're done ajaxifying our fragment, we will implement functionality to let the user add a new identifier.

First, we need to write a fragment action that adds the specified identifier and saves the patient. This is a quick first-pass that will reload the whole page when it's submitted.

Add Identifier fragment action, first pass

```
/**
 * Fragment Action for adding a new identifier
 */
public ActionResult addIdentifier(UiUtils ui,
                                @RequestParam("patientId") Patient patient,
                                @RequestParam("identifierType")
PatientIdentifierType idType,
                                @RequestParam("identifier") String
identifier,
                                @RequestParam("location") Location
location) {
    patient.addIdentifier(new PatientIdentifier(identifier, idType, location));
    Context.getPatientService().savePatient(patient);
    return new SuccessResult(ui.message("patientIdentifier.added"));
}
```

Next we need a form on the client side that will submit to this fragment action. We're going to take advantage of the "popupForm" widget, which both lets us write this functionality very quickly, and we can count on to fit into the user interface in a consistent way.

fragment view, now with add button

```
<%
    def id = config.id ?: ui.randomId("patientIdentifiers")
%>
<script>
    function refreshPatientIdentifierTable(divId, patientId) {
        jq('#' + divId + '_table > tbody').empty();
        jq.getJSON('${ ui.actionLink("getActiveIdentifiers", [returnFormat: "json"])
}', { patientId: patientId },
        function(data) {
            publish(divId + "_table.show-data", data);
        });
    }

    function refreshPatientIdentifiers${ id }(message, data) {
        if (data && data.activeIdentifiers)
            publish("${ id }_table.show-data", data.activeIdentifiers);
        else
            refreshPatientIdentifierTable('${ id }', ${ patient.patientId });
    }
</script>
```

```

    subscribe('${ id }.refresh', refreshPatientIdentifiers${ id });
    subscribe('patient/${ patient.patientId }.changed', refreshPatientIdentifiers${ id
});
    subscribe('patient/${ patient.patientId }/identifiers.changed',
refreshPatientIdentifiers${ id });
</script>

<div id="${ id }">
    ${ ui.includeFragment("widgets/table", [
        id: id + "_table",
        columns: [
            [ property: "identifierType", heading:
ui.message("PatientIdentifier.identifierType") ],
            [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
            [ property: "location", heading:
ui.message("PatientIdentifier.location") ]
        ],
        rows: patient.activeIdentifiers,
        ifNoRowsMessage: ui.message("general.none")
    ] ) }

    ${ ui.includeFragment("widgets/popupForm",
    [
        id: id + "_add",
        buttonLabel: ui.message("general.add"),
        popupTitle: ui.message("patientIdentifier.add"),
        fragment: "patientIdentifiers",
        action: "addIdentifier",
        submitLabel: ui.message("general.save"),
        cancelLabel: ui.message("general.cancel"),
        fields: [
            [ hiddenInputName: "patientId", value: patient.patientId ],
            [ label: ui.message("PatientIdentifier.identifierType"),
formFieldName: "identifierType", class: org.openmrs.PatientIdentifierType ],
            [ label: ui.message("PatientIdentifier.identifier"), formFieldName:
"identifier", class: java.lang.String ],
            [ label: ui.message("PatientIdentifier.location"), formFieldName:
"location", class: org.openmrs.Location ]
        ]
    ] ) }

```

```
    1) }  
</div>
```

We've added the "popupForm" fragment, which will give us:

- a button (labeled with "buttonLabel") that pops up a form in a dialog (titled by "popupTitle")
- the form submits to the given "action" of the given "fragment"
- the form will have submit and cancel buttons labeled with "submitLabel" and "cancelLabel"
- the form will have four fields: one hidden input, and three visible fields, with the specified labels
 - the first is a hidden input, with the given name and value
 - the next three will actually delegate to the "labeledField" widget for the given class, with the given label and form field name
 - TODO document the field widget

Also, we've removed our temporary testing links.

If you reload the page, you'll now see an "Add" button below the table, which pops up a modal dialog that lets you add an identifier, and reloads the page. (In a later step, we'll fix this up so it refreshes things via our ajax functions.)

Step 7: Letting users delete an identifier

Now that users can add identifiers, we also want to let them delete identifiers. In first pass we'll add this to the page such that deleting an identifier reloads the page, and we'll ajaxify the interaction in a later step.

First, we need a fragment action for deleting identifiers. This is straightforward:

Delete Identifier fragment action, first pass

```
/**  
 * Fragment Action for deleting an existing identifier  
 */  
public ActionResult deleteIdentifier(UiUtils ui,  
                                     @RequestParam("patientIdentifierId")  
Integer id) {  
    PatientService ps = Context.getPatientService();  
    PatientIdentifier pid = ps.getPatientIdentifier(id);  
    ps.voidPatientIdentifier(pid, "user interface");  
    return new SuccessResult(ui.message("PatientIdentifier.deleted"));  
}
```

Adding this to the view is a bit more complicated. The "table" widget we are using supports having columns with lists of "actions", so we'll take advantage of that:

fragment view, now with delete button

```
<%  
    def id = config.id ?: ui.randomId("patientIdentifiers")  
%>  
<script>  
    function refreshPatientIdentifierTable(divId, patientId) {  
        jq('#' + divId + '_table > tbody').empty();  
        jq.getJSON('${ ui.actionLink("getActiveIdentifiers", [returnFormat: "json"])  
'}, { patientId: patientId },  
            function(data) {  
                publish(divId + "_table.show-data", data);  
            });  
    };
```

```

    }

    function refreshPatientIdentifiers${ id }(message, data) {
        if (data && data.activeIdentifiers)
            publish("${ id }_table.show-data", data.activeIdentifiers);
        else
            refreshPatientIdentifierTable('${ id }', ${ patient.patientId });
    }

    subscribe('${ id }.refresh', refreshPatientIdentifiers${ id });
    subscribe('patient/${ patient.patientId }.changed', refreshPatientIdentifiers${ id
});
    subscribe('patient/${ patient.patientId }/identifiers.changed',
refreshPatientIdentifiers${ id });

    subscribe('${ id }_table.delete-button-clicked', function(message, data) {
        if (openmrsConfirm('${ ui.message("general.confirm") }')) {
            jq.post('${ ui.actionLink("deleteIdentifier") }', { returnFormat: 'json',
patientIdentifierId: data },
            function(data) {
                location.reload(true);
            })
            .error(function() {
                notifyError("Programmer error: delete identifier failed");
            })
        }
    });
</script>

<div id="${ id }">
    ${ ui.includeFragment("widgets/table", [
        id: id + "_table",
        columns: [
            [ property: "identifierType", heading:
ui.message("PatientIdentifier.identifierType") ],
            [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
            [ property: "location", heading:
ui.message("PatientIdentifier.location") ],
            [ actions: [
                [ action: "event",
                    icon: "delete24.png",
                    tooltip: ui.message("PatientIdentifier.delete"),
                    event: id + ".delete-button-clicked",
                    property: "patientIdentifierId" ]
            ] ]
        ],
        rows: patient.activeIdentifiers,
        ifNoRowsMessage: ui.message("general.none")
    ]) }

    ${ ui.includeFragment("widgets/popupForm",
    [
        id: id + "_add",
        buttonLabel: ui.message("general.add"),
        popupTitle: ui.message("PatientIdentifier.add"),
        fragment: "patientIdentifiers",
        action: "addIdentifier",
        submitLabel: ui.message("general.save"),
    ]
    )
    }

```

```
cancelLabel: ui.message("general.cancel"),
fields: [
    [ hiddenInputName: "patientId", value: patient.patientId ],
    [ label: ui.message("PatientIdentifier.identifierType"),
formFieldName: "identifierType", class: org.openmrs.PatientIdentifierType ],
    [ label: ui.message("PatientIdentifier.identifier"), formFieldName:
"identifier", class: java.lang.String ],
    [ label: ui.message("PatientIdentifier.location"), formFieldName:
"location", class: org.openmrs.Location ]
]
```

```
    1) }  
</div>
```

(Note: the details of this may change when the "table" widget is refactored - TRUNK-2060)

(Note that I had to change 'event: id + ".delete-button-clicked"' to 'event: id + "_table.delete-button-clicked"' to get this to work - Mark)

We've added two things: an action as one of the table columns, and an event subscription that listens for the button being clicked. (It might seem more natural if the action in the table row called a plain javascript function, but the event mechanism makes it easier for the table widget to handle the same action on the initial page load, and when the table is populated via ajax.)

The newly-added last column of the table is a list of actions (containing just one action). Visually, we define an icon (TODO document this) and a tooltip. Logically, we define an event to post (including the id of this fragment, so that it is safe to include on a page multiple times), and a property, whose value for the clicked row will be published as additional data in the message.

In the newly-added event subscription, we listen for (FRAGMENT-ID).delete-button-clicked, namely the event we defined in the action. The callback function we register with the subscription counts on being passed the patientIdentifierId (i.e. the "property" on the action) as extra data. We ask the user to confirm their action using the openmrsConfirm function. (Currently this just calls the standard Javascript confirm function, but by using our own OpenMRS method, we'll be able to make the look and feel prettier in the future, by changing code in just one place.) Assuming the user confirms the deletion, we do a standard jQuery post to our new deleteIdentifier action, passing the relevant id as data, and reloading the page on success. (This is a placeholder: we'll ajaxify shortly.) Finally, we define a function to be called on error. It is good practice to handle errors from every ajax call you make: displaying anything at all (even something not particularly meaningful to the end user) is better than a silent error.

If you reload your page, you'll now be able to add and remove identifiers.

Step 8: Ajaxifying the add and remove identifier interactions

Changing the add and remove actions so that they work via AJAX is actually quite easy, since we can let the UI framework do most of the hard work for us. The first things we need to do is change our addIdentifier and deleteIdentifier fragment actions to have them return Objects. All ajaxified patient fragments are expected to publish a patient changed event, with a javascript representation of the Patient object in a standard format. We are going to use a standard utility method to produce the json-ready patient object. (This utility method may not contain all the properties you want. You may want to tweak the method to return a patient object with more properties, but don't overdo it. You may also construct a result with the utility methods in the SimpleObject class.)

Controller, almost final

```
/**
 * Controller for the PatientIdentifiers fragment.
 * Displays a table of PatientIdentifiers, and allow the user to add another (via a
 short popup form),
 * or delete one (with confirmation) as long as it isn't the last one.
 */
public class PatientIdentifiersFragmentController {

    /**
     * Controller method when the fragment is included on a page
     */
    public void controller(PageModel sharedPageModel, FragmentConfiguration config,
    FragmentModel model) {
        model.addAttribute("patient", FragmentUtil.getPatient(sharedPageModel, config));
    }

    /**
     * Fragment Action for fetching list of active patient identifiers
     */
    public Object getActiveIdentifiers(UiUtils ui, @RequestParam("patientId") Patient
    patient) {
        return SimpleObject.fromCollection(patient.getActiveIdentifiers(), ui,
    "patientIdentifierId", "identifierType", "identifier",
        "location");
    }

    /**
     * Fragment Action for adding a new identifier
     */
    public Object addIdentifier(UiUtils ui, @RequestParam("patientId") Patient patient,
        @RequestParam("identifierType") PatientIdentifierType idType,
    @RequestParam("identifier") String identifier,
        @RequestParam("location") Location location) {
        patient.addIdentifier(new PatientIdentifier(identifier, idType, location));
        Context.getPatientService().savePatient(patient);
        return FragmentUtil.standardPatientObject(ui, patient);
    }

    /**
     * Fragment Action for deleting an existing identifier
     */
    public Object deleteIdentifier(UiUtils ui, @RequestParam("patientIdentifierId")
    Integer id) {
        PatientService ps = Context.getPatientService();
        PatientIdentifier pid = ps.getPatientIdentifier(id);
        ps.voidPatientIdentifier(pid, "user interface");
        return FragmentUtil.standardPatientObject(ui, pid.getPatient());
    }
}
}
```

The only change we've made is to return an Object representing a standard patient, instead of a SuccessResult.

Next, we need to change the add and delete actions in the view to support these new object return values (as JSON):

View, almost final

```
<%
  def id = config.id ?: ui.randomId("patientIdentifiers")
%>
<script>
  function refreshPatientIdentifierTable(divId, patientId) {
    jq('#' + divId + '_table > tbody').empty();
    jq.getJSON('${ ui.actionLink("getActiveIdentifiers", [returnFormat: "json"])
}', { patientId: patientId },
    function(data) {
      publish(divId + "_table.show-data", data);
    });
  }

  function refreshPatientIdentifiers${ id }(message, data) {
    if (data && data.activeIdentifiers)
      publish("${ id }_table.show-data", data.activeIdentifiers);
    else
      refreshPatientIdentifierTable('${ id }', ${ patient.patientId });
  }

  subscribe('${ id }.refresh', refreshPatientIdentifiers${ id });
  subscribe('patient/${ patient.patientId }.changed', refreshPatientIdentifiers${ id });
  subscribe('patient/${ patient.patientId }/identifiers.changed',
refreshPatientIdentifiers${ id });

  subscribe('${ id }.delete-button-clicked', function(message, data) {
    if (openmrsConfirm('${ ui.message("general.confirm") }')) {
      jq.post('${ ui.actionLink("deleteIdentifier") }', { returnFormat: 'json',
patientIdentifierId: data },
      function(data) {
        notifySuccess('${ ui.escapeJs(ui.message("PatientIdentifier.deleted"))
}');
        publish('patient/${ patient.patientId }/identifiers.changed', data);
      }, 'json')
      .error(function() {
        notifyError("Programmer error: delete identifier failed");
      })
    }
  });
</script>

<div id="${ id }">
  ${ ui.includeFragment("widgets/table", [
    id: id + "_table",
    columns: [
      [ property: "identifierType", heading:
ui.message("PatientIdentifier.identifierType") ],
      [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
      [ property: "location", heading:
ui.message("PatientIdentifier.location") ],
      [ actions: [
        [ action: "event",
```

```

        icon: "delete24.png",
        tooltip: ui.message("PatientIdentifier.delete"),
        event: id + ".delete-button-clicked",
        property: "patientIdentifierId" ]
    ] ]
],
rows: patient.activeIdentifiers,
ifNoRowsMessage: ui.message("general.none")
]) }

${ ui.includeFragment("widgets/popupForm", [
    id: id + "_add",
    buttonLabel: ui.message("general.add"),
    popupTitle: ui.message("PatientIdentifier.add"),
    fragment: "patientIdentifiers",
    action: "addIdentifier",
    submitLabel: ui.message("general.save"),
    cancelLabel: ui.message("general.cancel"),
    fields: [
        [ hiddenInputName: "patientId", value: patient.patientId ],
        [ label: ui.message("PatientIdentifier.identifierType"),
formFieldName: "identifierType", class: org.openmrs.PatientIdentifierType ],
        [ label: ui.message("PatientIdentifier.identifier"), formFieldName:
"identifier", class: java.lang.String ],
        [ label: ui.message("PatientIdentifier.location"), formFieldName:
"location", class: org.openmrs.Location ]
    ],
    successEvent: "patient/" + patient.patientId + "/identifiers.changed"

```

```
        1) }  
</div>
```

Handling the Add Identifier action is easy: we just need to add a "successEvent" configuration attribute to the popupForm widget. This automatically tells the form widget it should post data via AJAX, instead of a regular form submission, and publish the specified event, with the post's returned JSON value as a payload.

Handling the Remove Identifier action is easy as well. We had already defined a function callback for the delete-button-clicked event that did an AJAX post via jQuery. Now we just modify its success callback function slightly (to show a success message, and publish the patient changed message) and specify the 'json' return type (otherwise jQuery would pass a String to our success function instead of a javascript object).

Once you've made those changes, you should be able to reload the page, and add and remove multiple identifiers without having to do a full page reload.

Note: you'll probably notice a few framework bugs as you play around with the add and remove, specifically:

- TRUNK-2180 - form widget needs to clear itself after a successful ajax submission

Step 9: Additional validation

We've actually missed a bit of necessary validation: we shouldn't blindly just delete the identifier without doing some checks first:

- we shouldn't delete an identifier that has already been deleted (in case the user is looking at an old version of the page)
- we shouldn't delete the last unvoided identifier

In case of any errors while executing a fragment action, you just need to return a `FailureResult`, and the UI framework will take care of sending it back in the correct way. (If the fragment is called via ajax, an errors object is returned as json or xml. If the fragment is done as a regular form submission, it is returned in a session attribute and displayed at the top of the page.) Typically fragment actions will declare a return type of `Object`, so they may return either a `Success/Object` result, or a `FailureResult`.

The simplest `FailureResult` constructor takes a single error message--that's the one we'll use here.

First, we add the validation to our fragment action method:

Further validation in delete action

```
/**  
 * Fragment Action for deleting an existing identifier  
 */  
public Object deleteIdentifier(UiUtils ui, @RequestParam("patientIdentifierId")  
Integer id,  
                                @RequestParam(value="reason",  
defaultValue="user interface") String reason) {  
    PatientService ps = Context.getPatientService();  
    PatientIdentifier pid = ps.getPatientIdentifier(id);  
    // don't touch it if it's already deleted  
    if (pid.isVoided())  
        return new FailureResult(ui.message("PatientIdentifier.delete.error.already"));  
    // don't delete the last active identifier  
    if (pid.getPatient().getActiveIdentifiers().size() == 1) {  
        return new FailureResult(ui.message("PatientIdentifier.delete.error.last"));  
    }  
    // otherwise, we go ahead and delete it  
    ps.voidPatientIdentifier(pid, reason);  
    return FragmentUtil.standardPatientObject(ui, pid.getPatient());  
}
```

We also need to make one small change to the view in order to actually display those errors. Before we were flashing a generic message, that wasn't actually helpful to the user. Instead, we just pass the jQuery xml http response (it's the first argument that jQuery passes to the failure function for any ajax call) to a utility javascript method that the framework provides:

changing the view to support error messages raised by the delete action via ajax

```
subscribe('${ id }.delete-button-clicked', function(message, data) {
    if (openmrsConfirm('${ ui.message("general.confirm") }')) {
        jq.post('${ ui.actionLink("deleteIdentifier") }', { returnFormat: 'json',
patientIdentifierId: data },
        function(data) {
            notifySuccess('${ ui.escapeJs(ui.message("PatientIdentifier.deleted"))
}');
            publish('patient/${ patient.patientId }/identifiers.changed', data);
        }, 'json')
        .error(function(xhr) {
            fragmentActionError(xhr, "Programmer error: delete identifier
failed");
        })
    }
});
```

Step 10: Additional features

Up until this point, we've covered the very basic functionality that pretty much every patient-based list of items will have. Namely, you can display the list, and add and remove items. Now let's add another feature that's a bit custom for this particular fragment: we should visually identify which one of the identifiers is the preferred one (it should already be at the top of the list) and allow the user to mark any non-preferred identifier as the newly-preferred one.

Of course, we need to write a fragment action to support setting a non-preferred identifier as preferred:

Fragment Action for changing the preferred identifier

```
/**
 * Fragment Action for marking an identifier as preferred
 */
public Object setPreferredIdentifier(UiUtils ui,
                                   @RequestParam("patientIdentifierId")
                                   PatientIdentifier pid) {
    PatientService ps = Context.getPatientService();
    if (pid.isVoided())
        return new
FailureResult(ui.message("PatientIdentifier.setPreferred.error.deleted"));
    // silently do nothing if it's already preferred
    if (!pid.isPreferred()) {
        // mark all others as nonpreferred
        for (PatientIdentifier activePid : pid.getPatient().getActiveIdentifiers()) {
            if (!pid.equals(activePid) && activePid.isPreferred()) {
                activePid.setPreferred(false);
                ps.savePatientIdentifier(activePid);
            }
        }
        // mark this one as preferred
        pid.setPreferred(true);
        ps.savePatientIdentifier(pid);
    }
    return FragmentUtil.standardPatientObject(ui, pid.getPatient());
}
```

This code is straightforward. The only difference between this example and those in previous steps is that we're converting the `patientIdentifierId` parameter directly into a `PatientIdentifier` directly in the method signature, using Spring's automatic type conversion. (Doing this required adding a converter class, which is documented [Type Converters](#).)

The next step is to add a column to the table in the `gsp` page, which shows either a preferred, or non-preferred icon. The non-preferred icon should be clickable.

additions to the view to support changing the preferred patient identifier

```
...
    subscribe('${ id }.set-preferred-identifier', function(message, data) {
        jq.post('${ ui.actionLink("setPreferredIdentifier") }', { returnFormat:
'json', patientIdentifierId: data },
        function(data) {
            notifySuccess('${
ui.escapeJs(ui.message("PatientIdentifier.setPreferred")) }');
            publish('patient/${ patient.patientId }/identifiers.changed', data);
        }, 'json')
        .error(function(xhr) {
            fragmentActionError(xhr, "Failed to set preferred identifier");
        })
    });
...
    ${ ui.includeFragment("widgets/table", [
        id: id + "_table",
        columns: [
            [ property: "identifierType", heading:
ui.message("PatientIdentifier.identifierType") ],
            [ property: "identifier", userEntered: true, heading:
ui.message("PatientIdentifier.identifier") ],
            [ property: "location", heading:
ui.message("PatientIdentifier.location") ],
            [ heading: ui.message("PatientIdentifier.preferred"),
            actions: [
                [ action: "none",
                    icon: "star_16.png",
                    showIfPropertyTrue: "preferred" ],
                [ action: "event",
                    icon: "star_off16.png",
                    event: id + ".set-preferred-identifier",
                    property: "patientIdentifierId",
                    showIfPropertyFalse: "preferred" ]
            ]
        ],
        [ actions: [
            [ action: "event",
                icon: "delete24.png",
                tooltip: ui.message("PatientIdentifier.delete"),
                event: id + ".delete-button-clicked",
                property: "patientIdentifierId" ]
        ] ]
    ],
    rows: patient.activeIdentifiers,
    ifNoRowsMessage: ui.message("general.none")
    ] ) }
...

```

We add a callback function that will set a preferred identifier by doing a jQuery ajax post, which will be activated based on a message. Then we add a new column definition to the table, that contains two actions. These actions are different from what we've seen before because they are marked as being conditional by the attributes "showIfPropertyTrue" and "showIfPropertyFalse".

(Actually I just implemented this conditional display in TRUNK-2186. Generally speaking, if you're writing a fragment that needs functionality that will likely be needed elsewhere as well, you should be thinking about how to build this functionality in a shared way.)

So if we refresh our browser at this point, we'll see a new column, with a star that is either highlighted or greyed out, depending on whether the identifier is preferred.

When I actually clicked around on these I noticed some odd behavior (now fixed in the head revision of the project): marking an identifier as preferred doesn't immediately move it to the top of the list on the ajax refresh, but it *does* move to the top of the list the next time the page is loaded. Basically, there's a bug in the core OpenMRS API where changing the preferred identifier is not reflected in the `patient.getActiveIdentifiers()` method until the next time the patient is loaded from the database. I reported this as TRUNK-2188, and I introduced a workaround in the `SimplePatient` object used indirectly by `FragmentUtil.standardPatientObject()`.

Step 11: Refactoring

Early on in the tutorial, we referred to splitting up our javascript functionality into that which is specific to this instance of the fragment, and that which can be split out into an external resource file. (Any javascript we can put in an external resource file can be cached by the browser, and perhaps minimized by the UI framework, thus speeding up the application over slow internet connections.) Let's go ahead and split that code out.

As we've written things so far, only a single function ("refreshPatientIdentifierTable") can be pulled out. We could rewrite some of the other methods as well, but we'll leave that as an exercise for later. (Key point: javascript that's being moved into a shared resource file must not know about the configuration of a specific instance of the fragment, nor may it reference the fragment's id, or use the 'ui' groovy functions.)

The resource file we build will be cached, but over a satellite internet connection, even checking whether a cached resource has changed can slow down a page, so we're going to combine javascript functions for *all* of the patient fragments into a single file.

So, let's open "webapp/src/main/webapp/scripts/coreFragments.js", and move our function in there:

moving javascript into an external resource

```
var patientIdentifiersFragment = {  
  
    refreshPatientIdentifierTable: function(divId, patientId) {  
        $('#' + divId + '_table > tbody').empty();  
        $.getJSON(actionLink('patientIdentifiers', 'getActiveIdentifiers', {  
returnFormat: "json", patientId: patientId })), function(data) {  
            publish(divId + "_table.show-data", data);  
        });  
    }  
  
}
```

We've changed two things while moving this function here:

1. since we're going to be gathering functions for many fragments in this file, we create a "patientIdentifiersFragment" object that contains all functions for our fragment. (Just one, for now.)
2. we can't call the groovy `ui.actionLink` method, so we use a javascript function (defined in `openmrs.js`) instead.

As a result of this we need to change one line in `patientIdentifiers.gsp` to call this function with the "patientIdentifiersFragment." object prefix:

```
patientIdentifiersFragment.refreshPatientIdentifierTable('${ id }', ${  
patient.patientId })
```