Fleet Management Viewports, Modals and Tabs Lab

# Contextual Sidebar

The contextual sidebar is a popular application of viewports. You may have noticed that when creating templated pages in San Diego and prior you get a warning that “The page template has a protection policy”. If so, what’s the point of a template I can’t modify? One answer is viewports.

Using viewports you can create sub-pages separate from the main page and then use configuration to add tabs that use those sub-pages without changing the parent page.

To use this lab, first fork and install the Fleet Management app from Github: <https://github.com/ServiceNowNextExperience/flt-mgmt>

## Vehicle History Contextual Sidebar

This sidebar tab will show the history of requests involving the vehicle for which a request is currently being viewed. This means creating a new viewport page to render the list, retrieving information about the currently selected vehicle into the viewport page and adding a route tied to the contextual side bar.

Also note: I have already implemented this in the version you retrieved from source control (titled “Vehicle Request History (Reference)” if you wish to look at my configurations as you implement your own.

### Create the subpage

1. Open App Engine Studio or UI Builder and open the Fleet Vehicle Management workspace experience and switch to the “Record” page and the “default” variant.
2. Set the {table} = “x\_snc\_flt\_mgmt\_maint\_req” and {sysId} = “d7a83bda1b3819101363ff37dc4bcb43” parameters at the top of the palette in the “URL” bar.
3. It may take a few seconds, but you will now see the contextual side bar and a big “Edit Content” button (Don’t click this!).
   1. Since this is a shared template by default every Record page has the same tabs, which leads to...
   2. …this gnarly list of Tabs in the Config panel for Contextual sidebar that scrolls forever. Don’t worry—this really is as horrible as it looks because every single contextual sidebar tab is listed, but you can collapse the sections and only concern yourself with your own tabs. Only the applicable tabs will show when the page is rendered.
4. In order to add a new Vehicle History tab start by clicking “Add” in the Tabs section of the Contextual Sidebar Config.
   1. Name: Vehicle History
   2. Order: 0
   3. Template: None
5. Just click “done” on the “Success!” modal and don’t add any parameters.
6. You will notice that you seem to have left your Record page palette. You have! You are now editing your new viewport subpage, but to get back just use the “🡨 Back to Record Default” to return to the parent page.

### Add the Vehicle ID property to the subpage

We will use this property to reach out to the parent page to get the vehicle’s sys id so that we can query it inside the viewport. (Only properties at the “Body” level of a subpage may access the values of the parent).

1. Select “Body” in the content tree of the subpage.
2. On the Config panel next to “Properties” click the “edit” gear.
3. Select “+Add” and add two new page properties:
   1. Name: vehicleId, Type: String
   2. Name: currentMaintReq, Type: String
   3. Click “Add” button then “Save”
4. In the new “vehicleId” property click the “data” icon and bind to:
   1. @data.gform.nowRecordFormBlob.recordValues.vehicle.value
   2. (You may find this by going back to the parent Record page and opening the “Glide Form” data resource from the database icon in the bottom left of the screen and looking at the “nowRecordFormBlob” JSON tab in the tray).
5. In the “currentMaintReq” property bind to “@context.props.sysId” to get the sysId from the parent record (we will use this later so that we don’t see the record that we’re currently on in the list).
6. Now that we have the vehicle’s ID it’s a simple matter of retrieving a list of past Maintenance requests!

### Show a list of previous maintenance requests

Any time you are on the parent record page you can return to the tab content by selecting “Contextual sidebar” and finding it in the “Tabs” section and clicking the subpage’s little “square” icon.

1. Under the Body in the content tree add a new “List – Simple” component with these attributes:
   1. Title: Vehicle Request History
   2. Table: Maint reqs [x\_snc\_flt\_mgmt\_maint\_req]
   3. Edit Filter:
      1. Vehicle is @context.props.vehicleId
      2. AND Sys ID is NOT @context.props.currentMaintReq
      3. Sort By: Created descending
      4. Click “Apply”
   4. Columns Selected Fields: (You must click “Add” for each field one at a time) Number, Short Description (The sidebar is very narrow, so there’s not much room)
   5. Max Rows: 20
2. Save the page
3. Return to the Record page and open the page to render it.
4. It works!
5. Wait, dude, that does NOT work.
6. The vehicle request history is blank and we know that it should at least have the record that it is being rendered from. There must be a problem with the filter.
7. Let’s debug:
   1. Go back to the subpage and go to the bottom left corner of the page and add a new client script.

Add the following two lines of script inside the “handler” function:

|  |
| --- |
| console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  console.log(event);  console.log(`vehicleId: ${api.context.props.vehicleId}`); |

* 1. Go to the List – Simple 1 control and add this script to the “Data Fetch Requested” event.
  2. Reload the page.
  3. Open the Inspector and look at the console. You should see the “\*\*\*\*\*\*” followed by “Object”, which is the event that you can open and inspect.
  4. In the “payload” property you will see that query contains “vehicle=” and no id. Looking at the nowRecordFormBlob again I realize that I’m referencing the wrong property—recordValues is an array, but I instead I should be using formFieldValues.
  5. Go back to the “vehicleId” property of the sub page’s “Body” component and change the binding to this:   
     @data.gform.nowRecordFormBlob.formFieldValues.vehicle.value
  6. Save it and try it again.

1. Now it works!
2. (Don’t forget to go delete that page script you created for debugging!)

### But *why* does it work?

How were you able to “edit” a Record page that’s not editable? The answer is that we’re not editing the shared Record page but instead are adding metadata in the form of subpages (“sys\_ux\_screen”) and UX App Routes (sys\_ux\_app\_route\_list). If you look at the routes you will see that one was created for your tab and tied to the shared “Record” macroponent and the composition element id is “contextual\_sidebar”. This tells the contextual sidebar to include that named route in your sidebar tabs for your application.

## Pointer: Keep parent and viewport pages open at the same time

Navigating back and forth from the viewport page to the parent page can be slow and for me messes with my sense of context. You can keep the parent page open in App Engine Studio and then open UI Builder in another tab and navigate down to the viewport page.

## Bubbling up events

You bind data **down** to viewports (using Body properties and “@data…”.

You **bubble up** data from viewports by emitting events.

### Use Existing Handled Events in the Parent

Next, we will add the ability to click on a row in our contextual sidebar and direct the parent workspace to open that record in a new tab. This is another example of how we can control behaviors at the parent level without actually editing the parent page.

1. Open the Vehicle Request History subpage.
2. Click on “List – Simple 1” control and under config make sure “hide links” is turned off (you will need something on the screen to bind the row click event to).
3. Add a new client script from the icon in the bottom left and call it “Open Vehicle Request”.
4. Add the following script:

|  |
| --- |
| // route: page name  // fields: mandatory parameters  // params: optional parameters  var route = {  "route": "record",  "fields": {  "table": event.payload.table,  "sysId": event.payload.sys\_id  },  "params": {}  };  api.emit('NAV\_ITEM\_SELECTED', route); |

This does pretty much exactly what it looks like it does. You’ll notice that when a tab is open that the URL lists the page name (“record” in this case), the table name and the sys id. The event is captured by the parent page which is wired up to read the route and open the new tab.

1. Return to the List – Simple 1 and the Row Clicked event and add this script as the new event handler.
2. Go back to the page and reload it NOTE: you may want to do a shift-command-R to hard refresh it because sometimes takes these changes take a second to work.
3. Click the number and a new tab will open with that record.  
   NOTE: Don’t click on the vechicle record you’re currently looking at (If you have MREQ0001006 open then click on MREQ0001005).

# Building your own tabs and modals

O.K., now that you know all about how to interact with the mostly locked down AES record template page let’s build our own page and take control. We will create a page with some tabs and open a viewport modal too.

## Why use a viewport?

We probably should have covered this before, but it’s a good question to ask. You can create a new modal or tab set and add controls directly within it from the Content tree. Why mess with viewports in the first place?

1. Reusability. Being a subpage a viewport can be rendered from different pages. E.g. if you have a common need to show a summary of a record in a modal you can do that with a viewport and by calling an event (as you will see shortly) to re-use it anywhere within your application.
2. Encapsulation. The content tree on your application may get quite large quite rapidly. Viewports are a way to break your content and layout into smaller pieces.

## Modal Viewport

### Setup a new page and add a viewport modal

1. In the top of the left-hand panel click the snowman and “Create Page” and name it “Viewports" and add one required parameter:
   1. vehicleSysId
2. In the URL at the top click the parameter and use this value:
   1. d7a83bda1b3819101363ff37dc4bcb43
3. Add a new data resource and select “Vehicle Fleet Management > Get Vehicle Information”
4. Bind the “Vehicle SysID” property to “@context.props.vehicleSysId”
5. Quickly add a Stylized Text component to show the model of the vehicle at the top of the page (@data.get\_vehicle\_information\_1.vehicle.vehicleMetadata.model)
6. Now we will add a button to show a modal with the vehicle info (it’s kind of redundant but will show how we can instantiate a modal and read values from the parent page).
7. Add a new Container to hold our buttons horizontally. Change the “Direction” to “Row”.
8. Add a new Button component called “Vehicle Info”.
9. Modals are saved in a special area in the Content tree. Next to “Content” click “Add” and choose “Modal” and “Viewport Modal”.
10. A modal will open with no content.
11. Click “Edit Content” and create a new subpage.
12. Name it “Vehicle Info Modal” and save it with no parameters.

### Binding to the parent page’s data

1. To bind to a parent’s data we will access Body level of the page. In a page property at this level you can access the parent’s context and then reference that property within the page.
2. Click on “Body” and in the Config panel next to “Properties” click “Edit”.
   1. Add two new Properties:
      1. “vehicleInfo” of type “JSON”
      2. “maintReqs” of type “JSON”
   2. Now bind the properties. At this level accessing “@” gives you access to the parent’s context as if it was the same page.
      1. “vehicleInfo”: @data.get\_vehicle\_information\_1.vehicle.vehicleInfo
      2. “maintReqs”: @data.get\_vehicle\_information\_1.vehicle.maintReqs
   3. Note how this is a nice way to reduce the dot-walking you have to do to get access to data.
3. Add a new Stylized Text of type “h3” with the text “Vehicle Info”.
4. Add another stylized text to show the name of the vehicle model and the maintenance request count. To speed this up we’ll just create one “h3” and use script to build the content.
   1. Click the “</>” on the Text attribute.
   2. Add the following script.

|  |
| --- |
| return `Maintenance Requests: ${api.context.props.maintReqs.length || "0"}`; |

1. NOTE: Since the viewport doesn’t have the context of the parent page when you are editing it you will not see values until you render the page.
2. (We won’t get into building out more details, but you get the idea of how you can access the parent’s data to do things like render a list of maintenance requests or vehicle details. You may also pass down a sysId and implement your own data resources if you like.)
3. Return to the parent page and select the button and events to add a new “Button Clicked” event.
   1. Choose Add new event handler and select “Open or close modal dialog”.
   2. Activate the “Open modal dialog” Boolean at the top of the page.
   3. Select the Modal (in this case Modal Viewport 1) and Apply.
4. Render the page and click the button.

### Setup the page to pass user input back to the parent

1. Return to the subpage and we will add an input to enter a short description for a new request and a button to return the value to the parent as well as a cancel button to close the modal.
2. Add a new Client State Parameter of type “String” to hold the value of the input called “newRequestShortDescription”.
3. Add a new Input component
   1. Label: New Request Short Description
   2. Placeholder text: e.g. Mudflaps are loose
   3. Value: @state.newRequestShortDescription
   4. New event handler: “Input value set”: Update Client State Parameter
      1. Client State Parameter: newRequestShortDescription
      2. Value: @payload.value
4. Add a new Container of direction “row” to hold the buttons.
5. Add two buttons:
   1. Create New Request
   2. Cancel:
      1. Variant: Negative Primary (to make it red)

### Dispatching event to the parent to close the modal

1. Wire up the Cancel button
   1. For the Cancel Modal event we already have a parent event to handle this we can hook into it directly.
   2. First go back to the parent page and get the Viewport Id that you used when you opened the modal from the Vehicle Info button (it will be something like “viewport\_xxx”)
   3. Go the Cancel button’s events and add a new “Button clicked” event handler.
   4. Select “Open close modal dialog”.
   5. For the modal property choose: Modal Viewport 1 and the form will expand
   6. For “Viewport ID” set the value you copied earlier (you can leave everything else alone).
2. Go back to the page, reload and it, open the modal and click Cancel to see it in action.

#### Passing values from the viewport to the parent

1. To pass values back from the viewport we’ll use events, but this time we want to configure custom data to return so it will require a bit more configuration.
2. Dispatched Events are where you define the event to fire and the data to send on the viewport while the Parent page will implement a Handled Event to capture and process that data.
3. In the modal go to Body > Events > Dispatched Events and click “Add”
   * 1. Label: “Create New Request” (note this auto fills event name “CREATE\_NEW\_REQUEST”)
     2. Advanced mode: On
     3. Add a Payload Field and save: Name: shortDescription, Label: Short Description, String
4. Go to the Create New Request button and add a new event handler.
5. Select your new “Create New Request” handler at the bottom of the Page-level event handlers section.
6. Bind the event field for short description to (you guessed it) @state.newRequestShortDescription

#### Handling event from viewport at the parent

Now that our event is dispatching data to the parent, we need to handle it and do something with it.

1. Return to the parent “Viewports” page and go to Body > Handled Events and “Add”.
2. *Give it the exact same configuration as the dispatched event* from the previous step.
   1. Label: “Create New Request” (note this auto fills event name “CREATE\_NEW\_REQUEST”)
   2. Advanced mode: On
   3. Add a Payload Field and save: Name: shortDescription, Label: Short Description, String
3. What this does is allows you to create a new Page Event Mapping on the page config. Save the page and you will now see “Create New Request” listed in the events on the Body.
4. You may use this to bind to client state parameter or call a page script.

# Links and Resources

[Creator Toolbox – Viewports YouTube video](https://www.youtube.com/watch?v=PwZLd2phUZQ) by Brad Tilton. Good examples of most use cases.