

SAP Fiori Mobile (v2.4.x)

Introduction

What are hybrid mobile apps?

Hybrid mobile apps are web applications wrapped up their own browser which can run on any of the mobile platforms (Android, iOS, Windows etc.). The big advantage of a hybrid app is that it has a single code base, whereas a native app must be re-written for each platform.

Apache [Cordova](#) is an open-source framework for developing hybrid apps. To use native features (beyond those already available via hmtl5,) apps can make use of plugins. These plugins handle the platform-specific code and are available for single or multiple platforms. Plugins are available to access the GPS, accelerometer or camera, for example.

SAP have written their own plugins, branded as [Kapsel](#), to handle things like logon and offline OData. These bring an enterprise flavour to hybrid apps.

What is Fiori Mobile?

Until recently SAP's recommended approach was to use the Hybrid Application Toolkit (or HAT) in order to build hybrid apps. The app build took place on the developer's PC or Mac. The HAT gained a reputation for being tricky to set up and that hindered the use of hybrid apps within the SAP ecosystem.

Fiori Mobile has at its core a cloud-build service, so there is no need to install the HAT. We can trigger the build from Web IDE. We specify the (already deployed) Fiori app(s) that we want to package and after a few minutes we can download the .apk (for Android) and .ipa (for iOS) files ready to be installed on mobile devices.

Does it work?

The cloud build service works well. It typically build for both platforms in about 8 minutes without issues. There have only been a couple of occasions when builds failed due to technical problems.

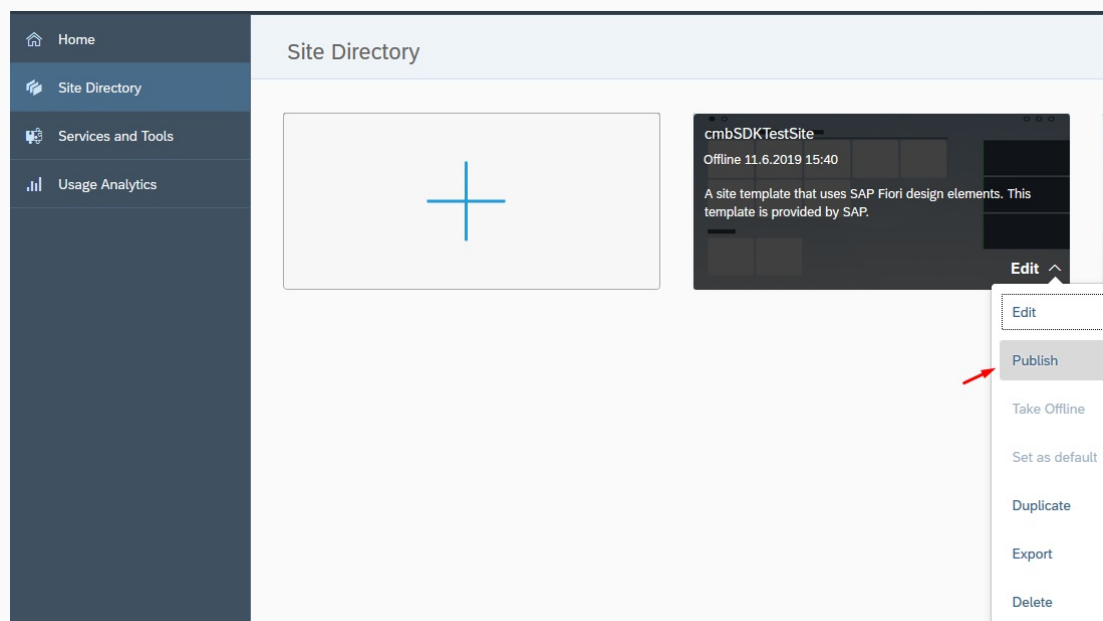
There are some facets to the technology which don't feel mature yet. One area is the offline OData features. When using Fiori Mobile things work differently to the more conventional hybrid and native apps. One example would be the destinations, because Fiori Mobile apps use a generated destination which reference the SAP Cloud Platform (CP) portal service.

My point is not that offline OData seems immature, rather that the offline features can be tricky when used with Fiori Mobile apps. We are still receiving assistance from SAP to get the offline features of our app optimised. This is a shame, because shared data and the handling of delta requests, for example, are two features which justify the 'middleware' approach to offline that SAP have taken.

Authentication has also been a challenge. On a previous project I used the Fiori Client in conjunction with SAP Cloud Identity (now officially SAP Cloud Platform Identity Authentication). The Fiori Client stored the username and password on the device so that the user didn't need to enter them every time their session timed out. We haven't achieved that yet with our Fiori Mobile app.

Account and Services

- Sign up for a free account for the SAP Hana Cloud Platform, and activate your account from your mail activation link that you receive.
- Once you have activated your account browse on HCP Cockpit.
- Choose Trial Region and open your Personal Trial Account.



SAP Cloud Platform Cockpit

Overview

Applications >

Services

Solutions

SAP HANA / SAP ASE >

Connectivity >

Security >

Repositories >

Usage Analytics

Members


Platform Roles

Useful Links

Internet of Things

Remote Data Sync


Enabled

 Synchronize data between remote databases and a cloud SAP HANA database.

Mobile

Mobile Services, preview


Enabled


 Run integration and regression tests and explore new mobile features. NOT FOR PRODUCTIVE USE.

Runtimes & Containers

HTML5 Applications


Enabled

 Develop and run HTML5 applications in a cloud environment.

User Experience 

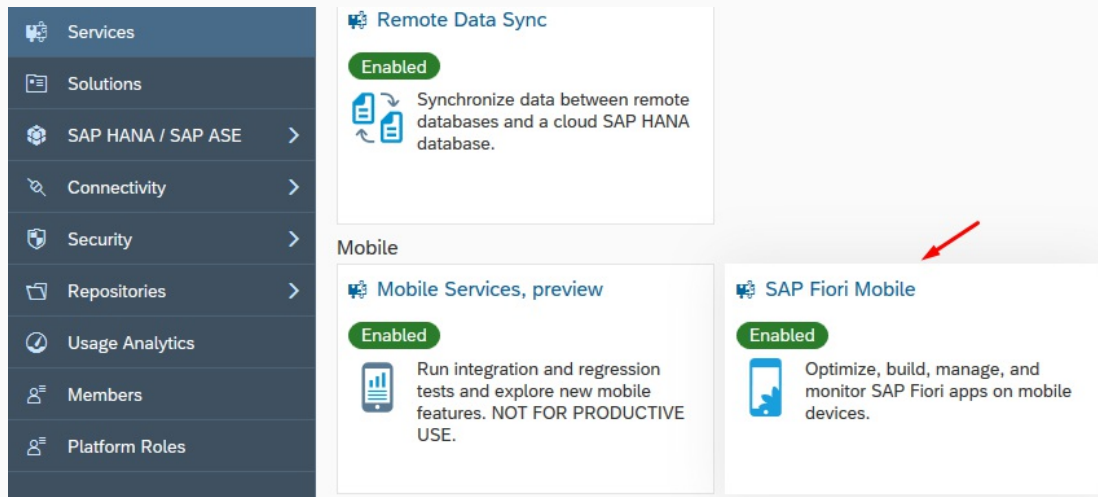
Portal

Enabled

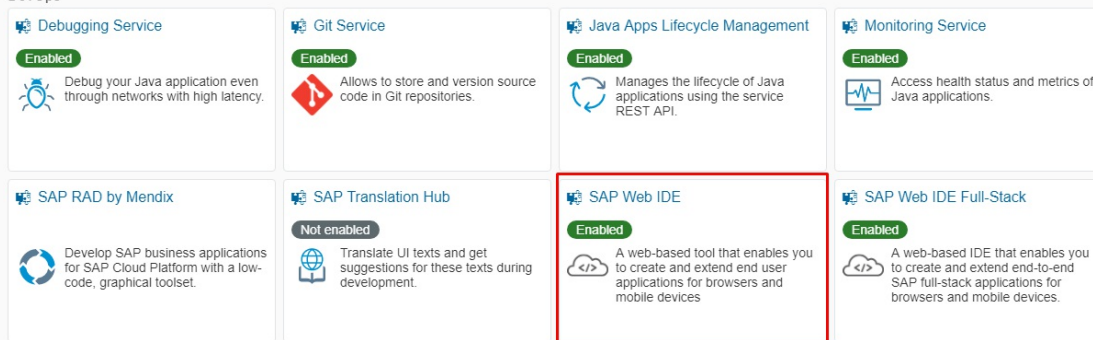
 Create role based, multi-channel sites to access business apps and content.

To create Fiori Mobile Application you will need to use **Portal**, **SAP Web IDE** and **Fiori Mobile** services.

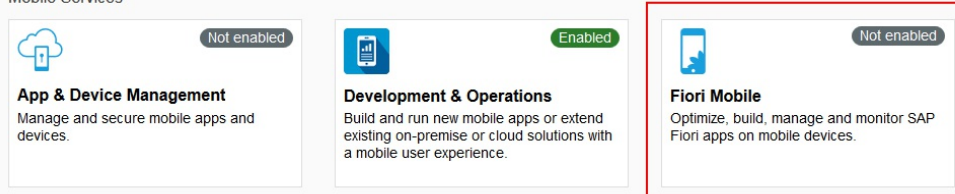
Go in Services tab find these services and enable them if they are disabled.



DevOps



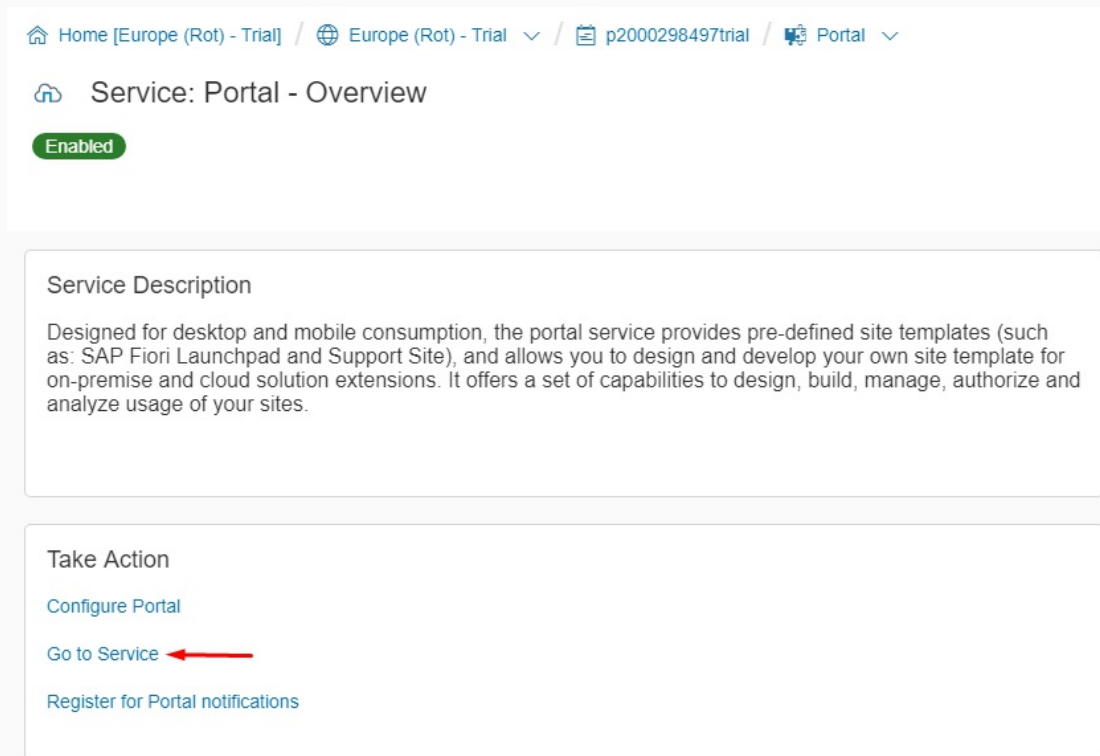
Mobile Services



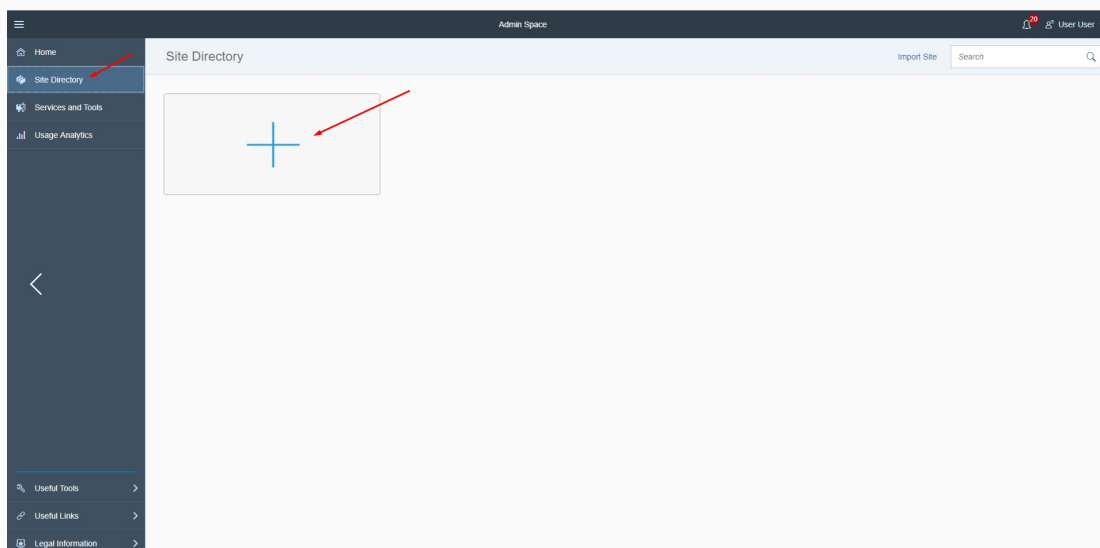
Create Fiori Launchpad

To create Fiori Application we need to create and publish the Site where application will be deployed.

Go to **Portal Service** and click Go to Service.



In your browser a new tab will be opened. In this new window go to the **Site Directory** tab, and click on Create Site.



A popup will be displayed. Enter your **Site Name**, choose **SAP Fiori Launchpad** as Template Source and click Create

Create Site

*Site Name

FioriLaunchpad

Available Site Templates

Search

Q

Template Source

All

Starter Site

Basic Layout Set

Admin Site

SAP Fiori Launchpad

Create

Close

Following this you will be redirected to SAP Fiori Configuration Cockpit. Close the tab and go back to Admin Space for Portal Service.

Now we need to Publish this Site and Set as default.

Site Directory

+

FioriLaunchpad

Offline 19.4.2018 15:53

A site template that uses SAP Fiori design elements. This template is provided by SAP.

Edit ^

Edit

Publish

Take Offline

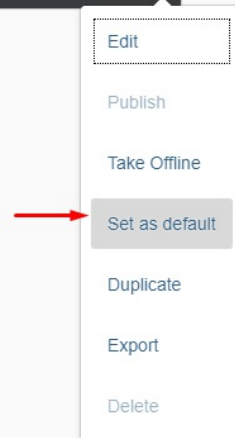
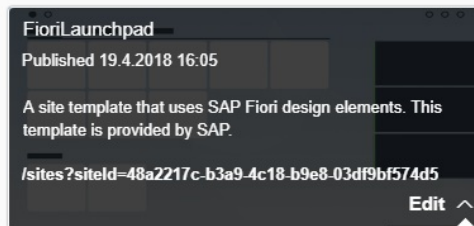
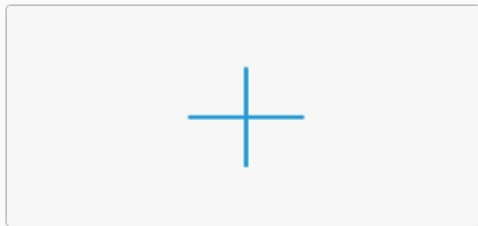
Set as default

Duplicate

Export

Delete

Site Directory



Create Fiori Project

Once we create and publish the Site we can create our Fiori project.

Go to Services tab again, open SAP Web IDE service, and click on Go to Service

Service: SAP Web IDE - Overview

Enabled

Service Description

SAP Web IDE is a web-based tool that enables you to create and extend end user applications for browser and mobile devices. It simplifies the end-to-end application lifecycle: prototyping, development, packaging, deployment, and customer extensions for SAPUI5 and SAP Fiori applications, and allows developers to collaborate with business experts and designers to fulfill end user requirements and expectations more effectively.

Take Action

[Configure Service](#)

[Logs](#)

[Go to Service](#)

A new tab will be opened. From here we will start to create our Fiori project.

- Click on New Project from Template

General Information | Tile Configuration | Assignment | Confirmation

Register to SAP Fiori Launchpad

General Information

*Provider Account

*Application Name

Description

Intent

<input type="checkbox"/>	Semantic Object	Action
<input type="checkbox"/>	<input type="text" value="MyFirstFioriApp"/>	<input type="text" value="Display"/>

[More information on intent properties.](#)

- Choose SAPUI5 Application from Template Selection

General Information **Tile Configuration** Assignment Confirmation

Register to SAP Fiori Launchpad

Tile Configuration

*Type	Static
*Title	cmbSDK Fiori App
Subtitle	Sample
Icon	sap-icon://approvals

- Set your Project Name

General Information Tile Configuration **Assignment** Confirmation

Register to SAP Fiori Launchpad

Assignment

*Site	cmbSDKTestSite
*Catalog	Sample Catalog
*Group	Sample Group

- Choose the View Type and set the View Name. In our sample we will use XML View. Click Finish and your Fiori project is created.

Template Selection Basic Information **Template Customization** Confirmation

New SAPUI5 Application

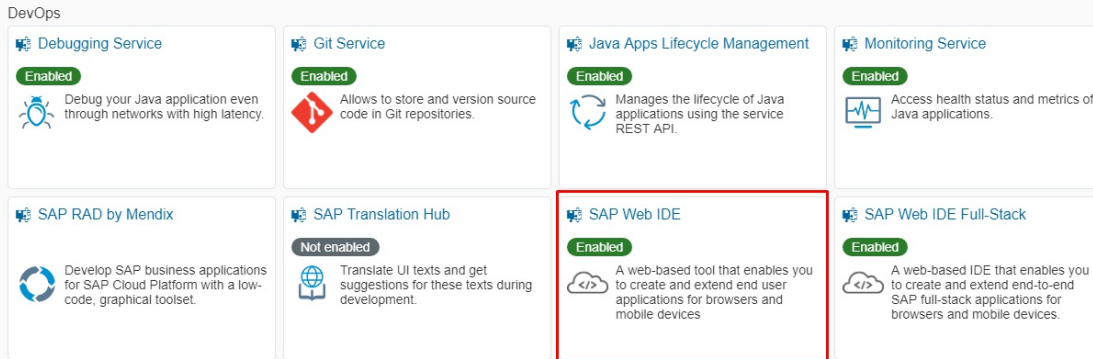
Template Customization

Initial View Details

View Type*	XML
View Name	ScanView

How to use cmbSDK cordova plugin

In the SAP Web IDE Workspace we can see all files that our Fiori project includes.



Open *Component.js* file.

Component.js is the first point of our application, we can say that it serves as index which encapsulates all our applications details, i.e. view names, routing details, main view, applications type(Full Screen or SplitApp), application service configuration etc..

Here in **init** function we will configure our plugin. Set properties that we need, set callback functions, call some methods, etc..

```
sap.ui.define([
    "sap/ui/core/UIComponent",
    "sap/ui/Device",
    "MyFristFioriApp/model/models"
], function(UIComponent, Device, models) {
    "use strict";

    var oEventBus = sap.ui.getCore().getEventBus();
    var scannerIsInitialized = false;

    return UIComponent.extend("MyFristFioriApp.Component", {

        metadata: {
            manifest: "json"
        },

        /**
         * The component is initialized by UI5 automatically during the startup of the app and calls the i
         * nit method once.
         * @public
         * @override
         */
        init: function() {
            // call the base component's init function
            UIComponent.prototype.init.apply(this, arguments);

            // set the device model
            this.setModel(models.createDeviceModel(), "device");
        }
    });
});
```

```

if (!scannerIsInitialized) {

    scannerIsInitialized = true;

    window.readerConnected = 0;
    window.scannerActive = false;

    cmbScanner.addOnResume(function(result) {

        cmbScanner.setAvailabilityCallback((readerAvailability) => {
            if (readerAvailability === cmbScanner.CONSTANTS.AVAILABILITY_AVAILABLE) {
                oEventBus.publish("ScanView", "SetConnectionStatus", {
                    statusText: "AVAILABLE"
                });
            }

            cmbScanner.connect((result) => {

            });

        } else {
            oEventBus.publish("ScanView", "SetConnectionStatus", {
                statusText: "NOT AVAILABLE"
            });
        }
    });

    if (Device.os.android) {
        cmbScanner.connect((result) => {

        });
    }
});

cmbScanner.addOnPause(function(result) {

    if (Device.os.android) {
        cmbScanner.disconnect();
    }
    cmbScanner.setAvailabilityCallback();
});

cmbScanner.setPreviewContainerPositionAndSize(0, 0, 100, 50);

cmbScanner.setConnectionStateDidChangeOfReaderCallback((connectionState) => {

    if (connectionState === cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED) {

        oEventBus.publish("ScanView", "SetConnectionStatus", {
            statusText: "CONNECTED"
        });
    }

    if (window.readerConnected !== connectionState) {
        cmbScanner.setSymbologyEnabled("SYMBOL.DATAMATRIX", true);
        cmbScanner.setSymbologyEnabled("SYMBOL.C128", true);
        cmbScanner.sendCommand("SET TRIGGER.TYPE 2");
    }

    var sdkKEY = "";
    if (Device.os.android)
        sdkKEY = this.getModel("i18n").getProperty("MX_MOBILE_LICENSE_ANDROID");
    else

```

```

        sdkKEY = this.getModel("i18n").getProperty("MX_MOBILE_LICENSE_ios");

cmbScanner.registerSDK(sdkKEY, (res) => {
    switch (res) {
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_OK:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_KEY:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_CHECKSUM:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_APPLICATION:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_SDK_VERSION:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_KEY_VERSION:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_PLATFORM:

            break;
        case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_KEY_EXPIRED:

            break;
        default:
            break;
    }
});

} else if (connectionState == cmbScanner.CONSTANTS.CONNECTION_STATE_DISCONNECTED) {
    oEventBus.publish("ScanView", "SetConnectionStatus", {
        statusText: "DISCONNECTED"
    });
} else if (connectionState == cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTING) {
    oEventBus.publish("ScanView", "SetConnectionStatus", {
        statusText: "CONNECTING"
    });
} else if (connectionState == cmbScanner.CONSTANTS.CONNECTION_STATE_DISCONNECTING) {
    oEventBus.publish("ScanView", "SetConnectionStatus", {
        statusText: "DISCONNECTING"
    });
}

window.readerConnected = connectionState;
});

cmbScanner.setAvailabilityCallback((readerAvailability) => {
    if (readerAvailability === cmbScanner.CONSTANTS.AVAILABILITY_AVAILABLE) {
        oEventBus.publish("ScanView", "SetConnectionStatus", {
            statusText: "AVAILABLE"
        });
    }

    cmbScanner.connect((result) => {

    });

} else {

```

```

        oEventBus.publish("ScanView", "SetConnectionStatus", {
            statusText: "NOT AVAILABLE"
        });
    }
});

cmbScanner.setActiveStartScanningCallback((result) => {
    window.scannerActive = result;
});

cmbScanner.setPreviewOptions(cmbScanner.CONSTANTS.PREVIEW_OPTIONS.DEFAULTS | cmbScanner.CONSTANTS.PREVIEW_OPTIONS.HARDWARE_TRIGGER);
cmbScanner.setCameraMode(cmbScanner.CONSTANTS.CAMERA_MODES.NO_AIMER);

cmbScanner.loadScanner(0, (result) => {
    cmbScanner.connect((result) => {

    });
});

oEventBus.subscribe("Component", "LoadScanner", this.loadScanner, this);
},

loadScanner: function(sChanel, sEvent, oData) {

    cmbScanner.disconnect((result) => {
        cmbScanner.loadScanner(oData.selectedDevice, (result) => {
            cmbScanner.connect((result) => {

            });
        });
    });

    onExit: function() {

        oEventBus.unsubscribe("Component", "LoadScanner", this.loadScanner, this);

        cmbScanner.disconnect();
        cmbScanner.setAvailabilityCallback();
    }
});
});

```

cmbScanner is an object that represents our plugin. With this object we can access all [API methods](#) and Constants from our plugin.

- **scannerIsInitialized** - variable to make sure that we set some functions for initialization only one time
- **window.readerConnected** - global variable that is changed every time when the reader connection state is changed
- **window.scannerActive** - global variable that shows us if scanning is active or not
- **cmbScanner.addOnPause** - in this method we disconnection from reader device on Android (on iOS this is handled automatically), and stop listening to availability changing

- **cmbScanner.addOnResume** - in this method we try to connect with reader device again and set callback for availability changing
- **cmbScanner.setPreviewContainerPositionAndSize** - to set the size and position of the preview container. Learn more [here](#).
- **cmbScanner.setConnectionStateDidChangeOfReaderCallback** - all **connect/disconnect** events should be handled within the callback function set with this API method. If connection state is **connected** we set some reader device settings with [API methods](#) or directly with [sendCommand\(\)](#) method. Here in this example we enable symbologies and set trigger type. Also here we should Licensing the SDK. We will speak about this later in this section
- **cmbScanner.setAvailabilityCallback** - to monitor for availability of the MX device. If MX device is available we try to connect. Learn more [here](#).
- **cmbScanner.setActiveStartScanningCallback** - to monitor the state of the scanner. Learn more [here](#).
- **cmbScanner.loadScanner** - to get a scanner up and running. Learn more [here](#).

Here we are only configuring reader device, handling connections, availability, etc.. We will do scanning and getting results in other views. If we want to notify users about every connection state changed or other info about reader device we will be using [EventBus](#) sap object. With this object we can publish function and call them from any view in the application. In this example on connection state changed with EventBus we are calling **SetConnectionStatus** function that is implemented in view where scanning is performed and set label text to show user current connection state.

Now open **ScanView.view.xml** and add this code:

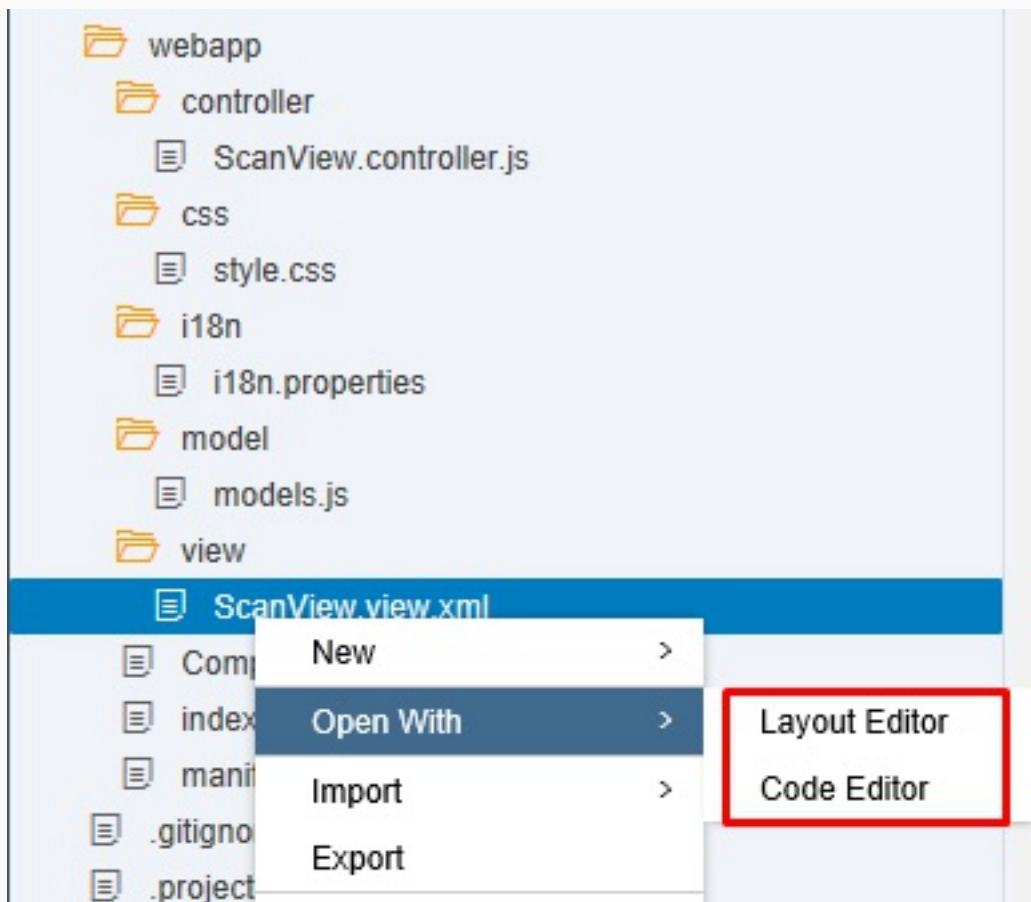
```
<mvc:View xmlns:mvc="sap.ui.core.mvc" xmlns:html="http://www.w3.org/1999/xhtml" xmlns:core="sap.ui.core"
controllerName="MyFristFioriApp.controller.ScanView" displayBlock="true">
  <App xmlns="sap.m">
    <pages>
      <Page title="MWBCameraDemo" showHeader="false">
        <subHeader>
          <Toolbar id="__toolbar2" width="100%">
            <content>
              <FlexBox id="__box0" width="100%" alignContent="Center" alignItems="Start" direction="Column"
fitContainer="true" justifyContent="Center">
                <items>
                  <Select id="selectActiveDevice" items="{/Devices}" textAlign="Center" selectedKey="0"
change="activeDeviceChanged">
                    <core:Item text="{text}" key="{key}" />
                  </Select>
                </items>
              </FlexBox>
              <Label id="lblStatus" text="DISCONNECTED" width="100%" textAlign="End" design="Bold" />
            </content>
          </Toolbar>
        </subHeader>
        <content>
          <FlexBox id="flexBoxContainer" width="100%" alignContent="Start" alignItems="Start" direction="Column"
fitContainer="true" />
        </content>
        <footer>
          <Toolbar id="__toolbar1" width="100%">
```

```

    <content>
      <Button id="btnScan" press="btnScanPress" text="Scan" width="100%" />
    </content>
  </Toolbar>
</footer>
</Page>
</pages>
</App>
</mvc:View>

```

You can design view with CodeEditor or with LayoutEditor.



After that open **ScanView.controller.js** and add this code

```

sap.ui.define([
  "sap/ui/core/mvc/Controller"
], function(Controller) {
  "use strict";

  var oEventBus = sap.ui.getCore().getEventBus();

  var oModel = new sap.ui.model.json.JSONModel();

  oModel.setData({
    Devices: [{
      key: "0",
      text: "MX Device"
    }
  ]
});

```

```

    }, {
      key: "1",
      text: "Mobile Camera"
    }]
  });

  return Controller.extend("MyFristFioriApp.controller.ScanView", {

    onInit: function() {

      this.getView().setModel(oModel);
    },

    onAfterRendering: function() {

      window.scannerActive = false;

      switch (window.readerConnected) {
        case cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED:
          this.getView().byId("lblStatus").setText("CONNECTED");
          break;
        case cmbScanner.CONSTANTS.CONNECTION_STATE_DISCONNECTED:
          this.getView().byId("lblStatus").setText("DISCONNECTED");
          break;
        case cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTING:
          this.getView().byId("lblStatus").setText("CONNECTING");
          break;
        case cmbScanner.CONSTANTS.CONNECTION_STATE_DISCONNECTING:
          this.getView().byId("lblStatus").setText("DISCONNECTING");
          break;
        default:
          this.getView().byId("lblStatus").setText("UNKNOWN");
          break;
      }

      oEventBus.subscribe("ScanView", "SetConnectionStatus", this.setConnectionStatus, this);

      cmbScanner.setResultCallback((result) => {
        if (result && result.readString) {

          var verticalLayoutContainer = new sap.ui.layout.VerticalLayout(null, {
            width: "100%"
          }).addStyleClass("sapUiSmallMarginTop");

          verticalLayoutContainer.addContent(new sap.m.Label({
            text: result.symbologyString + ":",
            textAlign: "Begin",
            design: "Bold"
          }).addStyleClass("sapUiSmallMarginBegin"));
          verticalLayoutContainer.addContent(new sap.m.Text({
            text: result.readString,
            textAlign: "Begin"
          }).addStyleClass("sapUiSmallMarginBegin"));

          this.getView().byId("flexBoxContainer").addItem(verticalLayoutContainer);
        }
      });
    },

    btnScanPress: function() {

```

```

if (window.readerConnected === cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED) {
  if (window.scannerActive === true) {
    cmbScanner.stopScanning();
  } else {
    cmbScanner.startScanning();
  }
}
},

setConnectionStatus: function(sChanel, sEvent, oData) {
  this.getView().byId("lblStatus").setText(oData.statusText);
},

activeDeviceChanged: function() {

  cmbScanner.disconnect((result) => {
    oEventBus.publish("Component", "LoadScanner", {
      selectedDevice: parseInt(this.getView().byId("selectActiveDevice").getSelectedKey())
    });
  });
},

onExit: function() {

  oEventBus.unsubscribe("ScanView", "SetConnectionStatus", this.setConnectionStatus, this);

  if (window.readerConnected === cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED) {
    if (window.scannerActive === true) {
      cmbScanner.stopScanning();
    }
  }

  cmbScanner.setResultCallback((result) => {
    return false;
  });
}

});
});

```

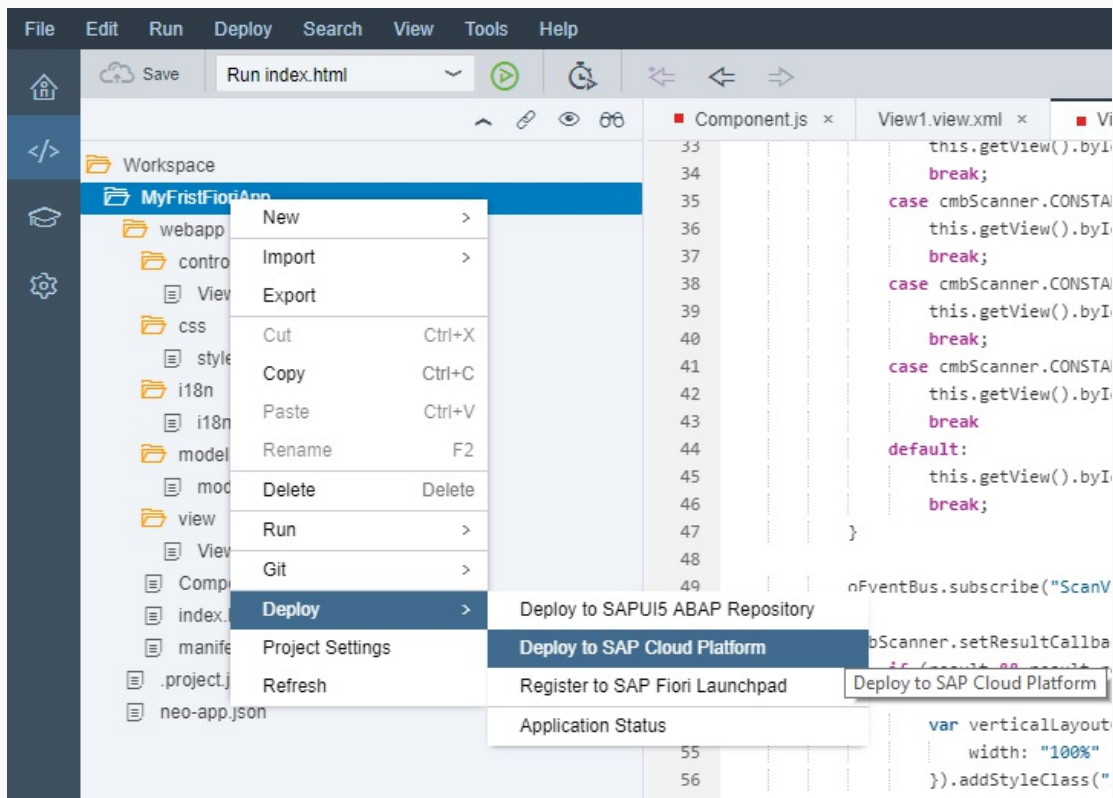
In this view we have a Scan button to [startScanning\(\)/stopScanning\(\)](#) and [cmbScanner.setResultCallback](#) to handle successful scan results.

Deploy app to HCP

After we finish creating and developing our Fiori project we need to prepare for mobile deployment.

First we need to deploy our project on [SAP HANA Cloud Platform](#).

- Right click on project folder then select Deploy and choose Deploy to SAP HANA Cloud Platform



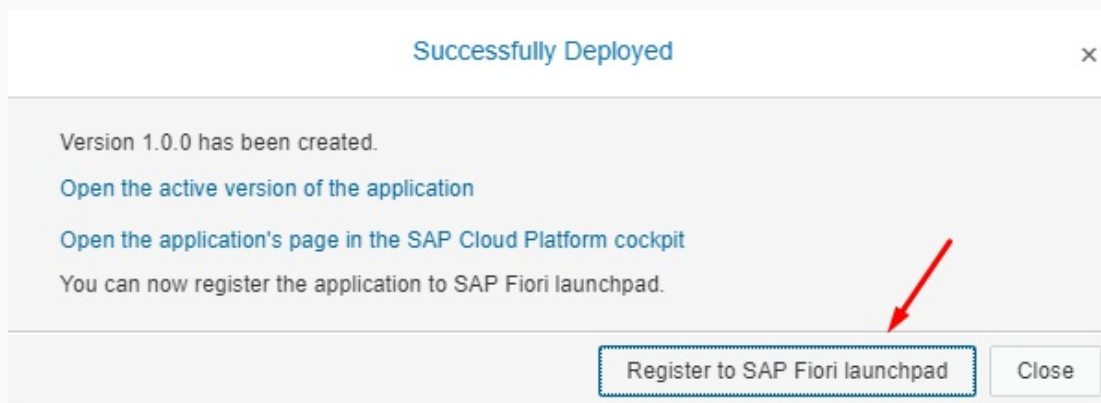
- Set application details and click Deploy

The screenshot shows the 'Deploy Application to SAP Cloud Platform' dialog box. It contains the following sections and fields:

- Application Details**
 - Radio buttons: ☒ Deploy a new application, ☐ Update an existing application
 - Account *: p2000298497trial (with a dropdown arrow and a 'Get Accounts' button)
 - Project Name: MyFristFioriApp
 - Application Name *: myfristfioriapp
- Version Management**
 - Version *: 1.0.0 (with a checkbox labeled 'Activate' that is checked)
- Buttons: Deploy, Cancel

A red arrow points to the 'Deploy' button.

- After a successful deploy, the next step is to Register to SAP Fiori launchpad that we've created in the previous section.



- Choose account and set application name.

General Information | Tile Configuration | Assignment | Confirmation

Register to SAP Fiori Launchpad

General Information

Provider Account * trial (fioriportal)

Application Name * MyFioriApp.MyFioriApp

Description

Intent

<input type="checkbox"/>	Semantic Object	Action
<input type="checkbox"/>	MyFioriApp	Display

[More information on intent properties.](#)

- Set Type (Static by default), set Title, and Subtitle.

General Information | Tile Configuration | Assignment | Confirmation

Register to SAP Fiori Launchpad

Tile Configuration

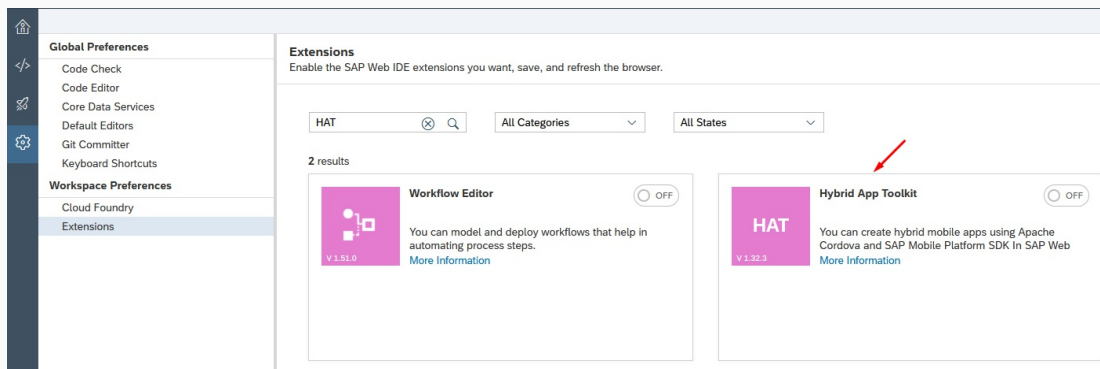
Type * Static

Title * FioriApp

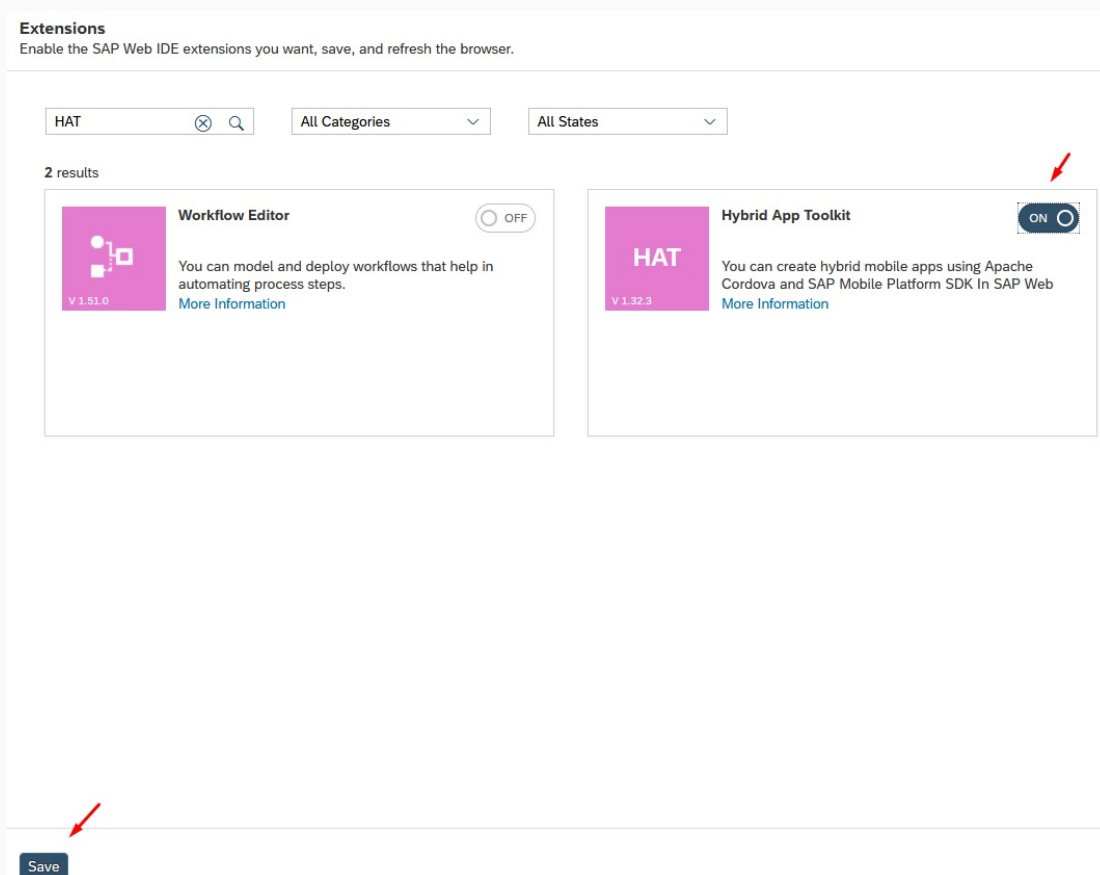
Subtitle Sample

Icon sap-icon://approvals Browse

- Choose Site that we create before, catalog, and group (sample catalog and group is created by default).



- Click finish on Confirmation tab and if everything is fine the popup below will be shown. Click OK to close this popup.



Build Fiori Mobile Application

A

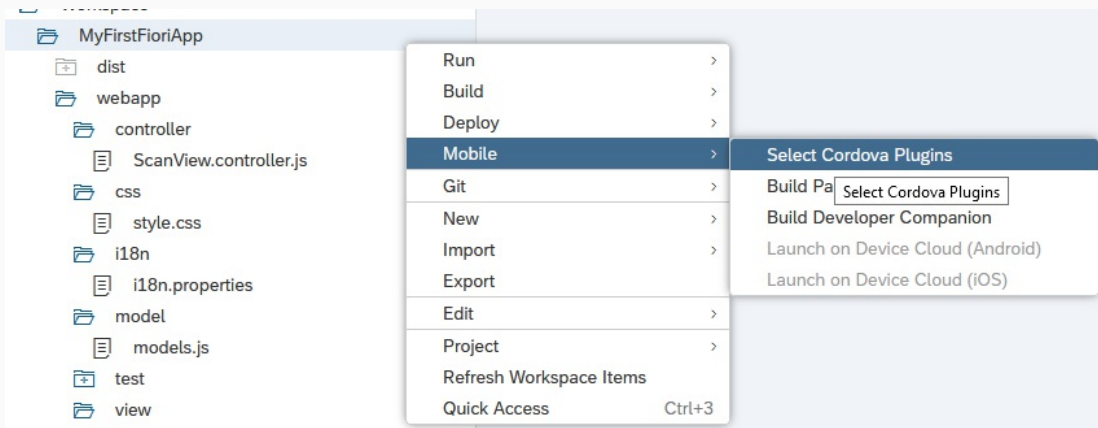
Licensing the SDK

If you plan to use the **cmbSDK** to do mobile scanning with a smartphone or a tablet

(without the MX mobile terminal), the SDK requires the installation of a license key. Without a license key, the SDK will still operate, although scanned results will be blurred (the SDK will randomly replace characters in the scan result with an asterisk character).

Contact your Cognex Sales Representative for information on how to obtain a license key including trial licenses which can be used for 30 days to evaluate the SDK.

After obtaining your license key open `i18n.properties` file in your project on SAP WEB IDE service, and set your obtained keys.



Then go back to the `Component.js` file and check this code:

```
cmbScanner.setConnectionStateDidChangeOfReaderCallback((connectionState) => {

    if (connectionState === cmbScanner.CONSTANTS.CONNECTION_STATE_CONNECTED) {

        oEventBus.publish("ScanView", "SetConnectionStatus", {
            statusText: "CONNECTED"
        });

        if (window.readerConnected !== connectionState) {
            cmbScanner.setSymbologyEnabled("SYMBOL.DATAMATRIX", true);
            cmbScanner.setSymbologyEnabled("SYMBOL.C128", true);
            cmbScanner.sendCommand("SET TRIGGER.TYPE 2");
        }

        var sdkKEY = "";
        if (Device.os.android)
            sdkKEY = this.getModel("i18n").getProperty("MX_MOBILE_LICENSE_ANDROID");
        else
            sdkKEY = this.getModel("i18n").getProperty("MX_MOBILE_LICENSE_ios");

        cmbScanner.registerSDK(sdkKEY, (res) => {
            switch (res) {
                case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_OK:
                    break;
                case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_KEY:
                    break;
                case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_CHECKSUM:
                    break;
                case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_APPLICATION:
                    break;
                case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_SDK_VERSION:
```

```
        break;
    case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_KEY_VERSION:
        break;
    case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_INVALID_PLATFORM:
        break;
    case cmbScanner.CONSTANTS.REGISTER_RESULTS.REGISTRATION_KEY_EXPIRED:
        break;
    default:
        break;
    }
});

.....
```

You can see that after successful connection you read this key from `i18n.properties` and call **`cmbScanner.registerSDK`** method to register SDK with your license key.

Checking result object which is sent back in callback function can tell us if registration is successful or what is the problem.