UI5 Reuse Library

Generic search help

Table of Contents

[1 Introduction: 3](#_Toc27141125)

[2 How to install in your app : 5](#_Toc27141126)

[2.1 Manifest dependencies. 5](#_Toc27141127)

[2.2 Declare the library. 6](#_Toc27141128)

[2.2.1 Define the library helper. 6](#_Toc27141129)

[2.3 Start the service with specific settings 7](#_Toc27141130)

[3 How to use in View. 8](#_Toc27141131)

[3.1 For list with more than 200 entries. 8](#_Toc27141132)

[3.1.1 Dependency set up. 9](#_Toc27141133)

[3.2 For list with less than 200 entries. 11](#_Toc27141134)

[4 Additional features. 12](#_Toc27141135)

[4.1 Call search help function 12](#_Toc27141136)

[4.2 Promises. 13](#_Toc27141137)

[4.2.1 Component loaded promise. 13](#_Toc27141138)

[4.2.2 Local list loaded promise. 13](#_Toc27141139)

[4.3 Events. 14](#_Toc27141140)

[4.3.1 itemSelected. 14](#_Toc27141141)

[4.3.2 dataLoaded. 14](#_Toc27141142)

[4.3.3 dataError. 14](#_Toc27141143)

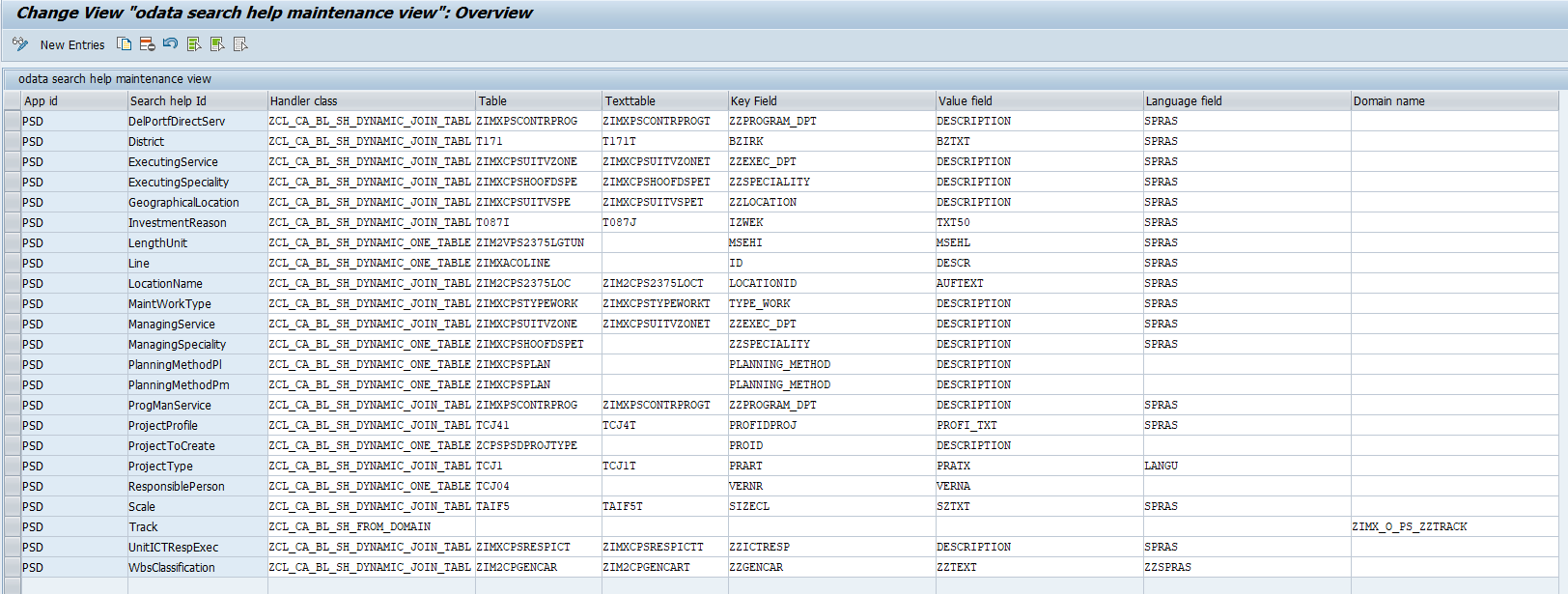
# Introduction:

The reuse generic search help library offers a simplified way to use combo box, select and input field with suggestion/search help in view.

It manages :

* The data loading (in local storage and/ in backend linked ).
* The search help popup generation and processing.
* The suggestion item list generation and processing.
* Dependency ( filtering of search help Vs another input field vlue ) is managed too.

The backend search engine is made through a generic SQl request based on data entered in maintenance view ZV\_CASEARCHHELP. In this view, per Application Id, we can define different list. It can be a simple table selection, or a combination between table and text table or domain value or even a call to a provider class ( for more complex selection ).



There are two modes of loading. If list is tiny and used in combo box/select (+/-200 entries), the whole list is loaded from the backend and store in a local JSON model. If the list is bigger ( > 200) it is managed as for an usual oData model (data loaded by interval / search and filter in the backend ).

The reusable component builds and handles the search help fragment. All corresponding function (close/search) is also fully managed by the reusable component.

Initialization, suggestion process and change on an input is processed fully by the reusable component. No additional function is needed in the application controller.

A Demo application that uses the reusable component is available in GIT under <https://git/EXA326/shdemoapp/tree/master>

In Fiori Dev, under AGOR group, a demo application is available too



Source of library available at <https://git/sapfiori/CrossApplication/reusesearchhelplibrary>

# How to install in your app :

It consists to :

* Add dependence in the manifest to load the library
* Declare the library.
* Start the service with specific settings.
* Use the enhanced Input, ComboBox and Select controls from the library.

## Manifest dependencies.

        "dependencies": {

            "minUI5Version": "1.30.0",

            "libs": {

                "sap.ui.core": {},

                "sap.m": {},

                "sap.ui.layout": {},

                "sap.ushell": {},

                "sap.ui.comp": {},

                "sap.f": {},

                "be.infrabel.reuse.cross": {}

            }

        },

**...**

## Declare the library.

Library is declared preferably in the component.

### Define the library helper.

In the component properties, you might define two properties to store the helper instance and the library component instance for later use. This is optional, since it exists functions to retrieve these instances anywhere.

sap.ui.define([

    "sap/ui/core/UIComponent",

    "be/infrabel/reuse/cross/genericsearch/Helper"

], function (UIComponent, InputHelper) {

    "use strict";

    var oComponent = UIComponent.extend("be.infrabel.genericshdemo.Component" /\*\*@lends be.infrabel.genericshdemo.controllers.Component.prototype \*\*/ , {

        metadata: {

            manifest: "json",

            properties: {

                valueHelpComponent: {

                    type: "be.infrabel.reuse.cross.genericsearch.Component"

                },

                inputHelper: {

                    type: "be.infrabel.reuse.cross.genericsearch.Helper"

                }

            }

        }

    });

## Start the service with specific settings

The service is started preferably in the component Init.

1. Create an instance of the library helper with input setting
2. Start the service with function helper.start :
   1. Library component is instanced
   2. Local data is loaded
   3. The function return a promise that is resolved when the library component is instanced.

//instance the input helper service

this.setInputHelper(new InputHelper({

            appId: "PSD",

            compId: "be.infrabel.genericshdemo",

            listToLoad: [

                "GeographicalLocation",

                "LengthUnit",

                "District",

                "Track"

            ]

        }));

 //start the input helper service

 this.getInputHelper().start()

            .then(function (oValueHelpComponent) {

                // store the value help component for a later usage

                this.setValueHelpComponent(oValueHelpComponent);

            }.bind(this))

            .catch(function (oError) {

                MessageToast.show(oError.message);

            });

Input Settings:

* appId: the application Id for the backend customising ( ex : PSD )
* compId: id for the library component that will be created at service start.
  + If the library is used with only one instance, this id can be blank.
  + If several instances of the library are generated in the application ( with different input settings for ex ) then compId is required to distinguish tools instances
  + If the application calls or is called by others applications that uses also the tool ( cross navigation and/or component usage ), it is strongly advised to fulfil the compId setting to avoid any disturbances between service instances.
* listToLoad 🡪 list of search help id that must be loaded locally (preferably small list < 200 )

Don’t forget the destroyer function to clean the library component. The system will also deregister the service instance stored statically in the helper function.

    oComponent.prototype.destroy = function () {

        if (this.getValueHelpComponent()) {

            this.getValueHelpComponent().destroy();

        }

        UIComponent.prototype.destroy.apply(this, arguments);

    };

# How to use in View.

Library offers 3 extended controls base on sap.m lbrary :

* Input : extended from sap.m.Input
* Combo : extended from sap.m.Combo
* Select : extended from sap.m.Select

To use these control, Add the *be.infrabel.reuse.cross.genericsearch* library in the view dependencies

<core:View xmlns:core="sap.ui.core"  xmlns="sap.m"

    xmlns:searchHelper="be.infrabel.reuse.cross.genericsearch"

    controllerName="be.infrabel.genericshdemo.controller.main">

## For list with more than 200 entries.

Use preferably the extended input control with a non-preloaded list.

 <Label text="{i18n>ControllingArea}" />

 <searchHelper:Input type="Text"

                     showValueHelp="true"

                     showSuggestion="true"

                     selectedKey="{viewModel>/ControllingArea}"

compId="be.infrabel.genericshdemo"

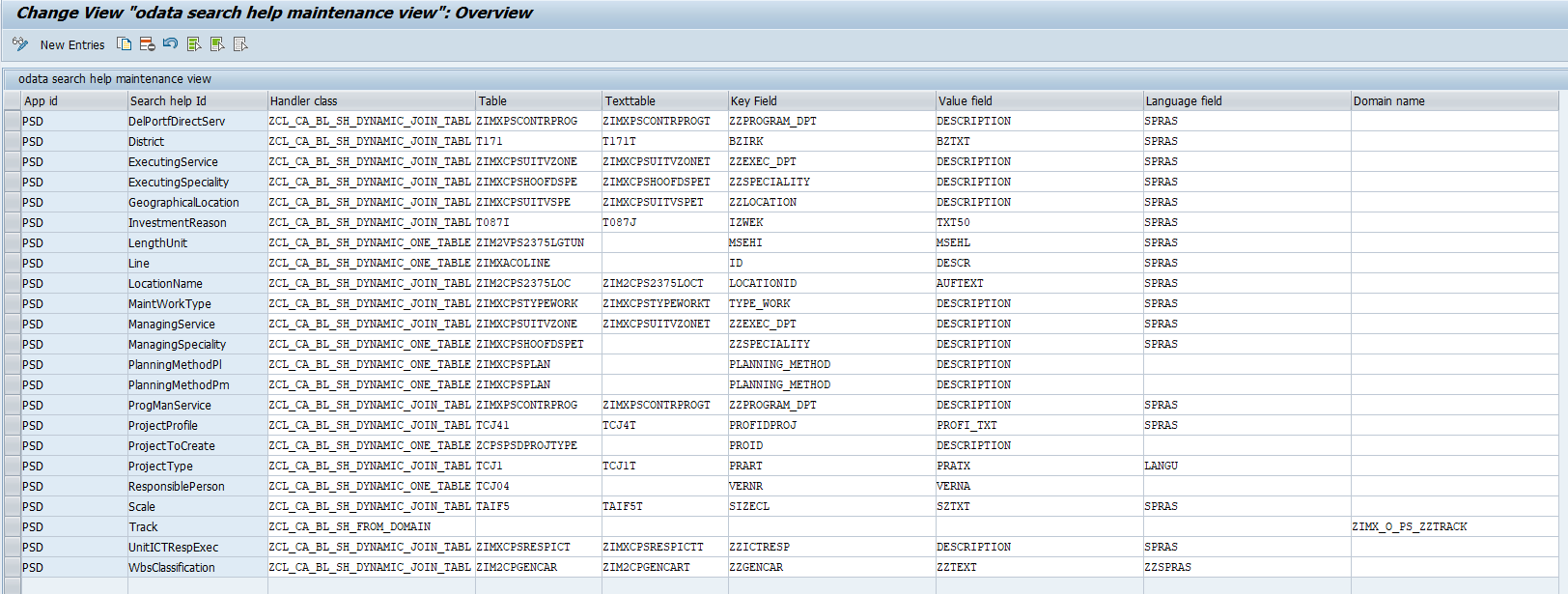
                     searchHelpId="ControllingArea"

                     displayMode="both"

                     listWithKey="true"/>

The extended properties for the input are:

* compId (optional) : id of the service instance ( defined at service start ). Can be ignored if id has been ignored at service start.
* searchHelpId (required)🡪 the list id to use with this input



* displayMode (optional) 🡪 determine how the selection is displayed in the input field
  + displayMode = ‘value’ : only the description of the selected item is shown in the field
  + displayMode = ‘key’ : only the key of the selected item is shown in the field
  + displayMode = ‘both’ (default) : key + description are shown in the field
* listWithKey (optional) 🡪 define if the key of the entry must be shown in the serach help list
  + true : the key is displayed in the search help result list .
  + False (default) : only the description is displayed in the search help result list .
* searchTitle (optional) 🡪 define the title to use for the search help popup.
  + If not given, the text of the label attached to the input is taken as search help title.
  + Note : if input is used in a form or simple form, the label is automatically linked with the input control, otherwise the label must be linked to the input control through ‘labelFor’ association.

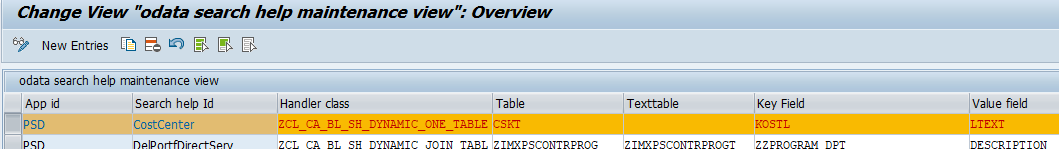
### Dependency set up.

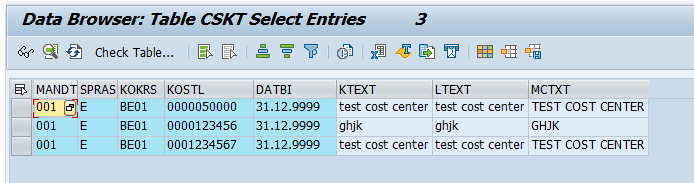
With the extended input, it is possible to set up a dependency to filter search help result.

Ex. : an input field for cost center is filtered by the controlling area.

To achieve this dependency, there are 3 specific extended properties:

* dependencyKey 🡪 the value that define the dependence ( in our example the controlling area key value ).
* dependencyField 🡪 the ABAP field in the backend table where apply the dependence. (in our example search help id CostCenter is used and "KOKRS" is the field that hold the dependence in the used cost center table CSKT ).





* dependencyRequired (optional) 🡪 is dependency is mandatory
  + true: an error message is displayed if the user try to get search help on the input and the dependence key is blank.
  + False (default) : the dependence key is optional. If blank, the result list of the search help is not filtered by the dependence.

<searchHelper:Input type="Text"

                    showValueHelp="true"

                    showSuggestion="true"

                    selectedKey="{viewModel>/CostCenter}"

compId="be.infrabel.genericshdemo"

                    searchHelpId="CostCenter"

                    displayMode="both"

                    listWithKey="true"

                    dependencyKey="{viewModel>/ControllingArea}"

                    dependencyField="KOKRS"

                    dependencyRequired="true"/>

## For list with less than 200 entries.

You can use a preloaded list and use the extended combobox or select control.

<searchHelper:ComboBox selectedKey="{main>Track}"

compId="be.infrabel.genericshdemo"

                       searchHelpId="Track"

                       displayMode="value"/>

<searchHelper:Select forceSelection="false"

                     selectedKey="{main>BkUnit}"

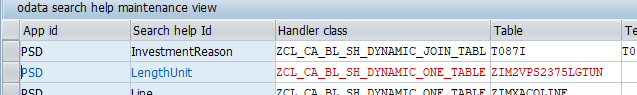
compId="be.infrabel.genericshdemo"

                     searchHelpId="LengthUnit"

                     displayMode="value"/>

The extended properties for the input are:

* compId (optional) : id of the service instance ( defined at service start ). Can be ignored if id has been ignored at service start.
* searchHelpId (required)🡪 the list id to use with this input



Note : since control should use preloaded list, the list id should also be declared at service instantiation in component ( [see 2.3](#_Start_the_service) ):

this.setInputHelper(new InputHelper({

            appId: "PSD",

            compId: "be.infrabel.genericshdemo",

            listToLoad: [

                "GeographicalLocation",

                "LengthUnit",

                "District",

                "Track"

            ]

        }));

* displayMode (optional) 🡪 determine how the suggestion list is build and how selection is displayed in the field .
  + displayMode = ‘value’ : only the description is shown in suggestion and in control
  + displayMode = ‘key’ : only the key is shown in suggestion and in control
  + displayMode = ‘both’ (default) : key + description are shown in suggestion and in control.

# Additional features.

Some additional feature are available in the service:

## Call search help function

In the library component, the function *callValueHelp* build a search help based on a list Id

The function return a promise where the resolve function return the popup control instance.

The selected item is given back through event *itemSelected* (see [4.3.1](#_itemSelected.)).

Here an example of implementation:

…to get the search help instance :

this.getValueHelpComponent().callValueHelp(this.getText("District"), "District", true)

.then(function (oValueHelp) {

                        oValueHelp.open();

                     }.bind(this));

…to get the selected item in the search help :

this.getValueHelpComponent().attachItemSelected(function (oEvent) {

                        const oSelectedItem = oEvent.getParameter("selectedItem");

                        const newGeoLoc = Merge({}, this.getGeoDataTemplate());

                        newGeoLoc.Key = oSelectedItem.key;

                        newGeoLoc.Description = oSelectedItem.value;

…

                    }.bind(this));

getValueHelpComponent *() is a getter from application component to get back the library component instance created at component init (see* [*2.3*](#_Start_the_service) *)*

## Promises.

Since a lot of initialization are made asynchronously when the service start, some promise are available to trigger processes in the calling application at the right time.

### Component loaded promise.

In the search help helper, a promise is resolved when the library component is loaded.

The promise is *isLoaded.* The resolve function return the library component instance.

There is an input parameter to give the component Id for which we want the promise.

Here an example of implementation:

this.getOwnerComponent().getInputHelper().isLoaded("be.infrabel.genericshdemo")

        .then(function(oComponent){

…

        }.bind(this));

*getInputHelper() is a getter from application component to get back the search help instance created at component init (see* [*2.3*](#_Start_the_service) *)*

### Local list loaded promise.

In the library component, a promise is resolved when the local list are loaded in the library component.

The promise is *isListLoaded.* The resolve function return the JSON model containing the loaded lists.

## Events.

The library component triggers 3 events:

### itemSelected.

This event is triggered when user has selected an item in the search help popup of the extended input control. It give back the Input control instance that generate the event and the selected value.

Here an example of implementation in a view controller :

this.getOwnerComponent().getInputHelper().isLoaded("be.infrabel.genericshdemo")

        .then(function(oComponent){

            oComponent.attachItemSelected(function (oEvent) {

                const oParameters = oEvent.getParameters();

…

            }.bind(this))

        }.bind(this));

*getInputHelper() is a getter from application component to get back the search help instance created at component init (see* [*2.3*](#_Start_the_service) *)*

### dataLoaded.

This event is triggered when local list are loaded in the library component. It give back the JSON model containing the loaded lists. If no local list have been defined, the event is triggered with an empty list.

### dataError.

This event is triggered when local list loading has failed. It give back the error message.