Community Network

SAP Fiori Launchpad for Developers
Navigation Concepts

Applies to:

User Interface Add-On 1.0 SPS 06 and higher

Summary

This article describes the concepts for navigation in the SAP Fiori launchpad. It covers navigation between apps (cross-app navigation) as well as navigation within a single app (inner-app navigation).

Intent-based navigation allows you to decouple the navigation triggers from the actual navigation targets, which can be flexibly configured for different roles and device types. The navigation URLs allow bookmarking, deep linking of specific app states, and going back to previous app states using the standard browser history.

Based on these concepts, the article also provides best practices and some code examples for typical development tasks.

Author(s): Gerd Forstmann, Olivier Keimel

Company: SAP SE

Created on: 04 November 2014

Author Bio

Gerd Forstmann, SAP SE

Gerd is working as a software architect in the UI Technology department, with a focus on UI integration topics.

Olivier Keimel, SAP SE

Olivier is working as a knowledge architect in the P&I Technology Engineering Services department, with a focus on UI integration topics.

Table of Contents

Overview

This article describes the concepts for navigation in the SAP Fiori launchpad.

The SAP Fiori launchpad is displayed in a web browser and offers a web-like navigation experience as average users are accustomed to. For example, it allows users to create bookmarks for specific apps and enables navigation using the web browser's standard back and forward buttons.

When an app offers links for navigating within the app or to another app, these links are regular HTML links, which means that built-in browser functions like *Open in New Tab* can be used out of the box.

The URLs displayed in the browser's address bar are human-readable and express a navigation intent that average users can relate to. Users can send the URL of launchpad apps to other users by e-mail, and if you use the capabilities of the launchpad to their full potential, different users navigating with the same URL will see different content, depending on the user roles that they are assigned to.

These capabilities have been implemented into the SAP Fiori launchpad based on the following key concepts described in this section:

- SAP Fiori launchpad architecture, with a single HTML document hosting all apps
- Navigation based on URL fragments
- Intent-based navigation

SAP Fiori Launchpad Architecture

When a user starts the SAP Fiori launchpad, the home page is displayed by default, where the user can access apps by clicking on tiles. Alternatively, a user can access SAP Fiori apps using the search function. Any kind of navigation to an SAP Fiori app is handled within the same Web app. This is because all SAP Fiori apps are hosted inside one document: *FioriLaunchpad.html*.

The *FioriLaunchpad.html* file bootstraps the unified shell (see Architecture Overview), which provides services (JavaScript APIs) common for all apps, independent from their server platform. This means that the *FioriLaunchpad.html* file is the only HTML document which is loaded by the browser, and therefore the only SAPUI5 root application. All following interactions are implemented by dynamic modifications of the hosting HTML document using JavaScript.

SAP Fiori apps are technically SAPUI5 components which are loaded into an application container. They are dynamically injected into the DOM of the *FioriLaunchpad.html* file.

For more information about the architecture of the SAP Fiori launchpad, see the Architecture Overview.

Navigation Based on URL Fragments

In the SAP Fiori launchpad, navigation is done using the fragment of a standard URL (the part beginning with the hash (#) character):

http://<server>:<port>/<path>/FioriLaunchpad.html#<fragment>

Note: In a web browser, some parts of the URL need to be encoded.

Since only one HTML page is loaded from the server, all information about the apps to be loaded is encoded in the URL fragment.

Navigation can be performed in either of the following ways:

- By setting the browser URL fragment explicitly with a JavaScript call.
- Implicitly by rendering an href-based link which contains the relative URL with the hash fragment. In this case, the hash change is performed by the browser itself when the user clicks on the link.

The unified shell services provide a listener which is registered for the hash change event and loads the app according to the information encoded in the URL hash.

Intent-based Navigation

Rather than directly encoding the name of the target app into the URL fragment, app developers provide a navigation **intent**. An intent expresses **what** you want to do next (rather than **how** to do it).

$$\text{http(s)://<server>:<port>/<path>/FioriLaunchpad.html}\underbrace{\text{\#<SemanticObject>-<Action>}}_{\text{Fragment}}$$

This intent consists of a combination of a **semantic object** and an **action**.

Example:

http://<server>:<port>/<path>/FioriLaunchpad.html#Employee-display

This approach allows you to decouple the intent, which is coded into the source app, from the actual navigation target (see the figure below). The target app can be defined using configuration, in a so-called target mapping that maps an intent to a concrete app. This approach has several benefits:

- You can develop and roll out new apps quickly, and adapt navigation targets later using configuration only.

  Example: You develop a new app from where you want to display employee data. For displaying employee data, let’s assume that you already have a Web Dynpro application available, but you are planning to replace it by an SAPUI5 application in the future.

  Thanks to the decoupling, you can develop and roll out your new app now independently from the Employee-display app. In your source app, you just code the navigation intent (rather than a concrete navigation target), and in the configuration, you map this intent to your existing Web Dynpro application.

  At a later point in time, when your new SAPUI5 application for displaying employee data is ready, you can simply map the same intent to the new SAPUI5 application, by simple configuration and without changing any line of code.

- You can start different apps depending on the device type.

  Example: If a user clicks on a link in your app in a desktop environment, you can start a desktop application that displays employee data. If the same user taps on the same link in your app on a mobile device, you can start a lightweight mobile app that displays the same data in a different way. You can do this by simple configuration, without changing any line of code in your source app.

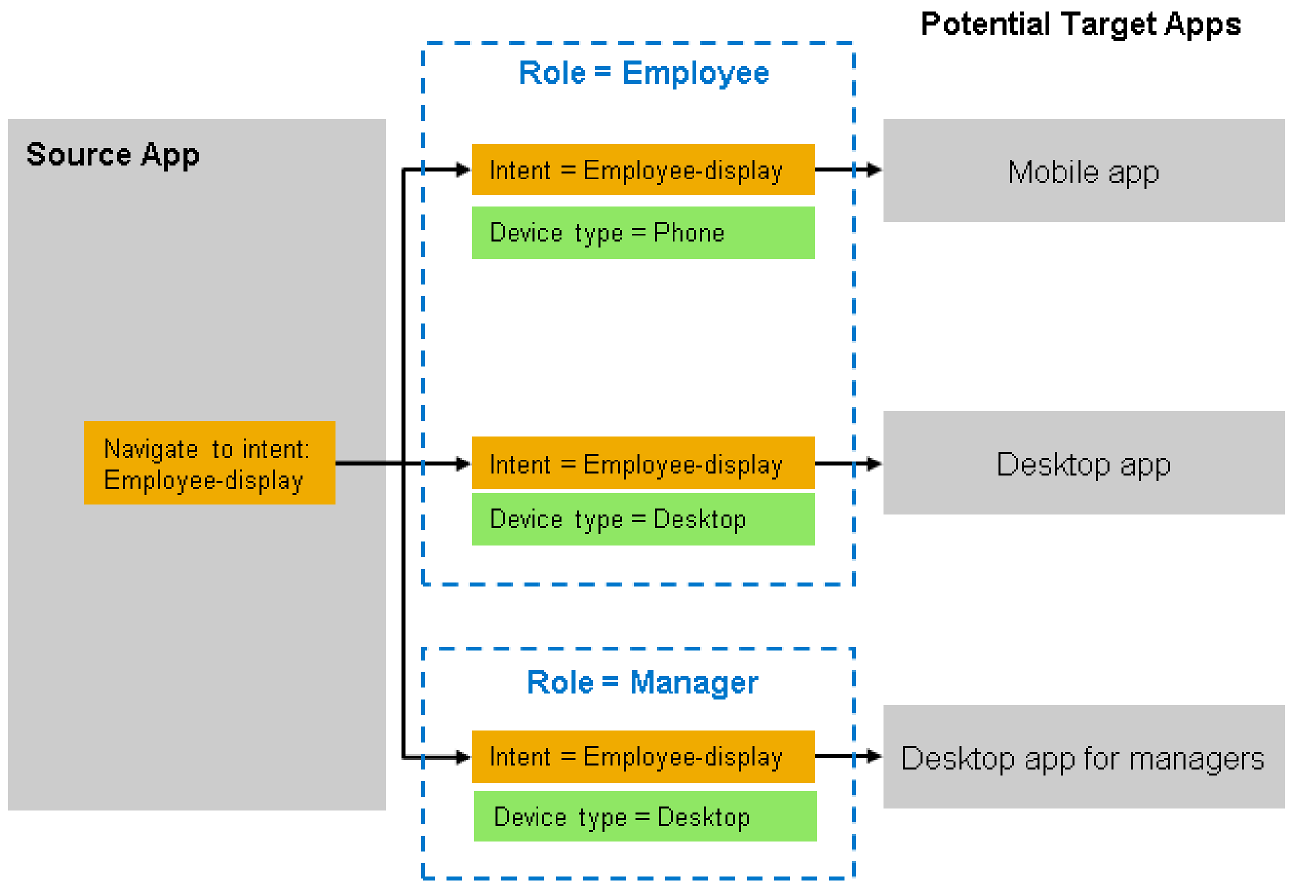
- You can even start different apps depending on the user’s role.

  Example: Let’s assume that managers in your organization should get a different view on employee data than regular employees, and you have created two views for this in your app that displays employee data. Here again, no modification is required in the code of the source app. For one intent, you can configure different navigation targets for different roles, which means that using the same URL, one of two different apps will be started depending on the user’s role.

- You can extend and customize SAP Fiori scenarios without modifying any SAP Fiori app code, just by configuration.

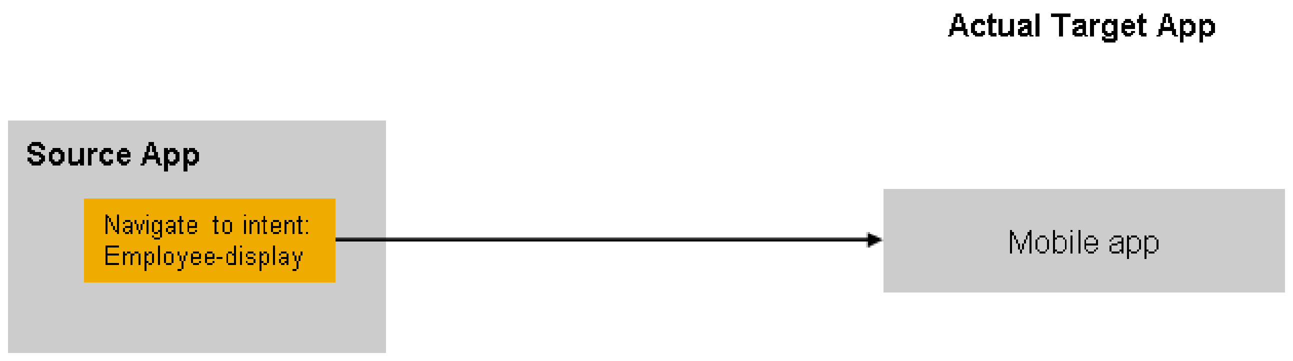
The following graphic shows an example for an intent coded into a source app. The intent expresses **what** you want to do next, on an abstract level.

Potential Navigation Targets



Depending on the device type where the navigation is triggered as well as the user's role, the same intent can be resolved to different target apps. The target apps express **how** the intent can be achieved.

Actual Navigation for an Employee using the source app on a Phone



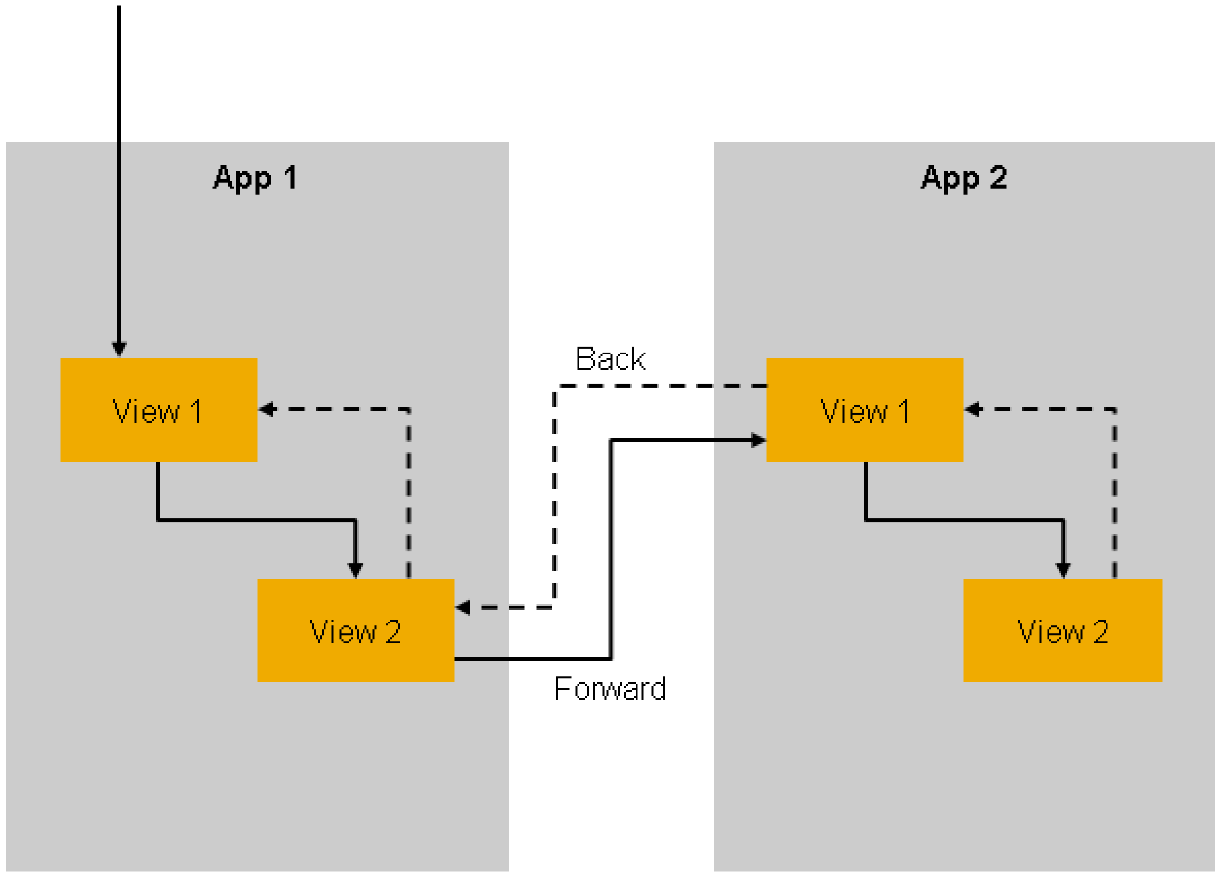
For information on how to configure navigation for different devices or roles, see [Setting up Navigation]() on SAP Help Portal.

Cross-App Navigation and Inner-App Navigation

Technically, there are 2 types of navigation operations:

- **Cross-app navigation**: Navigation from one app to another
- **Inner-app navigation**: Navigation within a single app
  This includes navigation between different views in an app, like a list view and a details view, but it also includes state changes like setting a filter in a drill-down app.

The first type of navigation is handled by the unified shell services. The second type of navigation has to be performed by the app. For both cases, the described intent-based approach is used. The hash consists of a general part which is controlled by the shell services and optionally an app-specific part which denotes the inner-app UI state. This has to be taken into account when running apps embedded in the launchpad. These apps must not directly modify the part of the URL hash that is interpreted by the unified shell services. To support you developing navigation, APIs are available both for cross-app navigation and inner-app navigation.



It is also possible to combine both types mentioned above and navigate to an app in a specific state. This happens during the back navigation from App2 to App1 in the example above. To enable back navigation to View 2 of App 1, the app needs to be able to restore the previous UI state from the app-specific information in the URL fragment, which usually contains parameters for loading the required data.

In the above illustration, all 4 states can be bookmarked, which is the main advantage of this approach. But it also implies two best practices for app developers:

- Do not clutter the browser history by adding “meaningless state” to the URL fragment.
- Be prepared to reestablish the state of an app based on the app URL fragment at any time.

Developing Navigation

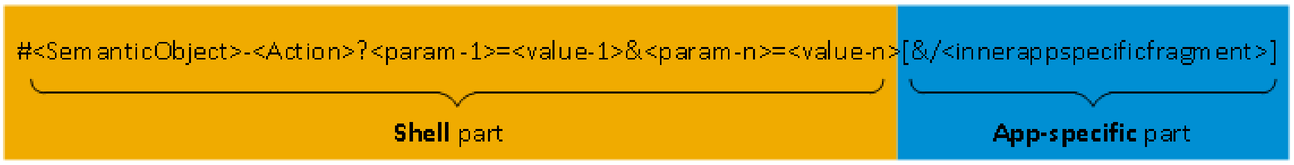
URL Syntax

A URL expresses an intent regarding the navigation target. This is something users can easily relate to, and with the help of the browser's history cache, they can easily return to their frequently-used apps, just by typing a few characters in the browser's address bar.

The complete syntax for the URL fragment looks as follows:

> *#<SemanticObject>-<Action>?<parameter-1>=<value-1>&…&<parameter-n>=<value-n>[&/<innerappspecificfragment>]*

The URL fragment has a shell part and an app-specific part:


#<SemanticObject>-<Action>?<param-1>=<value-1>&<param-n>=<value-n>[&/<innerappspecificfragment>]
Shell part
App-specific part

The **shell** part of the URL fragment is used for cross-app navigation. It is independent from the concrete implementation of an app.

The **app-specific** part of the URL is dependent from the concrete implementation of an app.

Note: You might come across URLs with the following pattern:

> *#<SemanticObject>-<Action>[~<InternalTargetID>]?<parameter-1>=<value-1>&…&<parameter-n>=<value-n>[&/<innerappspecificfragment>]*

The <InternalTargetID> (the part of the URL after the tilde (~) character) is a hash value which is set by the navigation target resolution service. While the combination of a semantic object and an action specify a navigation intent, the unique target ID identifies a specific target app. This part of the URL is usually not shown to end users and should not be touched by developers.

Cross-App Navigation

The following example shows a link for cross-app navigation:

> *<a href="#DaysSalesOutstanding-drilldown?kpiId=1234&amp;variantId=abcd" target="_self">Details</a>*

To make sure that the URL format is always correct, use the **CrossApplicationNavigation service** to construct such links. For more information, see the JSDocs.

Inner-App Navigation

The SAPUI5 core library provides an API for navigating with URL fragment changes. When an app is running within the SAP Fiori Launchpad, the implementation will consider the shell-specific navigation parts and allow both inner-app and app-to-app navigation by similar means.

The **navigation API** provides the following functionality:

- Directly trigger navigation to a parameterized navigation target
- Register navigation routes and listeners for fragment changes to restore a specific state

Apps can encode their inner state into the URL, which allows bookmarking not only the entry points into an app, but also a certain state or context within the app.

For more information, see Navigation in the SAPUI5 Developer Guide.

Best Practices for Navigation

The following best practices will help you develop navigation between your SAP Fiori apps:

- Use the CrossApplicationNavigation service

  When developing cross-app navigation, use the CrossApplicationNavigation service to construct shell fragments and links from local fragments. Do not construct them yourself.
- When developing cross-app navigation, use startup parameters to pass information to the next app.
- In the launchpad intent, use single-value parameters only.

  Multi-valued parameters are not recommended.
- Do not use deep links to the app-specific fragment for passing information to the next app.

How to Pass Startup Parameters to an SAPUI5 Fiori App

Startup parameters are transferred to a target app by encoding them in the URL.

Note: Be aware that this data is part of the URL, thus it is stored in the browser history, which might be a security or data protection issue. If security-critical content is to be passed, use an anonymized format and keep the sensitive data within your back-end application

Note: The length of a browser URL is limited and truncation may occur. Keep the length of the URL fragment below 512 characters.

Note: Information is transferred to the front-end server as part of navigation target resolution. This information may be persisted on the front-end server.

Passing Startup Parameters Dynamically with JavaScript

An app can dynamically pass parameters to another app using the CrossApplicationNavigation service, as shown in the following code example:

```
var href_For_Product_display = ( sap.ushell && sap.ushell.Container &&
sap.ushell.Container.getService("CrossApplicationNavigation").hrefForExternal({
   target : { semanticObject : "Product", action : "display" },
   params : { "ProductID" : "102343333", SupplierId : "90210" }
})) || "";
```

Receiving Startup Parameters Dynamically with JavaScript

Startup parameters are received by an embedded Component via the *componentData* property, member *startupParameters*, as shown in the following code example:

```
sap.ui.core.UIComponent.extend("AppNavSample.Component", {
   ...
   createContent : function() {
      /* contains e.g. {  startupParameters : { AAA : ["BBB"], DEF: ["HIJ","KLM"] }  } */
      /* note that parameter values are passed as an array!
      var oComponentData = this.getComponentData();
      jQuery.sap.log("app was started with parameters " +
JSON.stringify(oComponentData.startupParameters || {} ));
   ...
   }
});
```

Note that the *componentData* property always contains **arrays** with the parameter values.

Consuming Startup Parameters from Inside an Embedded View

For views created by the routing framework of SAPUI5 or directly from within the component, you can locate and identify the component using *getOwnerIdFor*, as shown in the following example:

```
// view controller.
   getMyComponent: function() {
      "use strict";
      var sComponentId = sap.ui.core.Component.getOwnerIdFor(this.getView());
      return sap.ui.component(sComponentId);
   }
   onCreate : function() {
      …
      var oStartupParameters = this.getMyComponent().getComponentData().startupParameters;
   }
…
```

For more information, see the JSDocs.

Other Sources of Startup Parameters

Startup parameters for an app can be set the following ways:

- Dynamically pass them from the source app as described above.
- Set them in the configuration of a navigation target in transaction LPD_CUST (see Customizing Navigation Targets).
- Define them as startup parameters in configuration (target mapping).
- The user enters them in the URL.

Note: Be aware that startup parameters are transparent to the end user in the URL and may be tampered with or serialized with minimal efforts.

Passing Query Parameters in a Standalone Index Page

If you use a standalone index page for testing your app in a local environment, you can pass the query parameters to the component as shown in the following example:

```
<head>
…
<script>
   var oStartupParameters = jQuery.sap.getUriParameters().mParams;
   var oComponent = sap.ui.getCore().createComponent({
      name: "sap.samples.SampleComponent",
      settings: {
         componentData: { startupParameters: oStartupParameters }}
   });
   new sap.ui.core.ComponentContainer({
      component: oComponent
   }).placeAt("content");

</script>
</head>
<body>
   <div id=”content”/>
</body>
```

Related Content

SAP Fiori Launchpad for Developers – Architecture Overview

Setting up Navigation

JSDocs for the CrossApplicationNavigation service

Navigation section in the SAPUI5 Developer Guide

Copyright