Oracle Fusion Field Service

Mobile Plugin Framework

Oracle Fusion Field Service
Mobile Plugin Framework

F75115-20

Authors: The Field Service Information Development Team

Contents

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the *Oracle Help Center*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we're working to remove insensitive terms from our products and documentation. We're also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Contacting Oracle

Access to Oracle Support

Customers can access electronic support through Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides. Please take one of the following surveys:

- For web-based user guide, *Web-based User Guide Survey*
- For tutorial feedback, *Tutorial Survey*

1 Overview of the Plugin Framework

The Plugin Framework

Oracle Fusion Field Service is a highly developed application that can be customized for the unique purposes and specialized business needs of organizations. That extensibility is achieved in part through the use of plugins, which can perform actions not found in the standard solution. Plug-ins appear as selectable links on the application. They open a new window, tab, or frame in a browser where an external HTML5 application is processed.

Plug-ins can be internal or external. Internal plugins can be created only by Oracle developers. External plugins, however, can be developed by anyone; they use externally stored data and communicate with the application by HTTP requests. The external plugins use HTML5 features such as offline work and persistence storage. However, be aware that if an external plugin has its own domain, offline mode is not supported for iOS Native Application. The plugin framework also allows these applications to exchange data with Oracle Fusion Field Service, in two ways:

- Traditional one-way communication when the plugin receives data from Oracle Fusion Field Service through the HTTP GET and POST parameters.
- Two-way communication using Oracle Fusion Field Service Plug-in API

The plugin framework offers these features:

- Integration with Oracle Fusion Field Service through an API and, therefore, the ability to perform complex tasks which could previously be performed only by internal plugins
- Ability to work offline with Mobility Cloud Service
- Plugin development by your organization or third-party developers without requiring Oracle developers

The plugins can manipulate these entities:

- Resource
- Activity
- Inventory
- Activity list
- Inventory list

Consequently, the plugins can be added to these contexts:

- Activity list
- Activity Details
- Inventory grid
- Add/Details inventory
- Route Map/Team Map (for Custom Map Layer assets)

About the Plugin API

The Plugin API enables customers or third-party integrators to extend the functionality of Oracle Fusion Field Service. Plugins that utilize this API are launched within Oracle Fusion Field Service, functioning as regular pages. These plugins can support complex business workflows, even in offline mode. The data exchanged between the plugin and Oracle Fusion Field Service is synchronized with the server-side automatically, eliminating the need for manual data transmission from the plugin to the server.

Accessing REST APIs from the Plugin Framework

This section outlines how you can access Rest APIs from custom plugins, and the ways in which you can authorize the APIs using different OAuth flows from plugins. Let's look at some of these options to determine the most appropriate approach for specific needs.

To determine the most appropriate method for accessing the REST API, you must consider these factors:

Which REST API is required?

- **Field Service API:** You can use this API to interact directly with Oracle Fusion Field Service data and functionality.
- **Fusion API:** Refers to the REST APIs provided by Oracle Fusion Cloud Applications, which you can access from within or alongside Oracle Fusion Field Service workflows.
- **Other API:** This encompasses any other external REST APIs you might need to connect to.

What Identity Provider is used to log in to Oracle Fusion Field Service?

An Identity Provider (IdP) manages your user identities and authentication for Oracle Fusion Field Service. Knowing which one you use is crucial for understanding the available authorization methods.

- **Internal Field Service:** Oracle Fusion Field Service uses its own internal user management system.
- **IDCS (Oracle Identity Cloud Service):** Your organization uses Oracle's cloud-based identity and access management service for Oracle Fusion Field Service login.
- **Other Identity Provider:** Your organization uses a different third-party identity management system for Oracle Fusion Field Service login.

What OAuth Authorization Grant Flow is supported by the Identity Provider?

OAuth 2.0 is a standard protocol for authorization. Different grant flows define how your application obtains permission to access protected resources. The supported flows depend on the Identity Provider in use.

- **OAuth Client Credentials Grant Flow:** This flow is typically used for server-to-server communication where an application (the "client") authenticates itself using its own credentials (a Client ID and Client Secret) to access resources. OAuth Client Credentials Grant flow doesn't involve a specific user's direct interaction.
- **OAuth User Assertion Grant Flow:** In this flow, the application uses an existing user's credentials or a signed assertion (like a JWT - JSON Web Token) to request an access token. This is often used when the application already can authenticate the user.
- **OAuth Authorization Code Grant Flow:** This flow involves a multi-step process. The user is redirected to the Identity Provider to grant permission. After the user authorizes, the application receives an authorization code,

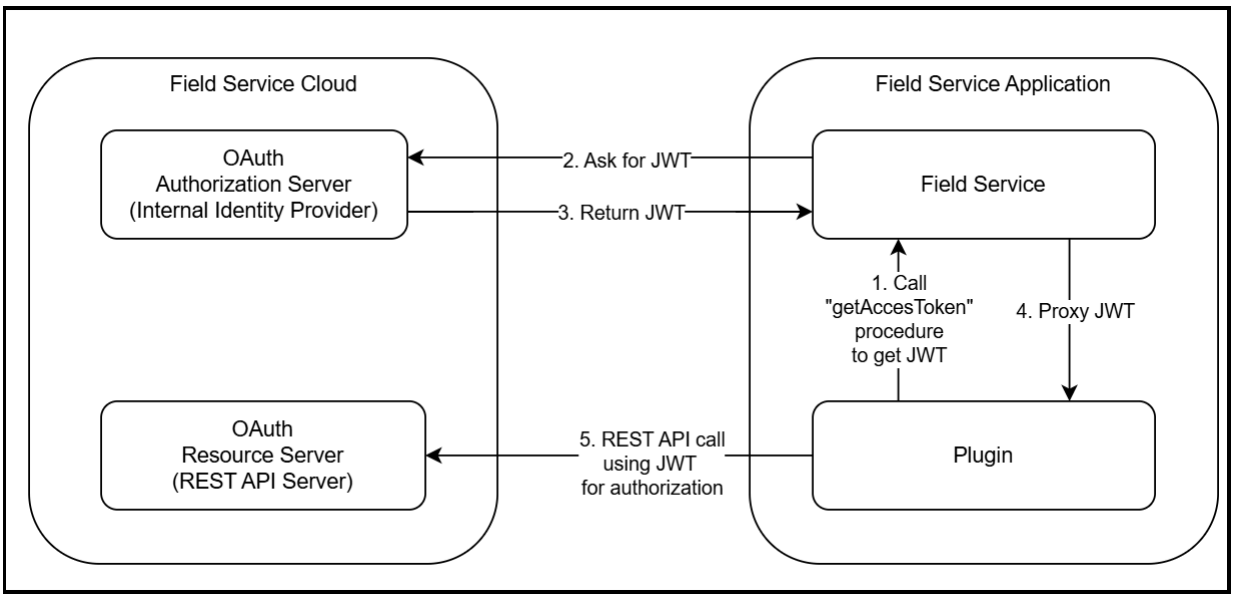
which it then exchanges for an access token. This flow is commonly used for web and mobile applications involving user interaction.

You can consider these three factors to determine the most suitable method for accessing the required REST API and the appropriate OAuth authorization flow. The specific steps for implementation depends on the selected API, your organization's identity provider configuration, and the capabilities of the application or tool you are using to access the API.

Accessing Oracle Fusion Field Service REST API

If a plugin requires access to the Oracle Fusion Field Service API, the most straightforward approach is to request a JSON Web Token (access tokens) directly from Oracle Fusion Field Service. Keep in mind that the JSON Web Tokens may be issued on a per-user basis.

This diagram illustrates the process workflow for a custom plugin accessing the Oracle Fusion Field Service API.



Advantage:

- The easiest way to obtain access to Oracle Fusion Field Service REST API.

Process Workflow Summary:

1. Add the Oracle Fusion Field Service API application using the **Configuration → Applications** page.
2. Configure API access as needed.
3. Register an application to the plugin using the **Edit Plugin** page.
4. Call the getAccessToken procedure from the plugin, providing this application in the procedure parameters.
5. Receive a JWT access token in the procedure's response.
6. Use the JWT access token for authorization in REST API requests.

Create Oracle Fusion Field Service Application

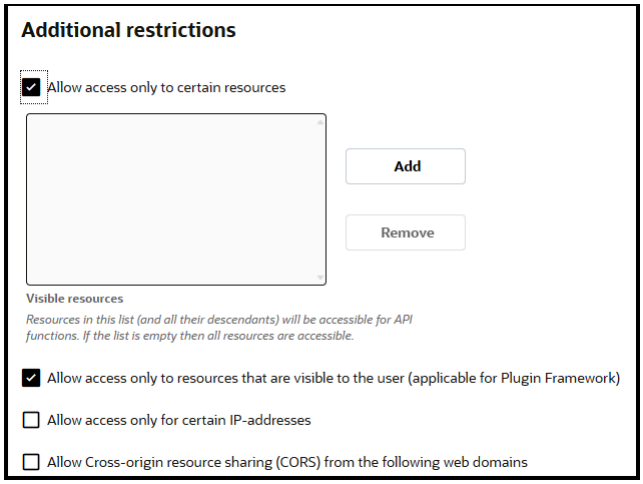
Note: Before you start creating an application, ensure to download the application's public key certificate.

To create an application with stricter API limitations,

1. Navigate to **Configuration > Applications** and then click **Add Application**. The Add Application page appears.
2. Select **Applications using Rest/SOAP API** from the **Application Type** drop-down list.
3. In the **Application Name** field, type the name of the application that you want to use/display on the Applications page.
4. Select **Field Service API** from the **Authenticate and Authorize** drop-down list.
5. In the **Application ID** field, specify the application ID.

   > **Note:** Ensure that the application ID you provide begins with an alphabet and contains only lowercase alphabets, numbers, and underscores (a-z, 0-9, _).

6. Click **Add**. The application is created and added to the Applications page.
7. Now, click the newly created application to modify the following:
   a. Select the **Active** checkbox to activate the application.
   b. In the **Authentication settings** section, select any of the following OAuth 2.0 authentication method.
      i. **Authenticate using Client ID/Client Secret for Client Credentials**: Note the generated Client ID and Client Secret.
      ii. **Authenticate using JWT assertion**: Upload your application's public key certificate.
   c. In the **API access** section, add and update the API access details per your requirement.
   d. In the **Additional restrictions** section, the **Allow access only to resources that are visible to the user (applicable for Plugin Framework)** checkbox.



   > **Note:** Enabling this setting ensures that the JWT access token is limited to the visibility scope of the user currently logged into the browser where the plugin is used. If the **Allow access only to certain resources** field is selected, the restrictions are applied in combination with this setting.

8. Click **Save**.

Configure Oracle Fusion Field Service Application to a Plugin

You can connect your Oracle Fusion Field Service application to a plugin through the Oracle Fusion Field Service configuration interface. The process involves making the plugin known to your Oracle Fusion Field Service environment and then making it accessible within the application's user interface.

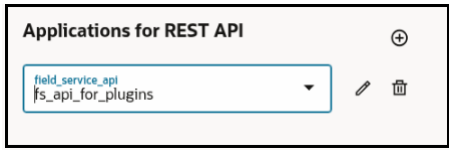
Plugins extend the functionality of Oracle Fusion Field Service, allowing you to integrate with external applications, adding custom UI elements, and automating specific workflows.

To connect the Oracle Fusion Field Service Application to a plugin:

1. Navigate to **Configuration > Displays > Forms and Plugins**.

   > **Note:** You can also access an active standard plugin using the **Configurations >Users Type> Screens** option.

2. Select the plugin from the available list that you would like to connect.
3. Click **Edit** from the **Action** menu. The **Edit Plugin** page appears.
4. In the **Applications for Rest API** section, click **Add** and specify the application key and select the required application from the list of Applications.



5. Click **Update**.

Sample Plugin to Obtain Access Token

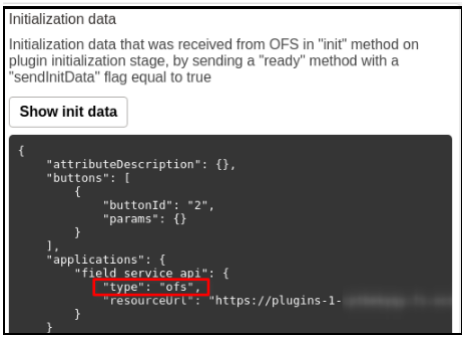
This topic outlines the typical flow when using the Sample Plugin to obtain an access token and subsequently fetch data from a REST API. This sample plugin example facilitates end-users' successful connection to and retrieval of data from REST APIs.

Initial Application List

Upon initialization, the plugin receives a message containing a list of available applications. This message includes the **resourceUrl** field for each application.

The **resourceUrl** is directly obtained from the plugin's configuration and is used by the Sample Plugin in its subsequent API requests.

The following screenshot provides details of the **Initialization data**, demonstrating how the plugin receives and utilizes the **resourceUrl**field from the configuration during its initialization process.



Obtaining an Access Token

To acquire an access token using the Sample Plugin, you use the **Procedures** functionality as shown in the screenshot below. You can

- Send a callProcedure method call to the plugin.
- Specify getAccessToken as the desired procedure within the callProcedure call.

Retrieving the Token

Once the callProcedure with getAccessToken is processed, the resulting access token is available in the **callProcedureResult** response method as shown below:

Validate the Token by Fetching Data

With the access token obtained, you can now fetch data from a REST API to verify its validity.

Note: When making requests to the Oracle Fusion Field Service API, the Sample Plugin supports calls not only to the base Oracle Fusion Field Service domain but also to domains prefixed with plugins-0- or plugins-1-. This allows for flexibility in accessing different environments or instances.

```
fetch("https://field-service-instance.com/rest/api", {
 headers: {
 Authorization: "Bearer <accessToken>"
 }
})
.then(resp => resp.json())
.then(json => console.log(json));
```

Accessing OAuth supported API

When Oracle Fusion Field Service acts as a client to access an external API secured with OAuth 2.0, it needs to obtain an access token. The method of obtaining this token depends on the OAuth 2.0 grant type supported by the target API and the capabilities of how Oracle Fusion Field Service (or its plugins/integrations) can interact with the authorization server.

If a plugin requires access to an API other than the Oracle Fusion Field Service API, these are the supported authorization flows:

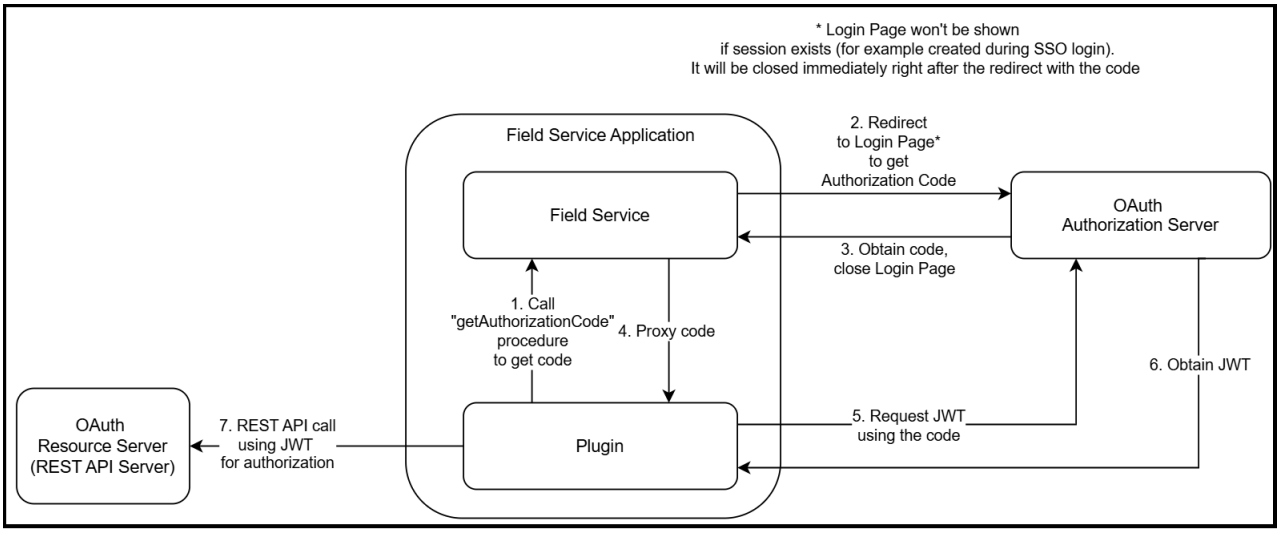
- OAuth Authorization Code Grant Flow (*OAuth Authorization Code Flow – getAuthorizationCode Procedure*)
- OAuth User Assertion Grant Flow (*OAuth User Assertion Flow (getAccessToken procedure)*)
- OAuth Client Credentials Grant Flow (*OAuth Client Credentials Flow (getAccessToken procedure)*)

Let's look at each of these in detail.

OAuth Authorization Code Flow – getAuthorizationCode Procedure

In this flow, authorization occurs by redirecting the user from the plugin to the Identity Provider's login page, returning to the plugin after authorization. If SSO is enabled and the user's session is active, the user is redirected without being prompted for login credentials.

This diagram provides a visual representation of the OAuth Authorization Code Grant Flow, illustrating the sequence of steps involved in obtaining authorization for accessing the API.



Advantages:

- JWT is issued on behalf of the user, not the application.
- No sensitive information is stored on the Oracle Fusion Field Service side.

Process Workflow Summary:

1. Configure an application on the Identity Provider to support the OAuth Authorization Code Flow.
2. Use credentials (Client ID, Scope, Identity Provider endpoint) to generate the URL to the Identity Provider's Authorization Code Endpoint.
3. Call the getAuthorizationCode procedure from the plugin with this URL in the procedure parameters.
4. Obtain an authorization code in the procedure response.
5. Obtain a JWT access token with this authorization code from the plugin.
6. Use the JWT access token for REST API request authorization.

This section outlines the process to:

1. Create an integrated application with OAuth support in Oracle Identity Cloud Service (IDCS), which will return an authorization code or access token (JWT).
2. Use a plugin to obtain an access token (JWT). This step assists in setting up the Fusion REST API backend in a Visual Builder Cloud Service (VBCS) application. You can skip this step if your VBCS instance already includes Fusion REST API in the Catalog.
3. Develop a simple VBCS application to demonstrate how to retrieve a JWT from a plugin and use it to send a standard Fusion REST API request.

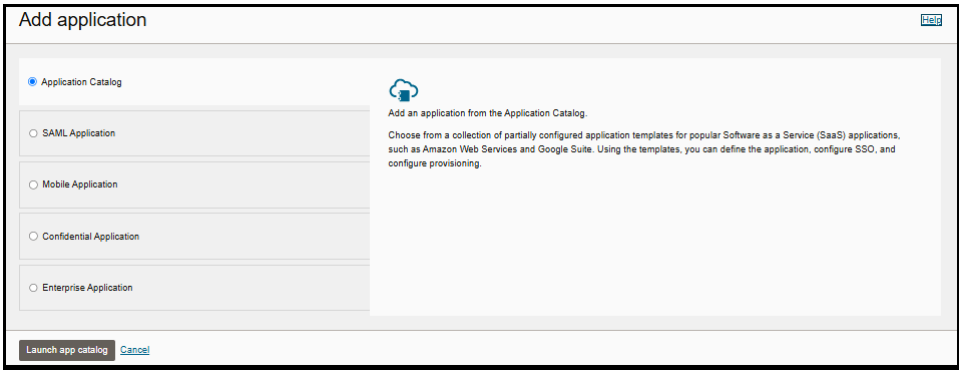
IDCS Configuration for OAuth Authorization Code Flow

The setup aims to create an integrated application in IDCS that obtains an access token (JWT) using an authorization code to make authorized requests to REST APIs accessible through the application.

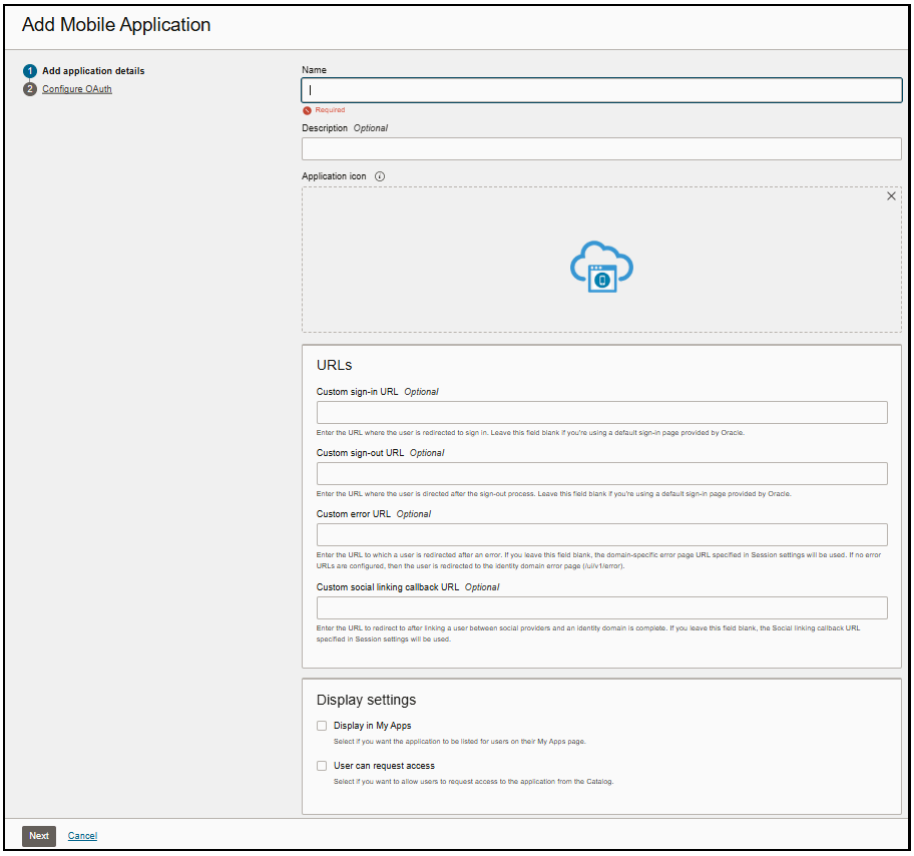
Steps to Create an Integrated Application in IDCS:

1. Navigate to the Identity Domain Configuration in IDCS.
2. Select the **Integrated Applications** section from the left pane.

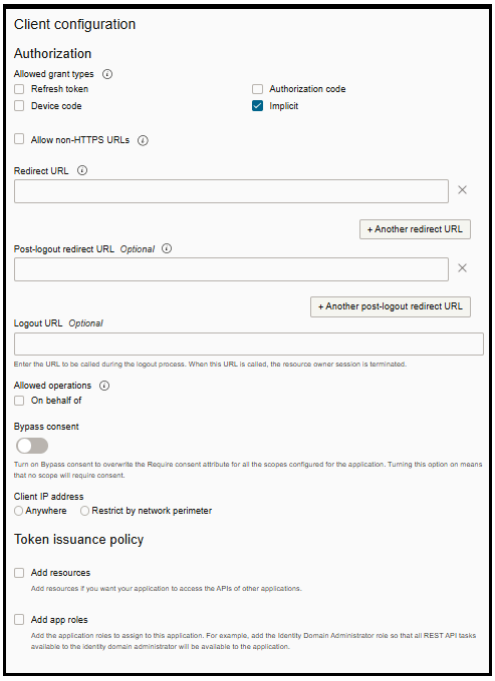
3. Click **Add application** at the top of the page.
   The **Add application** dialog box appears.



4. In the Add application dialog box, select **Mobile Application** and then click **Launch Workflow**. The **Add Mobile Application** page appears.



5. Name your application and click **Next** to proceed to the **Configure OAuth** step.

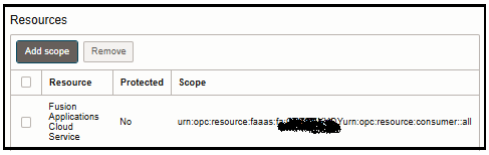


6. In the **Authorization** section, enable the **Authorization code** grant type. Uncheck other grant types unless necessary.

> **Note:** Enable the **Refresh Token** grant type if the plugin uses refresh token functionality.

7. In the **Redirect URL** field, enter the URL for redirection to your Field Service instance, appended with `/plugin-auth-redirect/` For example: `https://<your-field-service-instance-domain>/plugin-auth-redirect/`.
8. In the **Allowed Operations** section, enable the **On behalf of** checkbox. This allows the client application to access user-endpoints based on their privileges.
9. In the **Token Issuance Policy** section, select **Add resources** if you want your application to access the APIs of other applications.
10. In the **Resources** section, click **Add Scope**. A list of applications appears in the Add scope dialog box.
11. Select Fusion Applications Cloud Service and then click **Add**.

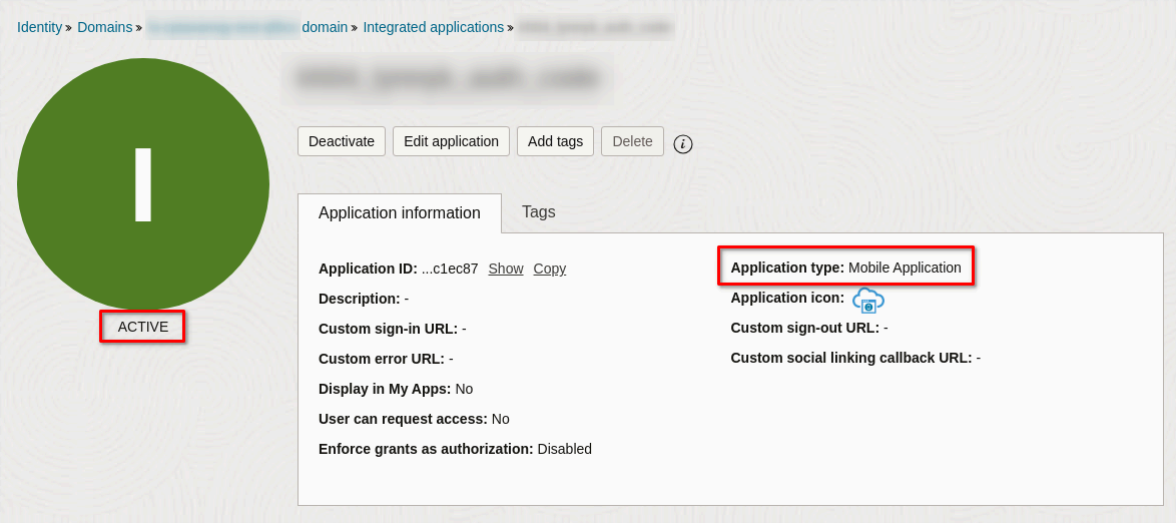
    The selected application is added to the Resource scope.

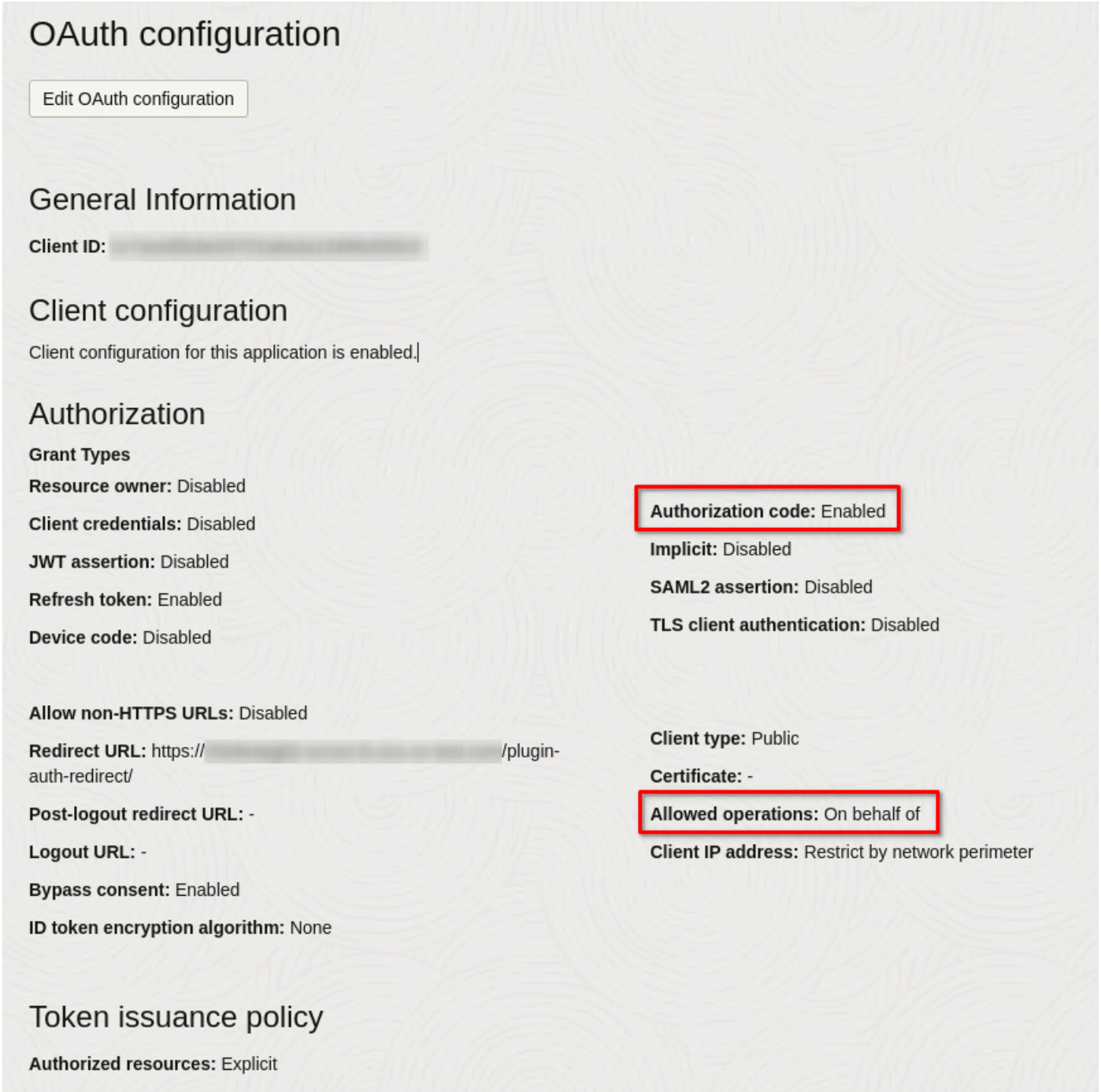


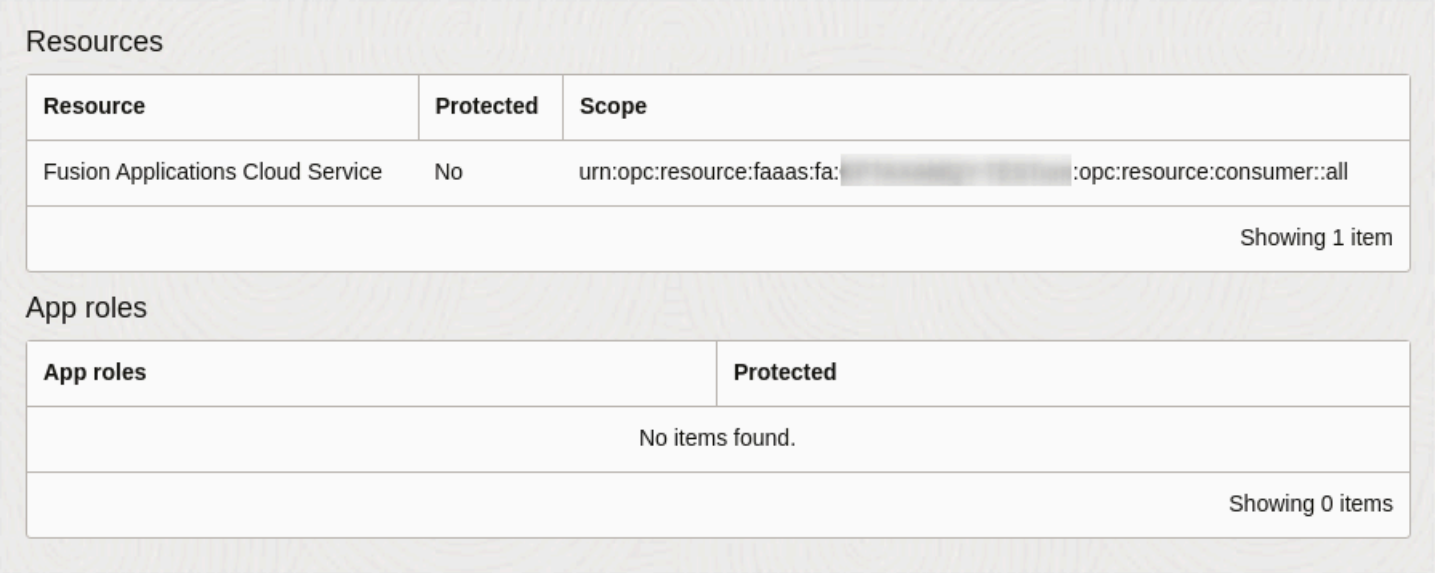
> **Note:** If Fusion Applications Cloud Service is not listed, it means Fusion Service is not linked with the IDCS domain. You need to create the application in the domain linked to Fusion Service.

12. Click **Finish**. The newly created application is now listed on the Integrated Applications page and its status is **Inactive**.

13. Click the **Activate** icon to activate the application. Once activated, your application appears as follows:




OAuth configuration
Edit OAuth configuration
General Information
Client ID:
Client configuration
Client configuration for this application is enabled.
Authorization
Grant Types
Resource owner: Disabled
Client credentials: Disabled
JWT assertion: Disabled
Refresh token: Enabled
Device code: Disabled
Authorization code: Enabled
Implicit: Disabled
SAML2 assertion: Disabled
TLS client authentication: Disabled
Allow non-HTTPS URLs: Disabled
Redirect URL: https:// /plugin-auth-redirect/
Post-logout redirect URL: -
Logout URL: -
Bypass consent: Enabled
ID token encryption algorithm: None
Client type: Public
Certificate: -
Allowed operations: On behalf of
Client IP address: Restrict by network perimeter
Token issuance policy
Authorized resources: Explicit



Simple Authorization Plugin (Pure JavaScript)

This topic provides step-by-step instructions to use a simple JavaScript plugin to showcase the OAuth Authorization Code flow within Oracle Fusion Field Service.

The plugin enables you to:

- Obtain an Authorization Code,
- Retrieve a JWT (JSON Web Token), and
- Leverage the JWT to access data from a REST API, with Fusion as an example.

> **Note:** This plugin is intended for demonstration and testing to understand the authorization flow. It works with Oracle Fusion Field Service version 25B and later.

Prerequisites

Before using this plugin, ensure that:

- **IDCS Application Configuration**: You have a properly configured application within Oracle Identity Cloud Service (IDCS) or your chosen Identity Provider. This application is set up to support the OAuth Authorization Code grant type.
- **Plugin Installation**: The Simple Authorization Plugin is successfully installed and configured within your Oracle Fusion Field Service environment.

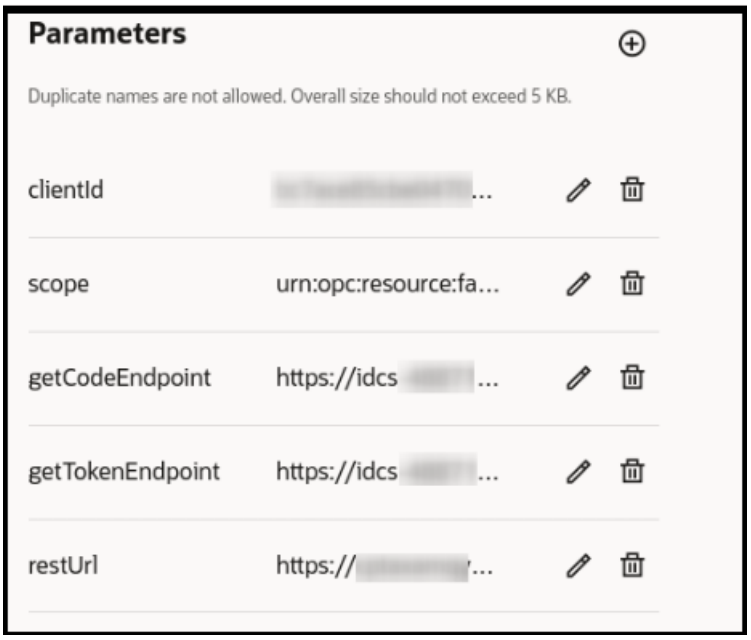
Plugin Parameters

These parameters are configured at the plugin level and define the endpoints and credentials required for the authorization flow. Your administrator will typically configure these.

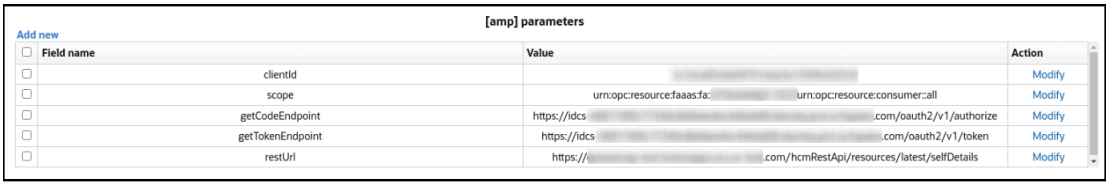
- getCodeEndpoint: The authorization endpoint of your Identity Provider. This is where the user will be redirected to authenticate and authorize the application. URL for obtaining the authorization code, For example:
  - IDCS: `https://{idcsUrl}/oauth2/v1/authorize`
  - Microsoft: `https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/authorize`
- getTokenEndpoint: The token endpoint of your Identity Provider. This endpoint exchanges the Authorization Code for an Access Token (JWT). URL for obtaining the token (For example:
  - IDCS: `https://{idcsUrl}/oauth2/v1/token`
  - Microsoft: `https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token`)
- restUrl: The URL of the REST API you want to access. (For example: IDCS: `https://{fusionUrl}/hcmRestApi/resources/latest/selfDetails`, Microsoft: `https://graph.microsoft.com/v1.0/me`)
- clientId: The client ID of the application you configured in your Identity Provider. This identifies your application to the Identity Provider.

- scope: The permissions your application requests from the user. These are defined in your Identity Provider application configuration.

  Example of plugin parameters configuration:



  Example of button parameters configuration:



Basic Plugin Workflow

You can utilize the Simple Authorization Plugin to go through the complete OAuth 2.0 Authorization Code flow and interact with protected REST APIs. Here's a step-by-step guide on how to use it.

1. Click **Get Code** within the plugin's user interface in Oracle Fusion Field Service. You will be automatically redirected to the login page of your configured Identity Provider. On desktop browsers, a new popup window or a separate tab appears if you are using the Oracle Fusion Field Service Mobile Application for user login.
2. On the Identity Provider's login page, enter your username and password to authenticate your identity.

   > **Note:** If you have an active Single Sign-On (SSO) session, you might be automatically redirected to Oracle Fusion Field Service without having to enter your credentials again.

3. Upon successful authentication and authorization, the Identity Provider will redirect you to the Oracle Fusion Field Service application. The URL in your browser will now contain an **Authorization Code**. This code is a temporary, single-use credential.

4. Click **Get JWT**. The plugin will communicate with the Identity Provider's token endpoint to obtain the access token (JWT). Upon a successful request, the Identity Provider will respond with an **Access Token**. This token is a JWT (JSON Web Token) and is a temporary credential granting access to protected resources.
5. Click **Get Data** to retrieve data from the REST API (restUrl from plugin parameters). The plugin will use the obtained JWT to request to the configured REST API endpoint in the background. If the Access Token is valid and the API call is successful, the REST API will respond with the requested data. This data will be displayed within the Simple Authorization Plugin user interface in Oracle Fusion Field Service.
6. This plugin can also be used to test the flow on pure JS to check if everything works fine. Moreover, you could use an access token to authorize the wizard of the REST API backend in the creation of a VBCS application (it is applicable if the VBCS instance has no access to REST API without authorization).

   Here is an example plugin screen with successfully received code:

The listing of plugin's code:

```
!--
Oracle Field Service Sample plugin
Copyright (c) 2023 Oracle and/or its affiliates.
Licensed under the Universal Permissive License v 1.0 as shown at https://oss.oracle.com/licenses/upl/
-->


<!DOCTYPE html>
<html lang="en-us">
<head>
 <title>Authorization micro plugin</title>
</head>
<body>

<p>1. Configuration.</p>
<p>Use plugin 'Secure Parameters' configuration to provide clientId, getCodeEndpoint and getTokenEndpoint
 parameters, or fill them manually.</p>
<div>
```

```
 <label for="field_get_code_endpoint">getCodeEndpoint: </label><input id="field_get_code_endpoint"><br/>
 Example: https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/authorize
 Example: https://{idcsUrl}/oauth2/v1/authorize
</div>
<div>
 <label for="field_get_token_endpoint">getTokenEndpoint: </label><input id="field_get_token_endpoint"><br/>
 Example: https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token
 Example: https://{idcsUrl}/oauth2/v1/token
</div>
<div>
 <label for="field_rest_url">restUrl: </label><input id="field_rest_url"><br/>
 Example: https://graph.microsoft.com/v1.0/me
 Example: https://{fusionUrl}/hcmRestApi/resources/latest/selfDetails
</div>
<div><label for="field_client_id">clientId: </label><input id="field_client_id"></div>

<div><label for="field_scope">scope: </label><input id="field_scope"></div>
<hr>
<div><label for="field_ofs_origin">Redirect URI: (used in generation of redirect URI)</label><input disabled
 id="field_ofs_origin">/plugin-auth-redirect/</div>
<div>redirectUri must be the same while get authorization code and access token</div>
<hr>
<input id="field_include_code_challenge" type="checkbox" value="field_include_code_challenge"> Include Code
 Challenge
<p>Auto-generated code challenge that will be used in Plugin API Call Procedure request (to obtain
 Authorization Code) and HTTP fetch (to obtain JWT Access Token).</p>
<div><label for="field_code_verifier">Code Verifier (random string): </label><input
 id="field_code_verifier"></div>
<div><label for="field_code_challenge">Code Challenge (signature): </label><input
 id="field_code_challenge"></div>
<button id="button_generate_code_challenge">Regenerate</button>
<p>Auto-generated unique procedure call ID (will be used in Plugin API Call Procedure request).</p>
<div><label for="field_call_id">Procedure call ID: </label><input id="field_call_id"></div>
<button id="button_generate_call_id">Regenerate</button>
<hr>


<p>2. Obtain Code using call procedure 'getAuthorizationCode' of Plugin API to open Secure Tab to authorize
 and get auth code.</p>
<div><textarea id="field_procedure_request" title="Procedure request" rows="10" cols="100"></textarea></div>
<button id="button_get_auth_code">Get code</button>
<div>List of called procedures:</div>
<pre id="call_procedures_list"></pre>
<div><label for="field_auth_code">Last received code: </label><input id="field_auth_code"></div>
<hr>

<p>3. Obtain Access Token (JWT) using Code by HTTP fetch function.</p>
<!--<button id="button_generate_jwt_fetch_request_for_debug">Generate JWT fetch request</button>-->
<pre id="field_jwt_fetch_request_for_debug"></pre>
<button id="button_get_auth_token">Get JWT</button>
<pre id="field_auth_response"></pre>
<div><label for="field_auth_token">Token: </label><input id="field_auth_token"></div>
<hr>

<p>4. Obtain REST data using JWT Access Token by HTTP fetch function. Please note to request new JWT you need
 request new code.</p>
<!--<button id="button_generate_rest_data_fetch_request_for_debug">Generate Data fetch request</button>-->
<pre id="field_rest_data_fetch_request_for_debug"></pre>
<button id="button_get_rest_data">Get Data</button>
<pre id="field_rest_data"></pre>


</body>
<script>
 const INIT_FLOW = 'init';
 const OPEN_FLOW = 'open';
```

```
let currentFlow;

let resolveReadyMessage;
let listOfRunProcedures = [];


function getRandomString() {
return crypto.randomUUID().replace(/\+/g, '-');
}

function base64UrlEncode(str) {
return btoa(String.fromCharCode.apply(null, new Uint8Array(str)))
.replace(/\+/g, '-')
.replace(/\//g, '_')
.replace(/=+$/, '');
}

async function generateCodeChallenge(codeVerifier) {
const encoder = new TextEncoder();
const data = encoder.encode(codeVerifier);
const digest = await crypto.subtle.digest('SHA-256', data);

return base64UrlEncode(digest);
}

async function fillCodeChallenge() {
let codeVerifier = getRandomString();
let codeChallenge = await generateCodeChallenge(codeVerifier);

document.getElementById('field_code_verifier').value = codeVerifier;
document.getElementById('field_code_challenge').value = codeChallenge;

prepareCallProcedureMessage();
}

function handleInitMessage(message, event) {
localStorage.setItem("field_ofs_origin", message.origin);

currentFlow = INIT_FLOW;

let initEndMessage = {
apiVersion: 1,
method: 'initEnd'
};

sendPostMessage(initEndMessage);

resolveReadyMessage();
}

function handleOpenMessage(message, event) {
currentFlow = OPEN_FLOW;

if (!message.allowedProcedures) {
console.error("allowedProcedures is not found in open message", JSON.stringify(message));
}

if (!message.allowedProcedures.getAuthorizationCode) {
console.error("getAuthorizationCode procedure must be allowed in 'open' plugin API message",
JSON.stringify(message));

alert('getAuthorizationCode procedure must be allowed');
}

let openParams = message.openParams || {};
let securedData = message.securedData || {};
```

```
document.getElementById('field_client_id').value = openParams.clientId || securedData.clientId || '';
document.getElementById('field_scope').value = openParams.scope || securedData.scope || '';
document.getElementById('field_get_code_endpoint').value = openParams.getCodeEndpoint ||
securedData.getCodeEndpoint || '';
document.getElementById('field_get_token_endpoint').value = openParams.getTokenEndpoint ||
securedData.getTokenEndpoint || '';
document.getElementById('field_rest_url').value = openParams.restUrl || securedData.restUrl || '';

resolveReadyMessage();
}

function updateListOfProcedures() {
document.getElementById('call_procedures_list').innerText = JSON.stringify(listOfRunProcedures, null, 4);
}

function handleProcedureResultMessage(message, event) {
//just log the response
const foundCallIndex = listOfRunProcedures.find(procedureCall => procedureCall.callId === message.callId);

if (foundCallIndex !== -1) {
foundCallIndex.response = message;
}

updateListOfProcedures();

//keep code if request is successful
if (message.resultData.result === 'completed') {
document.getElementById('field_auth_code').value = message.resultData.code;
generateJwtFetchRequestForConsole();
}
}

function handleProcedureErrorMessage(message, event) {

//just log the response
const foundCallIndex = listOfRunProcedures.find(procedureCall => procedureCall.callId === message.callId);

if (foundCallIndex !== -1) {
foundCallIndex.response = message;
}

updateListOfProcedures();
}

function sendPostMessage(message) {
window.parent.postMessage(message, document.referrer);
}

function prepareCallProcedureMessage() {
let getCodeEndpoint = document.getElementById('field_get_code_endpoint').value.trim();
let clientId = document.getElementById('field_client_id').value.trim();
let redirectUri = encodeURIComponent(document.getElementById('field_ofs_origin').value.trim() + '/plugin-
auth-redirect/');
let scope = document.getElementById('field_scope').value.trim();
// let state = document.getElementById('field_client_id').value.trim();
let includeCodeChallenge = document.getElementById('field_include_code_challenge').checked;
let codeChallenge = document.getElementById('field_code_challenge').value.trim();
let callId = document.getElementById('field_call_id').value.trim();

let getCodeUrl = `${getCodeEndpoint}?response_type=code&client_id=${clientId}&redirect_uri=
${redirectUri}&scope=${scope}`;

if (includeCodeChallenge) {
getCodeUrl += `&code_challenge_method=S256&code_challenge=${codeChallenge}`;
}
```

```
// if (state) {
// getCodeUrl += `&state=${state}`;
// }

let message = {
apiVersion: 1,
method: 'callProcedure',
procedure: "getAuthorizationCode",
callId: callId,
params: {
"url": getCodeUrl
}
};

document.getElementById('field_procedure_request').value = JSON.stringify(message, null, 4);
}

function refreshCallId() {
let callId = getRandomString();
document.getElementById('field_call_id').value = callId;

prepareCallProcedureMessage();
}

function sendCallProcedureMessage() {
let callProcedureMessage = document.getElementById('field_procedure_request').value

try {
let parsedCallProcedureMessage = JSON.parse(callProcedureMessage);
listOfRunProcedures.push({
callId: parsedCallProcedureMessage.callId,
request: parsedCallProcedureMessage,
response: 'Wait for response..'
});
updateListOfProcedures();

sendPostMessage(parsedCallProcedureMessage);
refreshCallId();
} catch(e) {
console.error(e);
}
}

function sendReadyMessage() {
let readyMessage = {
apiVersion: 1,
method: 'ready',
sendInitData: true,
sendMessageAsJsObject: true
};

return new Promise((resolve, reject) => {
sendPostMessage(readyMessage);
resolveReadyMessage = resolve;
})
}

function generateJwtFetchRequestForConsole() {
if (!document.getElementById('field_auth_code').value) {
console.error('Auth code could not be empty');
alert('Auth code could not be empty');

return;
}
```

```
let getTokenEndpoint = document.getElementById('field_get_token_endpoint').value.trim();
let clientId = document.getElementById('field_client_id').value.trim();
let redirectUri = document.getElementById('field_ofs_origin').value.trim() + '/plugin-auth-redirect/';
let authCode = document.getElementById('field_auth_code').value.trim();
let codeVerifier = document.getElementById('field_code_verifier').value.trim();
let includeCodeChallenge = document.getElementById('field_include_code_challenge').checked;


let requestParams = {
client_id: clientId,
grant_type: 'authorization_code',
redirect_uri: redirectUri,
code: authCode
};

if (includeCodeChallenge) {
requestParams.code_verifier = codeVerifier;
}

let getAccessTokenBody = new URLSearchParams(requestParams).toString();

let fetchRequestForDebug = `
fetch('${getTokenEndpoint}', {
headers: {
"content-type": "application/x-www-form-urlencoded; charset=UTF-8",
},
method: "POST",
body: '${getAccessTokenBody}'
});
`;

document.getElementById('field_jwt_fetch_request_for_debug').innerText = fetchRequestForDebug;
}

async function fetchToken() {
if (!document.getElementById('field_auth_code').value) {
console.error('Auth code could not be empty');
alert ('Auth code could not be empty');

return;
}

let getTokenEndpoint = document.getElementById('field_get_token_endpoint').value.trim();
let clientId = document.getElementById('field_client_id').value.trim();
let redirectUri = document.getElementById('field_ofs_origin').value.trim() + '/plugin-auth-redirect/';
let authCode = document.getElementById('field_auth_code').value.trim();
let codeVerifier = document.getElementById('field_code_verifier').value.trim();
let includeCodeChallenge = document.getElementById('field_include_code_challenge').checked;

let requestParams = {
client_id: clientId,
grant_type: 'authorization_code',
redirect_uri: redirectUri,
code: authCode
}

if (includeCodeChallenge) {
requestParams.code_verifier = codeVerifier;
}

let getAccessTokenBody = new URLSearchParams(requestParams).toString();

let response = await fetch(`${getTokenEndpoint}`, {
"headers": {
"content-type": "application/x-www-form-urlencoded; charset=UTF-8",
},
```

```
"method": "POST",
"body": getAccessTokenBody
});

if (response.status !== 200) {
console.error('Token was not obtained');
alert ('Token was not obtained');

return;
}

let jsonResponse = await response.json();
document.getElementById('field_auth_response').innerText = JSON.stringify(jsonResponse, null, 4);

if (jsonResponse.access_token) {
document.getElementById('field_auth_token').value = jsonResponse.access_token;
generateRestDataFetchRequestForConsole();
}
}

function generateRestDataFetchRequestForConsole() {
let authToken = document.getElementById('field_auth_token').value.trim();
let restUrl = document.getElementById('field_rest_url').value.trim();

let fetchRequestForDebug = `
fetch("${restUrl}", {
 headers: {
 authorization: 'Bearer ${authToken}'
 }
});
`;

document.getElementById('field_rest_data_fetch_request_for_debug').innerText = fetchRequestForDebug;

}

async function getRestData() {
if (!document.getElementById('field_auth_token').value) {
console.error('Auth token could not be empty');
alert ('Auth token could not be empty');

return;
}

let authToken = document.getElementById('field_auth_token').value.trim();
let restUrl = document.getElementById('field_rest_url').value.trim();

let response = await fetch(restUrl, {
"headers": {
authorization: `Bearer ${authToken}`
}
});

if (response.status !== 200) {
console.error('Data was not obtained');
alert ('Data was not obtained');

return;
}

let jsonResponse = await response.json();

document.getElementById('field_rest_data').innerText = JSON.stringify(jsonResponse, null, 4);
}

function initButtonHandling() {
```

```
document.getElementById('button_generate_code_challenge').addEventListener('click', async function () {
await fillCodeChallenge();
});
document.getElementById('button_generate_call_id').addEventListener('click', function () {
refreshCallId();
});

document.getElementById('button_get_auth_code').addEventListener('click', function () {
sendCallProcedureMessage();
});

// document.getElementById('button_generate_jwt_fetch_request_for_debug').addEventListener('click', function
() {
// generateJwtFetchRequestForConsole();
// });

document.getElementById('button_get_auth_token').addEventListener('click', async function () {
await fetchToken();
});

// document.getElementById('button_generate_rest_data_fetch_request_for_debug').addEventListener('click',
function () {
// generateRestDataFetchRequestForConsole();
// });

document.getElementById('button_get_rest_data').addEventListener('click', async function () {
await getRestData();
});

document.getElementById('field_get_code_endpoint').addEventListener('change', function () {
prepareCallProcedureMessage();
});

document.getElementById('field_include_code_challenge').addEventListener('change', function () {
prepareCallProcedureMessage();
});

document.getElementById('field_get_token_endpoint').addEventListener('change', function () {
generateJwtFetchRequestForConsole();
});

document.getElementById('field_client_id').addEventListener('change', function () {
prepareCallProcedureMessage();
});

document.getElementById('field_rest_url').addEventListener('change', function () {
generateRestDataFetchRequestForConsole();
});

document.getElementById('field_scope').addEventListener('change', function () {
prepareCallProcedureMessage();
});
}

function initPostMessageHandling() {
window.addEventListener("message", function(event) {
if (event.source === window) {
// ignore messages from itself

return;
}

if (new URL(event.origin).host !== new URL(document.referrer).host ) {
console.info("Came message from another origin", JSON.stringify(event.data));

return; // Ensure the message is from the expected origin
```

```
}

if (!event.data.apiVersion) {
// is not considered as a message from Plugin API

return;
}

if (!event.data.method) {
console.warn("No 'method' field in post message", JSON.stringify(event.data));

return;
}

switch (event.data.method) {
case 'init':
handleInitMessage(event.data, event);
break;

case 'open':
handleOpenMessage(event.data, event);
break;

case 'callProcedureResult':
handleProcedureResultMessage(event.data, event);
break;

case 'error':
handleProcedureErrorMessage(event.data, event);
break;
}
});
}

initPostMessageHandling();

document.getElementById('field_ofs_origin').value = localStorage.getItem("field_ofs_origin");

sendReadyMessage().then(() => {
if (currentFlow === OPEN_FLOW) {
initButtonHandling();

fillCodeChallenge();
refreshCallId();
}
});

</script>
```

Use VBCS Application as an Oracle Fusion Field Service External Plugin for REST API Access

This section describes how to use a VBCS application embedded as an Oracle Fusion Field Service external plugin to authorize and retrieve data from a REST API (specifically Oracle Fusion in this example). The authorization relies on a JWT obtained through the Plugin Framework's access code flow.

Prerequisites

- You have configured an IDCS application for the OAuth Authorization Code flow.
- You have created, staged, and configured a VBCS application as an external plugin in Oracle Fusion Field Service.
- You have run the whole flow. You have opened the VBCS Plugin, authorized, and obtained REST API data from Fusion.

Create a VBCS Application for Oracle Fusion Field Service Plugin Integration

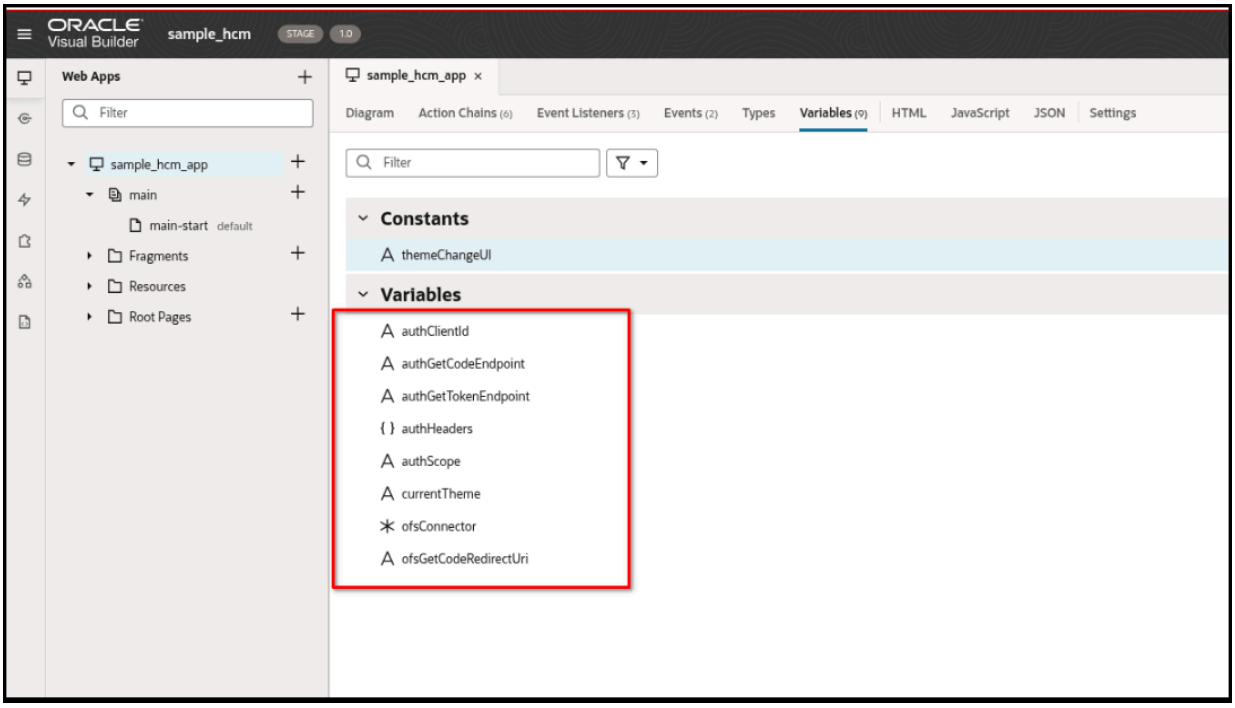
This topic provides the steps to create a VBCS (Visual Builder Cloud Service) application that can be used as an external plugin within Oracle Fusion Field Service to access a REST API.

Create a New VBCS Application

To create a new VBCS application:

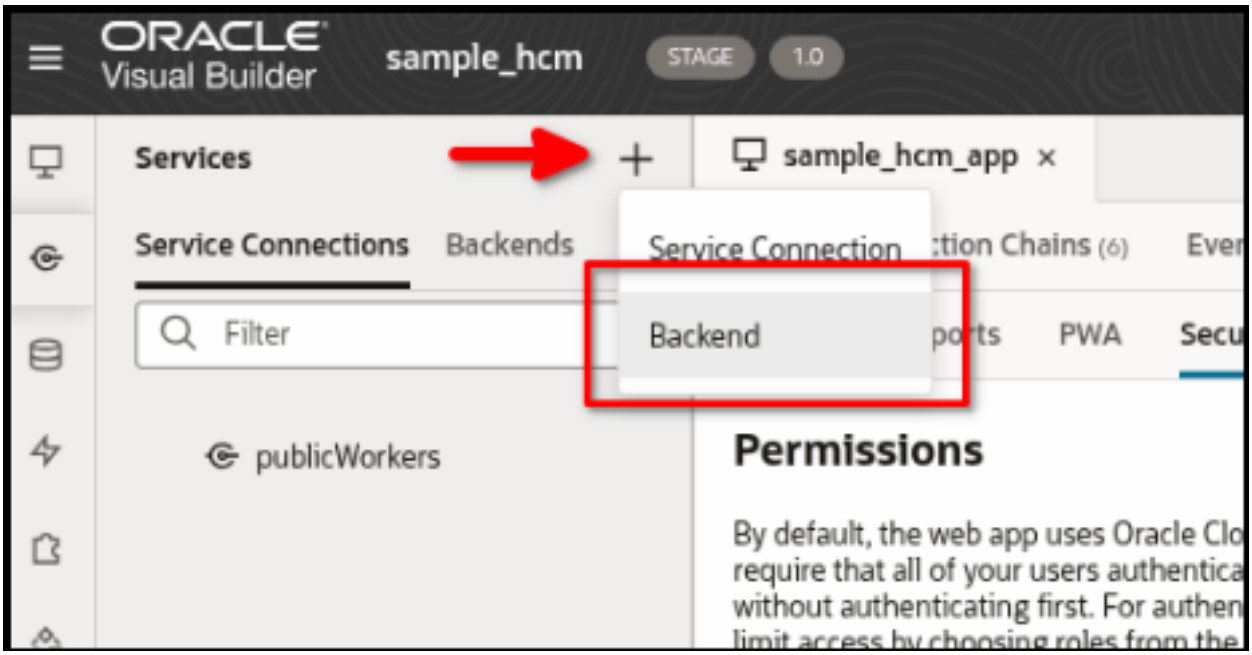
1. Open your VBCS environment.
2. Create a new VBCS application and specify a suitable name and ID for your application.
3. Within your newly created VBCS application, navigate to the **Variables** section.
    a. Create the necessary variables that your application will use during its operation. The specific variables you required depend on the logic of your plugin. You can use this code to add the required variables:

```
"variables": {
 "authClientId": {
 "type": "string"
 },
 "authGetCodeEndpoint": {
 "type": "string"
 },
 "authGetTokenEndpoint": {
 "type": "string"
 },
 "authHeaders": {
 "type": "object"
 },
 "authScope": {
 "type": "string"
 },
 "ofsGetCodeRedirectUri": {
 "type": "string"
 },
 "ofsConnector": {
 "type": "any"
 },
 ...}
```

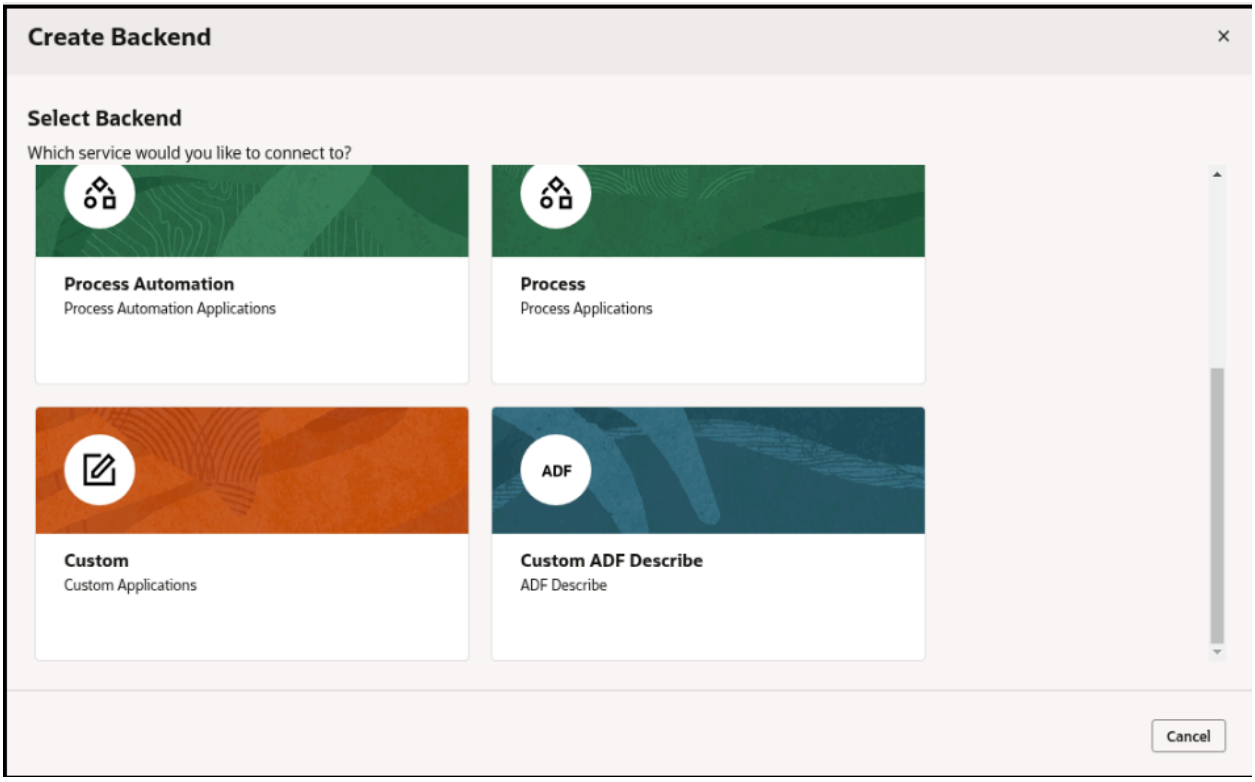


Add a Backend

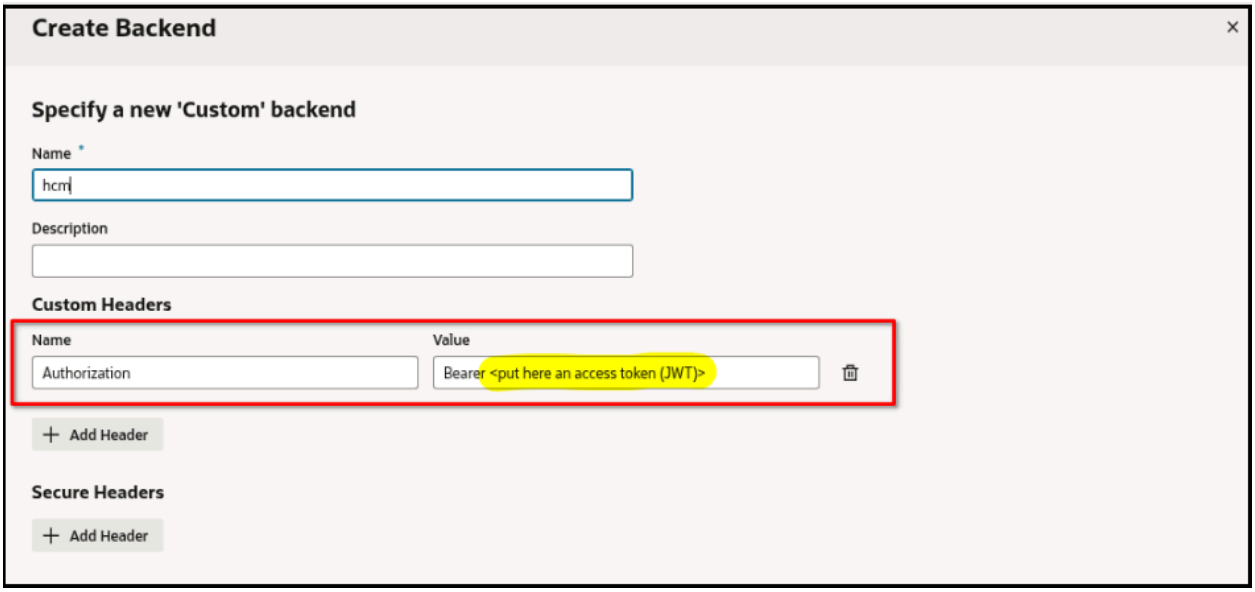
1. Navigate to the **Backend** section in VBCS.
2. Click the button to create a new backend.



3. Select **Custom** as the backend type.



4. Enter a name for your custom backend. For example, hcm.

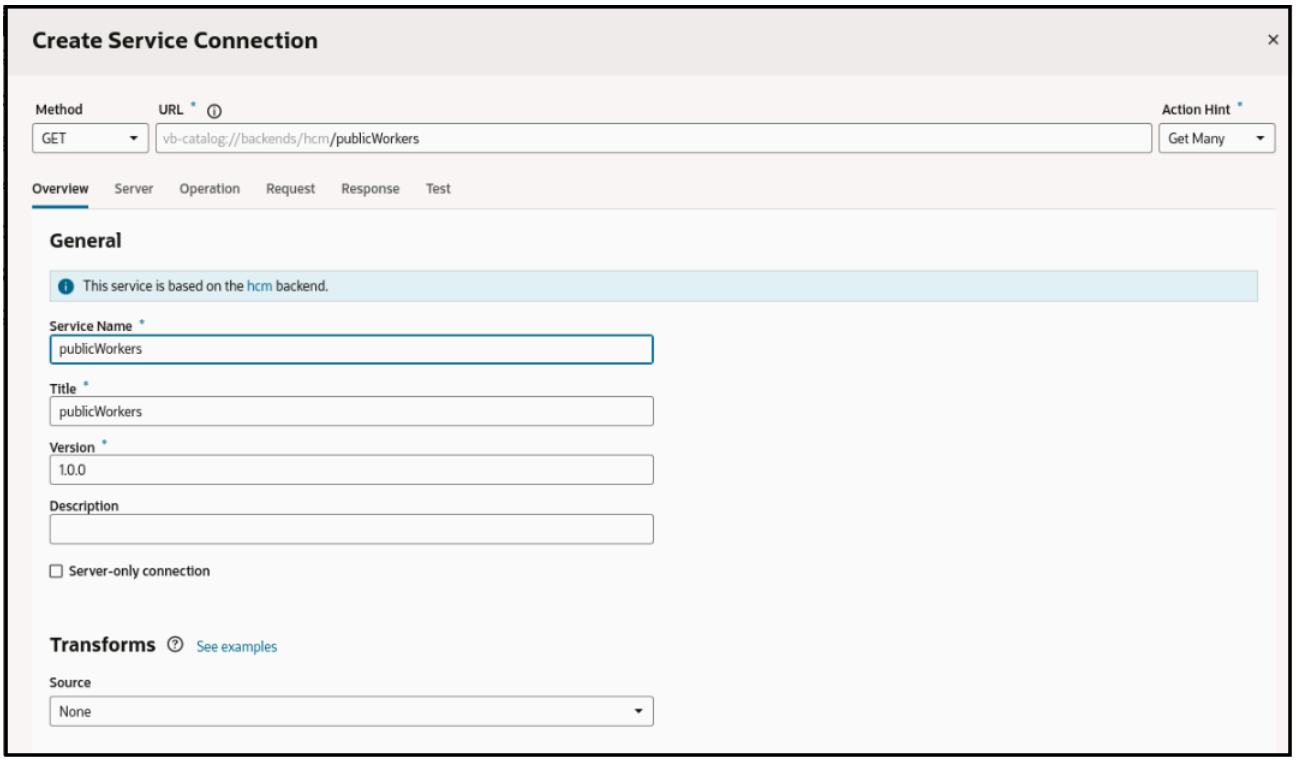


5. To allow VBCS to understand the structure of the REST API during development:
   a. Add a custom header named Authorization.
   b. Set the Value of this header to Bearer xxxx, where xxxx is a valid JWT (obtained through a tool like the Simple Authorization Plugin described in the earlier topic).

   > **Note:** This header serves a temporary function to facilitate the design process and will be removed in a subsequent phase. The JWTs have a finite lifespan and will require periodic updates during development.

Add a 'Public Workers' Service Connection

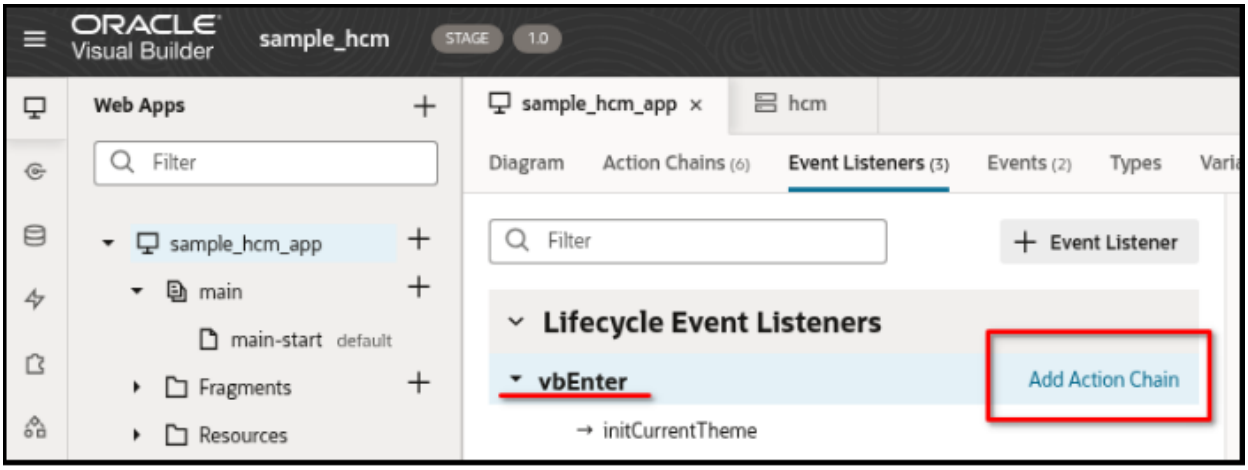
1. Next, navigate to the **Service Connections** section.
2. Click the button to add a new service connection.
3. Click **Select from Catalog**.
4. In the **Custom** section, select the custom backend you just created (for example, hcm).
5. Select **Define by endpoint**.
6. Set the **Service Name** and **Title** to publicWorkers.



7. Set the URL suffix to /publicWorkers. This will be appended to the base URL of your HCM backend. In our example, the plugin will request data from Public Workers HCM REST API *https://docs.oracle.com/en/cloud/saas/human-resources/24c/farws/op-publicworkers-get.html*

Implement Plugin Framework Initialization

1. Navigate to the main page or the relevant flow where the plugin will be used.



2. Open the **Event Listeners** tab for the page.
3. Select the **vbEnter** event.
4. Click **Add Action Chain** to create a new Action Chain **_init** associated with this event.
5. Within this _init Action Chain, add the following code:

```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class _init extends ActionChain {

 /**
 * @param {Object} context
 */
 async run(context) {
 const { $application, $constants, $variables } = context;
 //---------------------------------------

 class OfsConnector {
 constructor(params) {
 const {
 apiVersion = 1,
 onInit = (jsonData) => {},
 onOpen = (jsonData) => {},
 onCallProcedureResult = (jsonData) => {},
 onError = (jsonData) => {
 alert(jsonData);
 }
 } = params;


 this.API_VERSION = apiVersion;

 this.onInit = onInit.bind(this);
 this.onOpen = onOpen.bind(this);
```

```
this.onCallProcedureResult = onCallProcedureResult.bind(this);
this.onError = onError.bind(this);

this.TRANSMIT_METHODS = {
METHOD_READY: 'ready',
METHOD_INIT_END: 'initEnd',
METHOD_CLOSE: 'close',
METHOD_UPDATE: 'update',
METHOD_CALL_PROCEDURE: 'callProcedure',
METHOD_SLEEP: 'sleep'
};

this.RECEIVE_METHODS = {
METHOD_ERROR: 'error',
METHOD_INIT: 'init',
METHOD_OPEN: 'open',
METHOD_UPDATE_RESULT: 'updateResult',
METHOD_CALL_PROCEDURE_RESULT: 'callProcedureResult',
METHOD_WAKEUP: 'wakeup'
};

window.addEventListener("message", this.onPostMessage.bind(this), false);
}

/**
* @param {Object} dataToSend
* @returns {void}
*/
sendMessage(dataToSend) {
const originUrl = this.constructor._getOriginUrl();
const origin = originUrl ? this.constructor._getOrigin(originUrl) : '*';

dataToSend.apiVersion = this.API_VERSION;

parent.postMessage(dataToSend, origin);
}


onPostMessage(event) {
// Accept only external messages
if (event.source === window) {
return;
}

const jsonData = event.data;

if (!jsonData) {
this.onError('Received message without data: ' + jsonData);

return;
}

switch (jsonData.method) {
case this.RECEIVE_METHODS.METHOD_INIT:
this.onInit(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_OPEN:
this.onOpen(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_CALL_PROCEDURE_RESULT:
this.onCallProcedureResult(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_ERROR:
```

```
this.onError('Received message with error method: ' + jsonData);
break;

default:
this.onError('Received message with unsupported method: ' + jsonData);
}
}

static generateRandomString (length) {
return btoa(String.fromCharCode.apply(null, window.crypto.getRandomValues(new Uint8Array(length))))
.replaceAll('=', '')
.replaceAll('/', '')
.replaceAll('+', '').substr(0, length);
}

static _getOrigin(url) {
if (typeof url === 'string' && url !== '') {
if (url.indexOf("://") > -1) {
return (window.location.protocol || 'https:') + url.split('/')[2];
} else {
return (window.location.protocol || 'https:') + url.split('/')[0];
}
}

return '';
}

static _getOriginUrl() {
if (document.referrer) {
return document.referrer;
}

if (document.location.ancestorOrigins && document.location.ancestorOrigins[0]) {
return document.location.ancestorOrigins[0];
}

return null;
}
}

//---------------------------------------

const onInit = function(jsonData) {
if (jsonData.origin) {
localStorage.setItem('authRedirectOrigin', jsonData.origin || '');
}

this.sendMessage({
method: this.TRANSMIT_METHODS.METHOD_INIT_END
});
};

const onOpen = function(jsonData) {
let openParams = jsonData.openParams || {};
let securedData = jsonData.securedData || {};

// getCodeEndpoint
// Example: https://{idcsUrl}/oauth2/v1/authorize

// getTokenEndpoint
// Example: https://{idcsUrl}/oauth2/v1/token

$application.variables.authClientId = openParams.clientId || securedData.clientId || '';
$application.variables.authScope = openParams.scope || securedData.scope || '';
$application.variables.authGetCodeEndpoint = openParams.getCodeEndpoint || securedData.getCodeEndpoint
|| '';
```

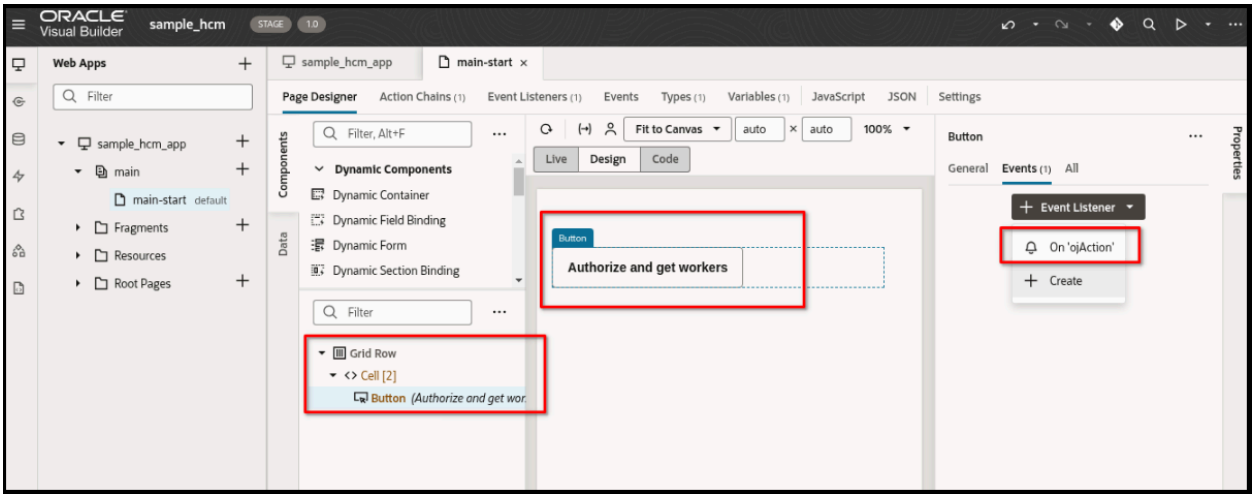
```
 $application.variables.authGetTokenEndpoint = openParams.getTokenEndpoint ||
 securedData.getTokenEndpoint || '';

 $application.variables.ofsGetCodeRedirectUri = localStorage.getItem('authRedirectOrigin') + '/plugin-
auth-redirect/';
 };

 $application.variables.ofsConnector = new OfsConnector({
 onInit: onInit,
 onOpen: onOpen
 });

 $application.variables.ofsConnector.sendMessage({
 method: $application.variables.ofsConnector.TRANSMIT_METHODS.METHOD_READY,
 sendInitData: true,
 sendMessageAsJsObject: true
 });

 //---------------------------------------

 }
 }

 return _init;
});
```

This code helps you:

a. Include the Plugin Framework connector class into your VBCS application.
b. Store this connector class in an application variable.
c. Initiate the communication with the Field Service Plugin Framework.

Create Authorization and Data Retrieval

1. On your VBCS page, place a **Button** with the label **'Authorize and get workers'**.



2. Create a new Action Chain named `_authorize` and associate it with the button's `ojAction` event listener.
3. Change code of the '_authorize' Action Chain, and add the following:

```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
```

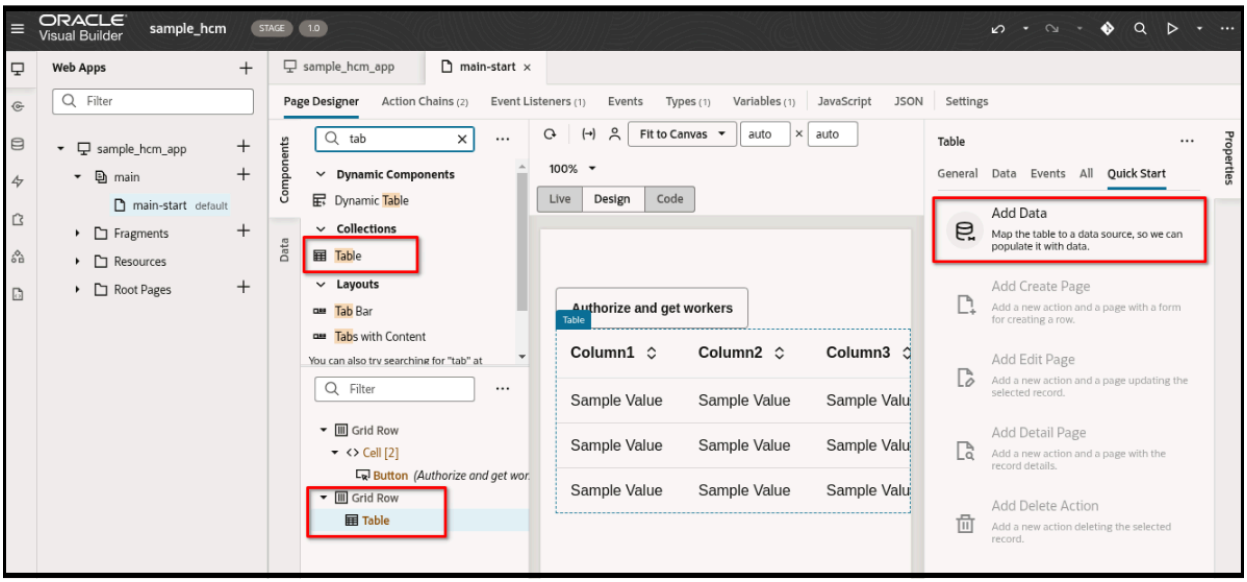
```
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class _authorize extends ActionChain {

 /**
 * @param {Object} context
 */
 async run(context) {
 const { $application, $constants, $variables } = context;




 //---------------------------------------

 // const state = '';
 // let authCodeChallenge = $application.variables.ofsConnector.constructor.generateRandomString(45);

 const callId = $application.variables.ofsConnector.constructor.generateRandomString(16);

 let authUrl = $application.variables.authGetCodeEndpoint
 + "?client_id=" + $application.variables.authClientId
 + '&response_type=code'
 // + '&challenge_method=plain'
 // + "&code_challenge=" + authCodeChallenge
 + "&scope=" + encodeURIComponent($application.variables.authScope)
 //+ "&state=" + JSON.stringify(state)
 + "&redirect_uri=" + encodeURIComponent($application.variables.ofsGetCodeRedirectUri);

 $application.variables.ofsConnector.onCallProcedureResult = async function(procedureResult) {
 if (!procedureResult.callId || procedureResult.callId !== callId) {
 return;
 }

 let code = procedureResult.resultData.code;

 const result = await fetch(
 $application.variables.authGetTokenEndpoint,
 {
 method: 'POST',
 headers: {
 'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'
 },
 body: new URLSearchParams({
 client_id: $application.variables.authClientId,
 grant_type: "authorization_code",
 redirect_uri: $application.variables.ofsGetCodeRedirectUri,
 // code_verifier: authCodeChallenge,
 code: code,
 })
 }
 );

 const jsonData = await result.json();

 if (jsonData.access_token) {
 $application.variables.authHeaders = { 'Authorization': 'Bearer ' + jsonData.access_token };
 }
 };
```

```
$application.variables.ofsConnector.sendMessage({
method: $application.variables.ofsConnector.TRANSMIT_METHODS.METHOD_CALL_PROCEDURE,
callId: callId,
procedure: "getAuthorizationCode",
params: {
url: authUrl
}
});

//--------------------------------------




}
}

return _authorize;
});
```

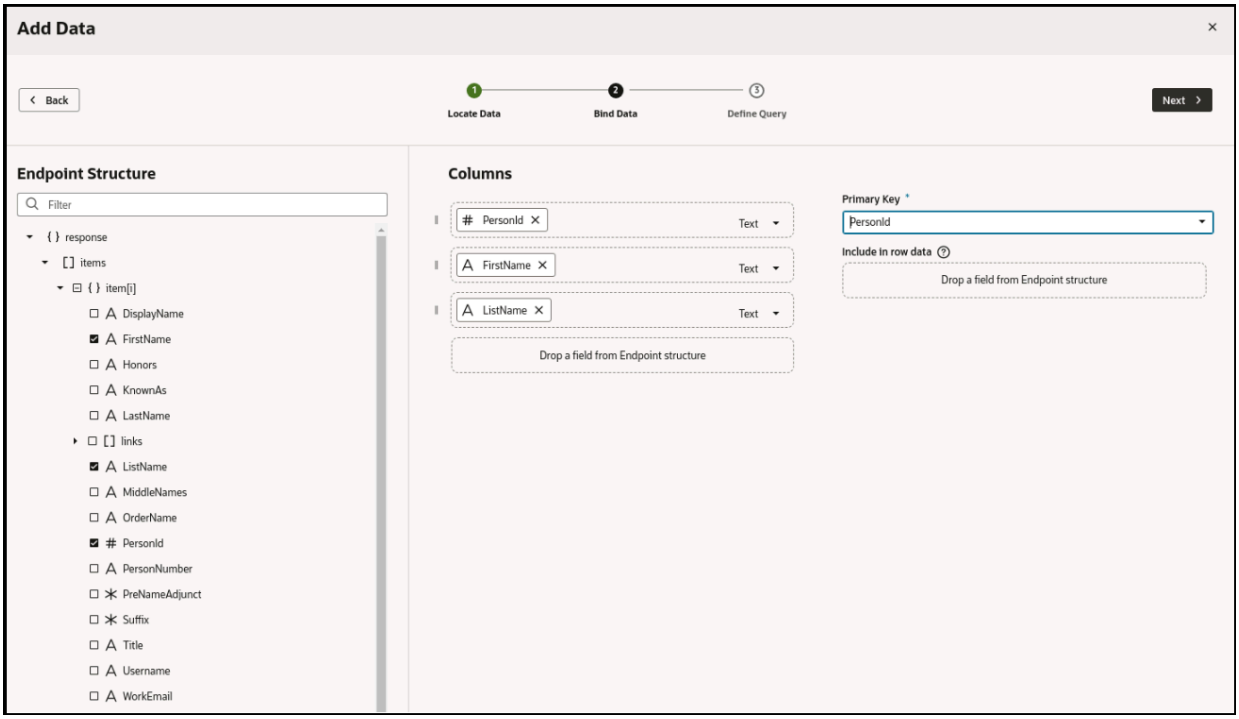
4. Click the **Authorize** button to trigger the Action chain. It calls the Plugin Framework function to get an authorization code. Next, by code it obtains a token and gets the REST API data into the table.

Display Data in a Table

1. Drag a Table component onto your VBCS page layout.\

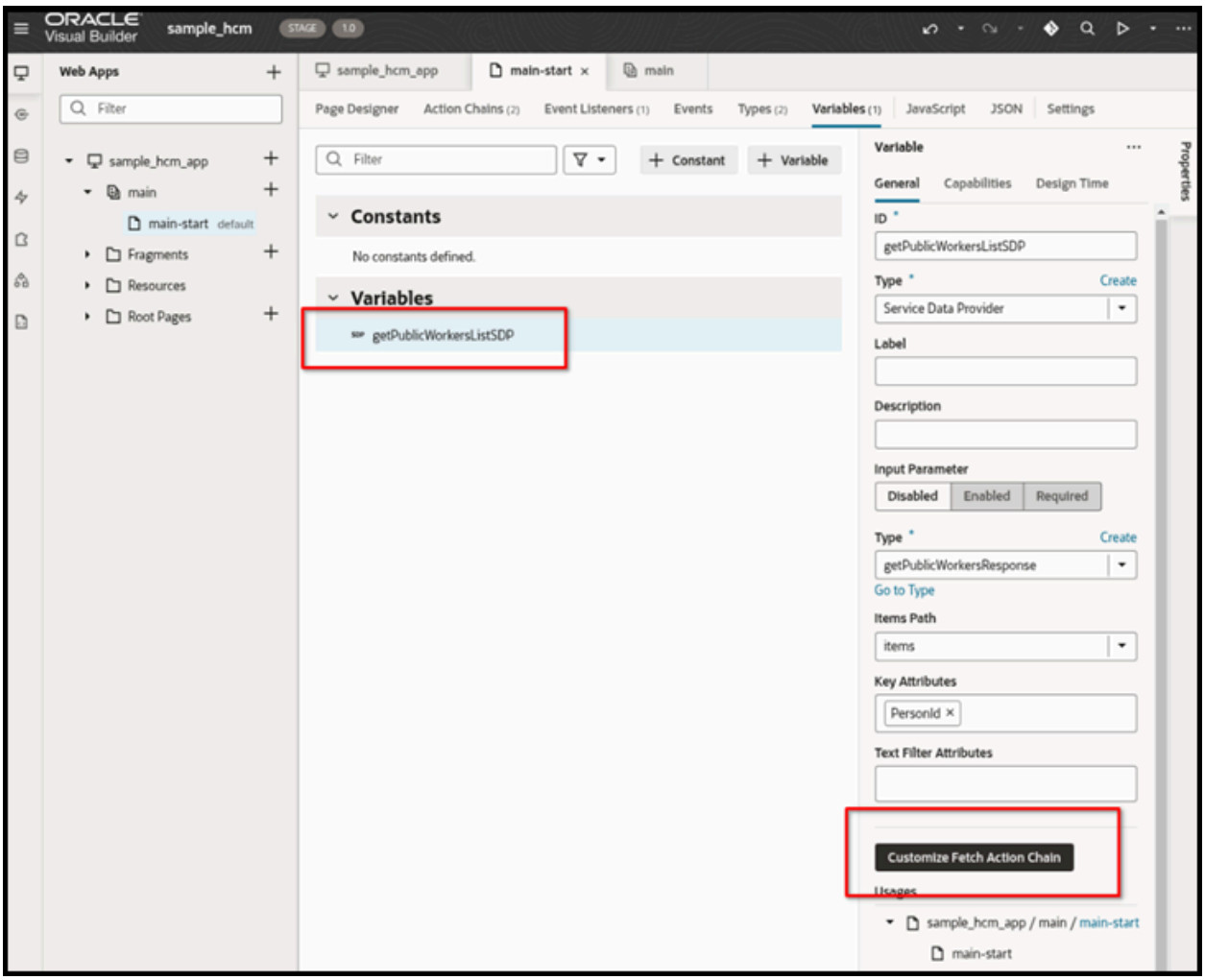


2. As the data source for the table, select the **Public Workers** service connection you created earlier.



3. Select the specific fields from the Public Workers endpoint structure that you want to display in the table. For example, PersonId, FirstName, LastName. Specify PersonId as the key field.

4. Navigate to the automatically created Service Data Provider variable, getPublicWorkersListSDP.



5. Click **Customize Fetch Action Chain**.
6. Within the getPublicWorkersFetch Action Chain, add the following code to the "Headers" section:

   ```
   headers: $application.variables.authHeaders,
   ```

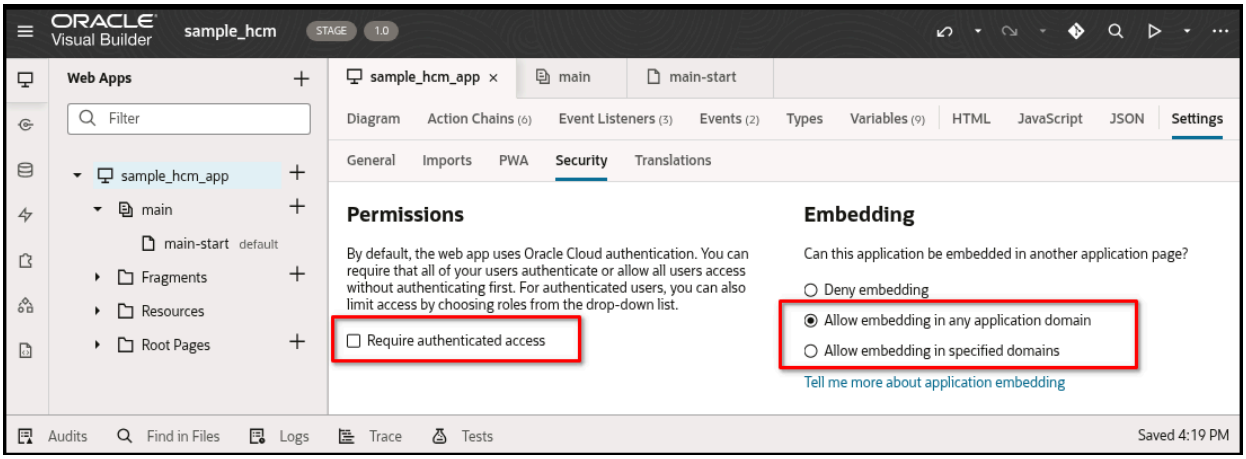
   This ensures that the authorization header (containing the JWT obtained via the Plugin Framework) is included when fetching data for the table.
7. To list 'getPublicWorkersFetch' Action Chain, use the following code:

```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class getPublicWorkersFetch extends ActionChain {

 /**
```

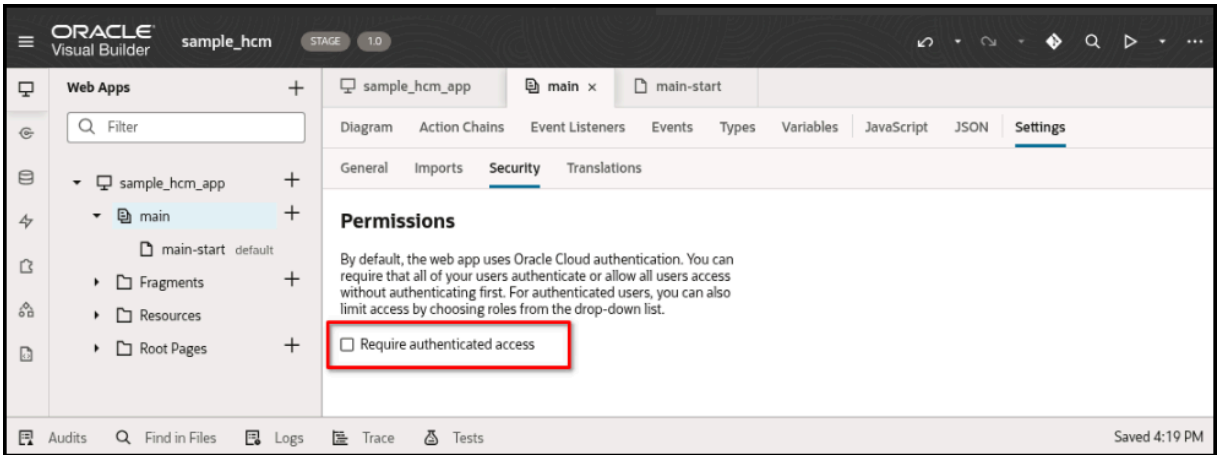
```
 * @param {Object} context
 * @param {Object} params
 * @param {{hookHandler:'vb/RestHookHandler'}} params.configuration
 */
 async run(context, { configuration }) {
 const { $page, $flow, $application, $constants, $variables } = context;
 const callRestEndpoint1 = await Actions.callRest(context, {
 endpoint: 'publicWorkers/getPublicWorkers',
 responseType: 'getPublicWorkersResponse',
 hookHandler: configuration.hookHandler,
 requestType: 'json',
 //-------------------------------------------------------------
 headers: $application.variables.authHeaders,
 //-------------------------------------------------------------
 });

 return callRestEndpoint1;
 }
 }

 return getPublicWorkersFetch;
});
```

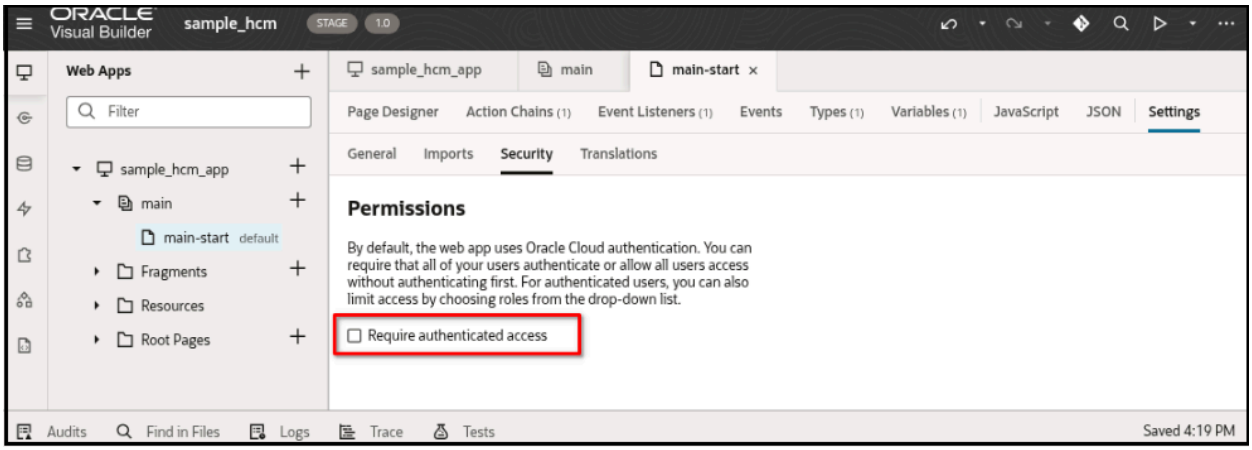
Configure Application Security

1. Navigate to the **Security** tab of your VBCS application.



2. Deselect the authentication access requirement at all three levels: Application, Flows, and Pages. This is necessary because the plugin will be loaded in a context where the user might not be directly authenticated with the VBCS application itself.





3. On the Security tab, configure the **Embedding** settings to allow embedding from any domain. This is crucial for the VBCS application to function correctly when embedded as an external plugin in Oracle Fusion Field Service.

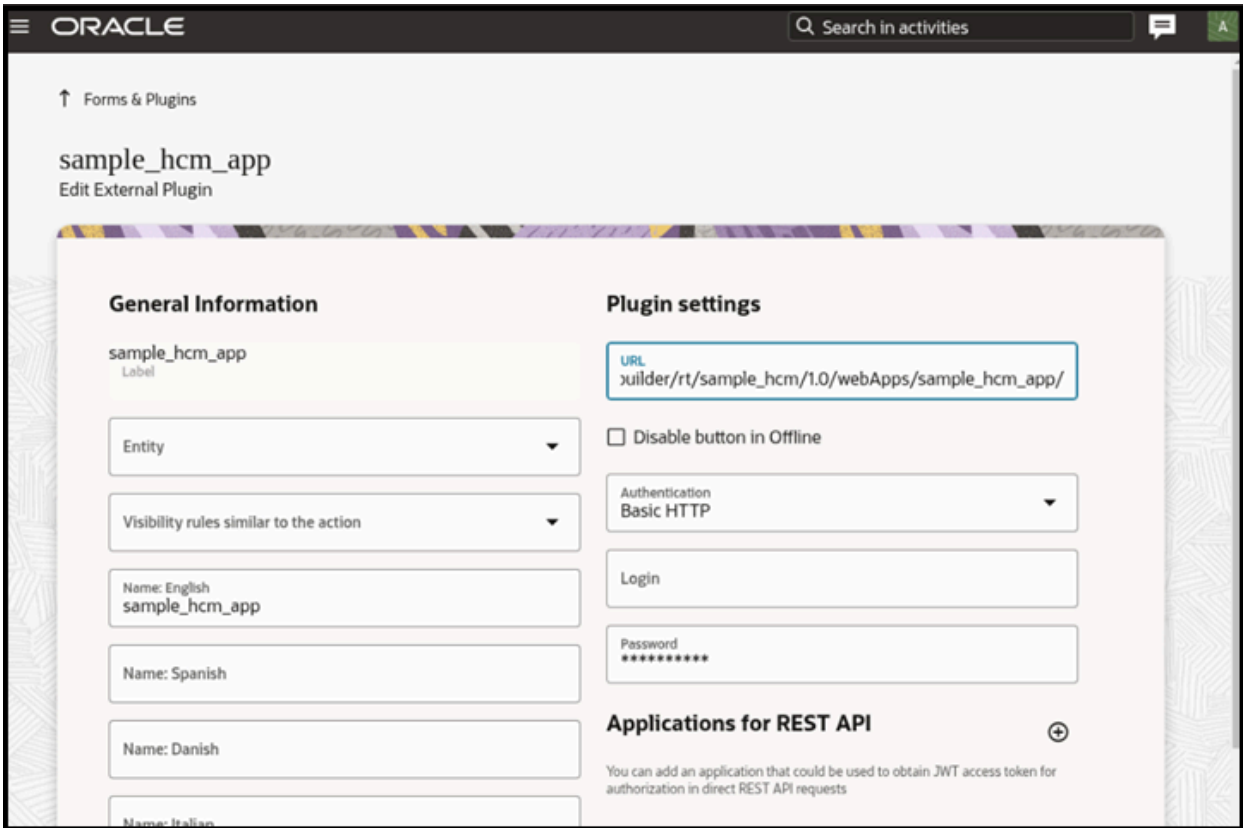
Remove Temporary Authorization Header

1. Navigate to the **Backend** section and select the custom backend (hcm in this scenario) you created.
2. Remove the temporary Authorization header that you added in the Add an HCM Backend section. The authorization header is used only to help the VBCS wizard obtain data from the REST API and configure columns and primary key. This header is only for design-time assistance and is no longer needed as the authorization will be handled dynamically through the Plugin Framework at runtime.

Stage VBCS Application for Oracle Fusion Field Service

1. Once your VBCS application is developed and tested, stage it to generate a URL that can be used as an external plugin in Oracle Fusion Field Service.

2. Next in Oracle Fusion Field Service, navigate to **Configuration → Forms & Plugins → Add Plugin → External Plugin** and open the external plugin that you have added.



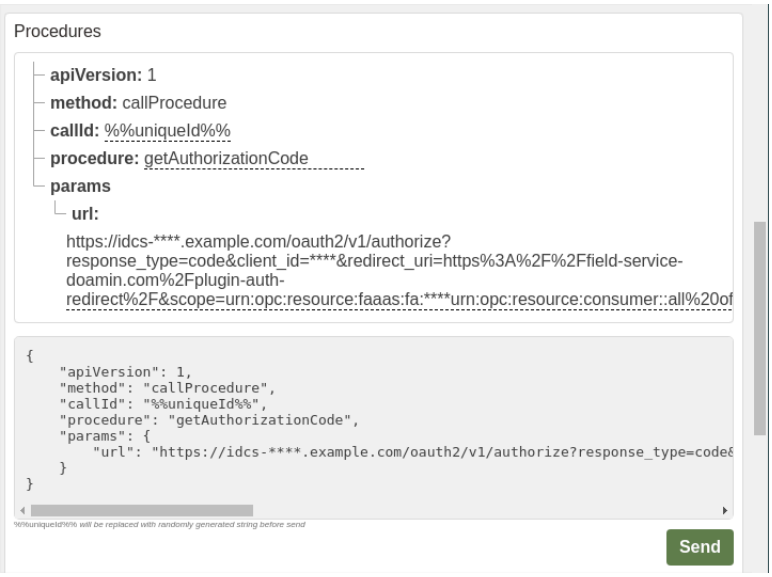
3. Configure the staged VBCS application as an external plugin within your Oracle Fusion Field Service environment. This involves providing the staged URL and configuring the necessary plugin parameters (for example, `getCodeEndpoint`, `getTokenEndpoint`, `clientId`, `scope`).

   VBCS application supports both **button parameters** and **plugin parameters**:

   a. getCodeEndpoint Example for IDCS: https://{idcsUrl}**/oauth2/v1/authorize**
   b. getTokenEndpoint Example for IDCS: https://{idcsUrl}**/oauth2/v1/token**
   c. clientId
   d. scope

Sample Plugin for Authorization Flow

The Sample Plugin can call the getAuthorizationCode procedure. When redirected to the Identity Provider, the plugin receives the callProcedureResult in response as shown below:



Obtain and Use a Refresh Token

This topic outlines how to obtain a refresh token and subsequently use it to acquire a new access token.

To obtain a refresh token, you need to explicitly request it during the initial authorization code retrieval. This is done by including the `"offline_access"` value in the `scope` parameter of the "getAuthorizationCode" procedure call.

The Plugin API message would then appear as follows:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "getAuthorizationCode",
 "callId": "d18243f2-e4f9-4cd2-a357-102fda444c6a",
 "params": {
 "url": "https://idcs-****.example.com/oauth2/v1/authorize?
response_type=code&client_id=****&redirect_uri=https%3A%2F%2Ffield-service-doamin.com%2Fplugin-auth-redirect
%2F&scope=urn:opc:resource:faaas:fa:****urn:opc:resource:consumer::all%20offline_access"
 }
}
```

After the receiving code and requesting access token in the usual way (request to "/oauth2/v1/token") the response looks like:

```
{
 "access_token": "eyJ4NXQjUzI1NiI6Ink5bm...6VBDe_Utj5C0kA",
 "token_type": "Bearer",
 "expires_in": 3349,
 "refresh_token": "AgAgZDliM2M1OGUwY...yVHFGigEP5AB7zfYQ=="
}
```

To get a new access token by using the refresh token you need to call the following request:

```
CLIENT_ID='****'
CLIENT_SECRET='****'
REFRESH_TOKEN='****'
curl --noproxy '*' --url 'https://idcs-****.com/oauth2/v1/token' -X POST \
```

```
 -H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' \
 -u "$CLIENT_ID:$CLIENT_SECRET" \
 -d grant_type=refresh_token \
 -d refresh_token="$REFRESH_TOKEN"
```

As an alternative the credentials could be sent in data fields:

```
CLIENT_ID='****'
CLIENT_SECRET='****'
REFRESH_TOKEN='****'
curl --noproxy '*' --url 'https://idcs-****.com/oauth2/v1/token' -X POST \
 -H 'Content-Type: application/x-www-form-urlencoded;charset=UTF-8' \
 -d grant_type=refresh_token \
 -d client_id="$CLIENT_ID" \
 -d client_secret="$CLIENT_SECRET" \
 -d refresh_token="$REFRESH_TOKEN"
```

The same request using Fetch API (JS):

```
fetch("https://idcs-****.com/oauth2/v1/token", {
 method: 'POST',
 headers: {
 'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8'
 },
 body: new URLSearchParams({
 grant_type: "refresh_token",
 client_secret: "****",
 client_id: "****",
 refresh_token: "****"
 })
})
.then(resp => resp.json())
.then(json => console.log(json));
```

For more information: see *https://docs.oracle.com/en/cloud/paas/identity-cloud/rest-api/ACWebServerAppAuth.html*

Troubleshooting

This section describes the troubleshooting tips that can help resolve common issues encountered when integrating your VBCS application with Oracle Fusion Field Service.

1. **HTTP 403 Error (Forbidden)**: If you encounter a "403 Forbidden" error, it might indicate that your user account in Fusion does not have the necessary permissions. Ensure that the ORA_HRC_HUMAN_CAPITAL_MANAGEMENT_INTEGRATION_SPECIALIST_JOB role is assigned to your Fusion user. You can check and assign this role in Fusion through **Tools -> Security Console -> Users**.
2. **CORS Error**: If you see a CORS (Cross-Origin Resource Sharing) error, the browser is preventing the VBCS application from making requests to the REST API due to security restrictions. Contact your Oracle Fusion Field Service administrator to verify the **ORA_CORS_ORIGINS** profile option code in Fusion: My Enterprise -> Setup and Maintenance → Tasks (visible in the Right Side of the screen) -> Search -> **Manage Administrator Profile Values** task -> Add full URL (**including "https"** prefix and **without trailing slash)** to Profile Value field. The value is a domain from which the plugin is loaded. For more information, see *https://docs.oracle.com/en/cloud/saas/fusion-service/facoe/c_chat_configure_for_cors.html*

   The domains that should be listed in the CORS policy field:

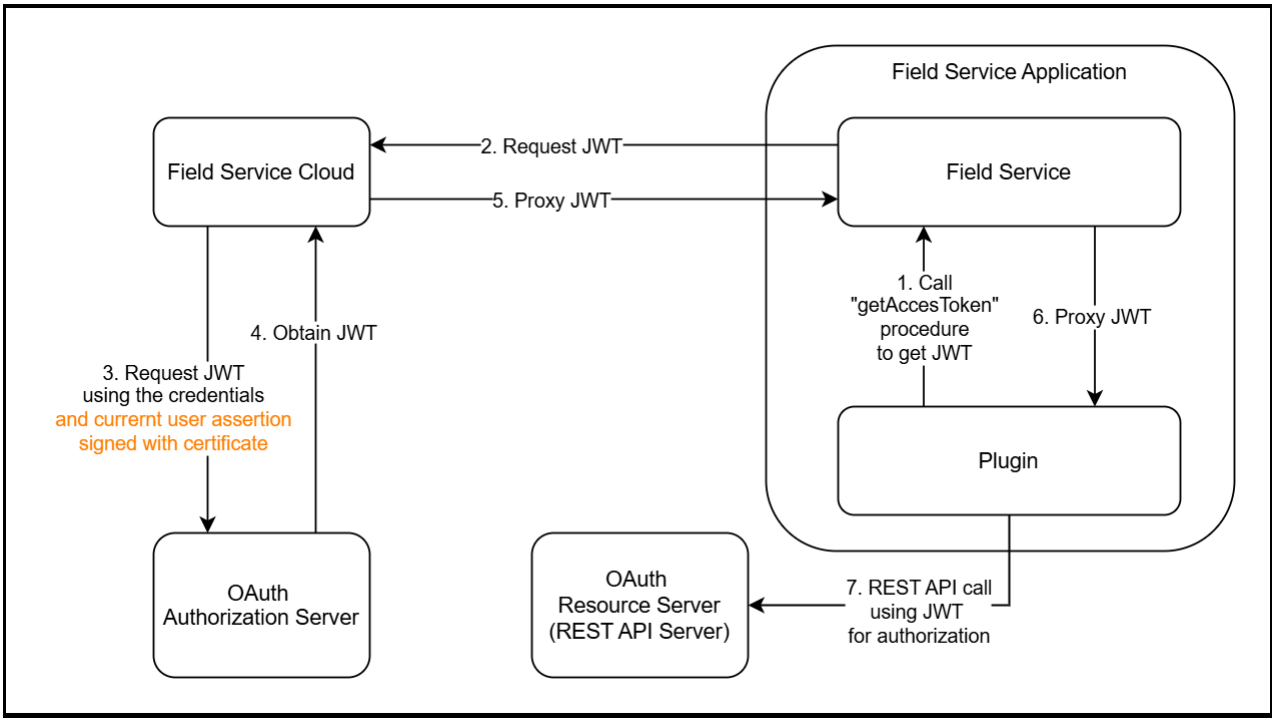
   a. for VBCS plugins - domain of VBCS application
   b. for standard plugins - **plugins-1-**{FS domain}, use environment name in the domain, for example, ofsc-******.test instead of alias for hosted plugins - **plugins-0-**{FS domain}, use environment name in the domain, for example, ofsc-******.test instead of alias

c. for requests from browser's console - name of the domain where request is called

OAuth User Assertion Flow (getAccessToken procedure)

You can use the OAuth User Assertion Flow to obtain an access to REST API on behalf of user that is logged in to Oracle Fusion Field Service using Single Sign On.

The diagram below illustrates the OAuth User Assertion Grant Flow, detailing the process of obtaining access to the REST API on behalf of a user logged in through Single Sign-On.



Advantages:

- Associates REST API access with specific user privileges.
- Suitable for scenarios requiring differentiation based on user credentials for REST API calls.

Process Workflow Summary:

1. Add the OAuth User Assertion application to the **Configuration → Applications** page.
2. Download the Field Service certificate. This certificate is used for upload to the Identity Provider side when you configure the IDCS application.
3. Configure the application on the Identity Provider to support the OAuth User Assertion Flow.
4. Enter credentials (Client ID, Scope, Identity Provider endpoint) on the Oracle Fusion Field Service side.
5. Add the application to the plugin using the **Edit Plugin** page.
6. Call the getAccessToken procedure from the plugin with this application in the procedure parameters.
7. Obtain a JWT access token in the procedure response.
8. Use the JWT access token for REST API request authorization.

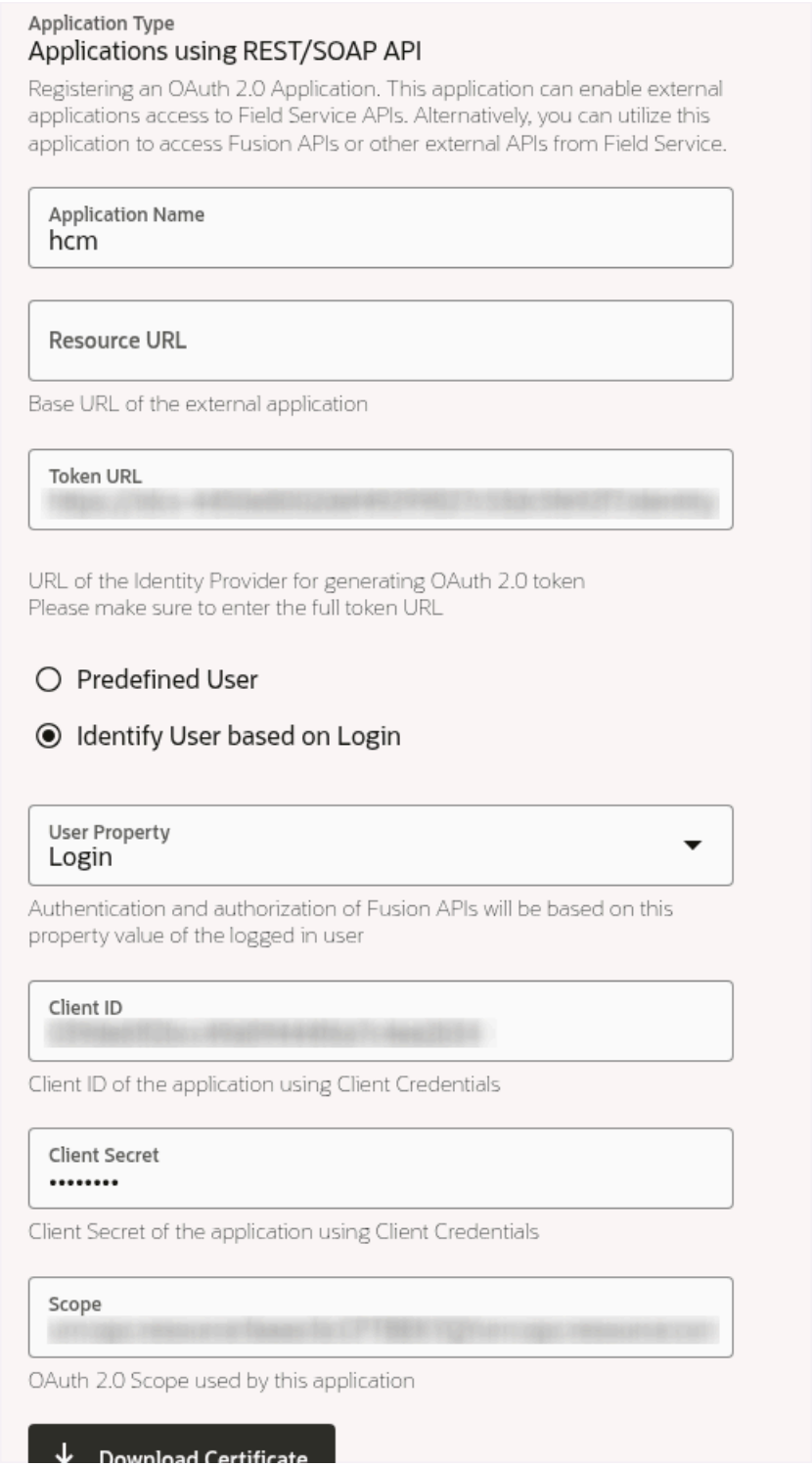
Note: This flow may not work with certain Identity Providers that require additional parameters not outlined in the RFC standard. For example: Microsoft Identity Platform requires the "requested_token_use" parameter in the token request. In that case, the OAuth Authorization Code Grant Flow could be used as an alternative.

Oracle Fusion Field Service Application Configuration for OAuth User Assertion

This topic outlines the steps to create and configure a Field Service application using the OAuth User Assertion flow to obtain an access token.

1. Navigate to **Configuration → Applications**.

2. Click **Add Application**. The Add Application page appears.



3. Select **Applications using REST/SOAP API**. You will be prompted to enter a name for this application; provide a descriptive name.

4. From the available application security types, select **OAuth User Assertion**. This indicates that this application will use a JWT assertion to obtain an access token from the Identity Provider.
5. In the **Resource URL** field, specify the URL that represents the OAuth 2.0 Resource Server (the server that provides REST data). It will be included in the "applications" section during the "init" method to avoid hardcoding the URL in the plugin.
6. In the **Token URL** field, enter the URL of the OAuth 2.0 Authorization Server (Identity Provider) that issues access tokens. For example, `https://{idcsUrl}/oauth2/v1/token` .

   > **Note:** In some cases, the OAuth 2.0 Resource Server and Authorization Server share the same domain.

7. Select which username will be included in the JWT assertion.
   - **Predefined User**: Select this option to use a fixed, predefined username. This is often used for testing purpose.
   - **Identify User Based on Login** : Select this option to use the username derived from a property of the currently logged-in Field Service user (commonly the "Login" property).
8. Enter the Identity Provider Details.
   a. **Client ID**: Obtained from the Identity Provider configuration while creating the application in the Authorization Server.
   b. **Client Secret:** Also retrieved from the Identity Provider configuration.
   c. **Scope:** A space-separated string used to restrict access. For example:
   `urn:opc:resource:fusion:xxxxxxxx:field-service`
9. Click **Download Certificate** to download the certificate. This certificate is used by the Identity Provider to verify the signature of the JWT assertion sent by Oracle Fusion Field Service. You must import this certificate into the Identity Provider's configuration for the integrated application.

IDCS Configuration for OAuth User Assertion

This topic describes how to configure an integrated application within Oracle Identity Cloud Service (IDCS) to enable the OAuth User Assertion flow and issue access tokens for use by Oracle Fusion Field Service.

1. Navigate to the identity domain configuration in IDCS.
2. Select the **Integrated Applications** section from the left pane.
3. Click **Add Application** at the top of the page. The **Add application** dialog box appears.
4. In the Add application dialog box, select **Confidential Application** and click **Launch Workflow**. The **Add Confidential Application** dialog box appears.
5. Provide a name for your new application.
6. Click **Submit**.
7. Next, click **Edit OAuth configuration** to proceed to the **Edit OAuth configuration** step.
8. In the Client Configuration section, select **Configure this application as a client now**.
9. Select the **JWT Assertion** grant type in the Authorization section. Leave all other grant types unchecked. In this example, only one grant type is used, but real integrations may involve multiple grant types for a single application.
10. In the **Client Type** section, select **Trusted**. Trusted clients can generate self-signed user assertions using the Field Service certificate.
11. Import the Field Service signing certificate, which can be downloaded from: **Configuration → Applications → Add Application (OAuth User Assertion support) → Download Certificate.**
12. In the **Allowed Operations** section, optionally select **On behalf of**. This allows the client application to access endpoints the user can access, even if the client doesn't have direct access.

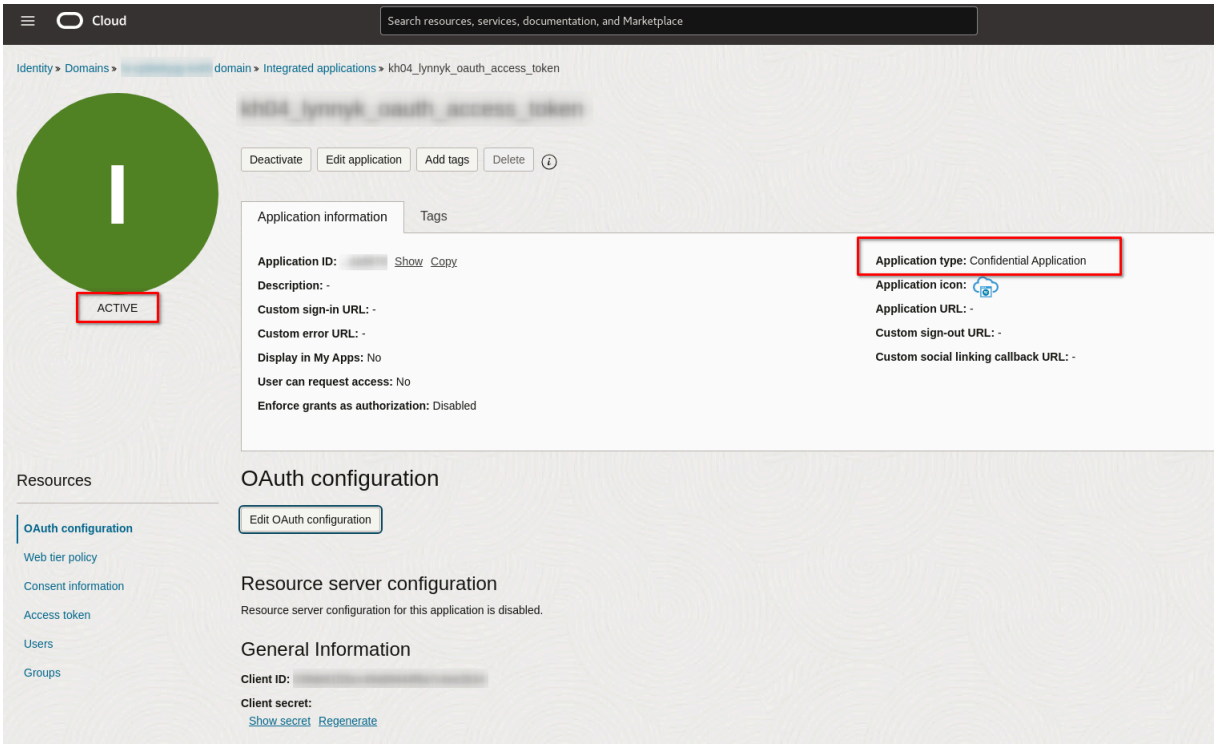
13. In the **Token Issuance Policy** section, select **Add resources** if you want your application to access the APIs of other applications.
14. In the **Resources** section, click **Add Scope**. A list of applications appears in the Add scope dialog box.
15. Select the scope of the target resource, such as Fusion Applications Cloud Service, and then click **Add**.

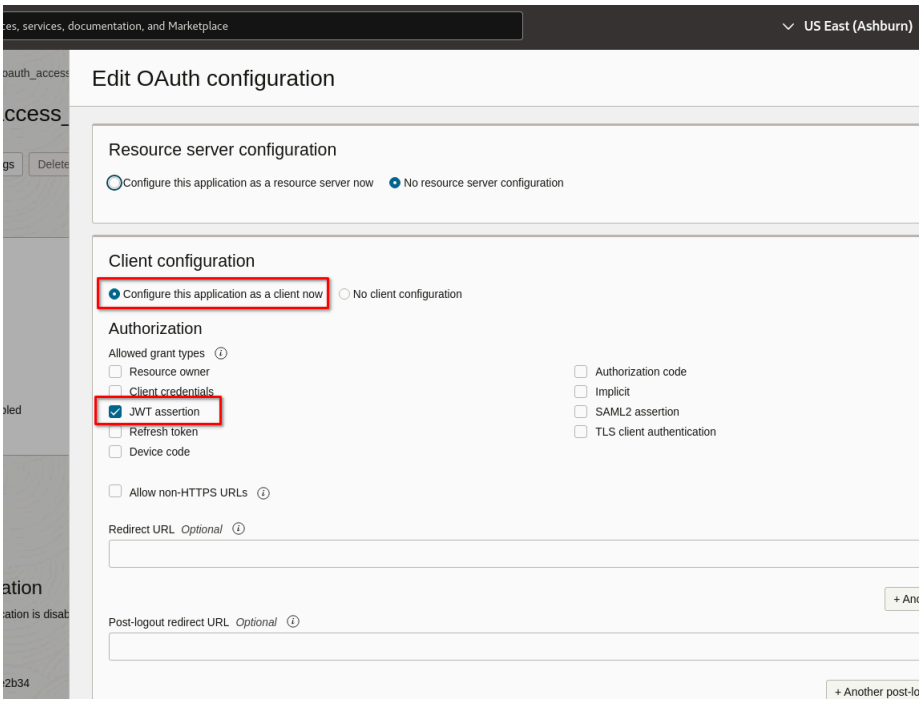
    The selected application is added to the Resource scope.

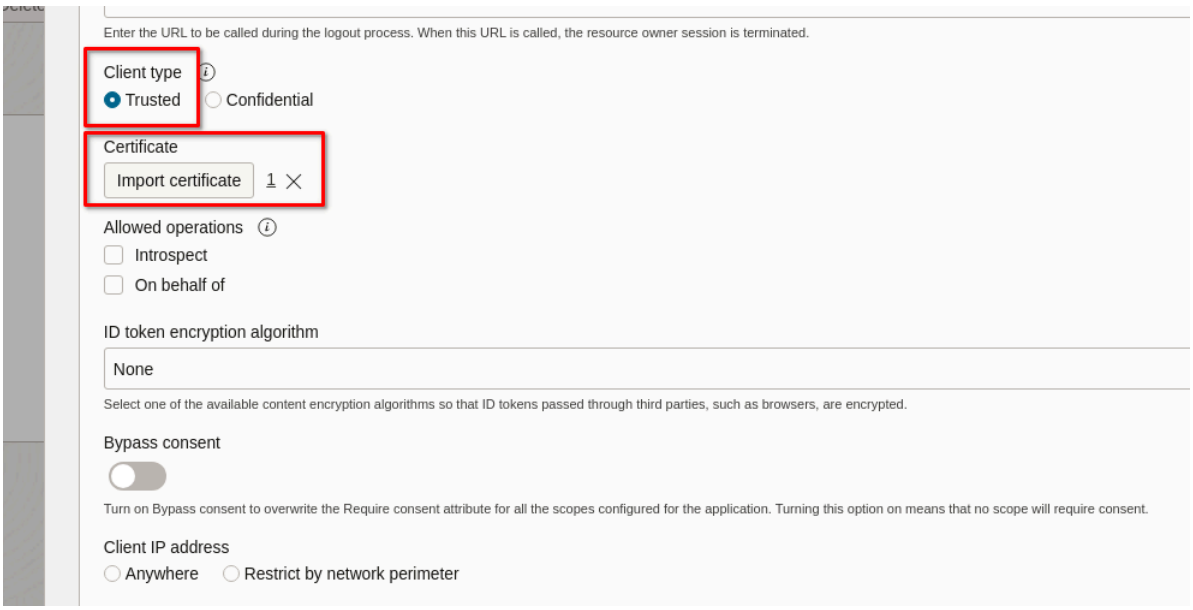
    > **Note:** If Fusion Applications Cloud Service is not listed as a resource, it indicates that Fusion Service is not linked to the IDCS domain. In this case, you must create the integrated application in the domain linked to Fusion Service.

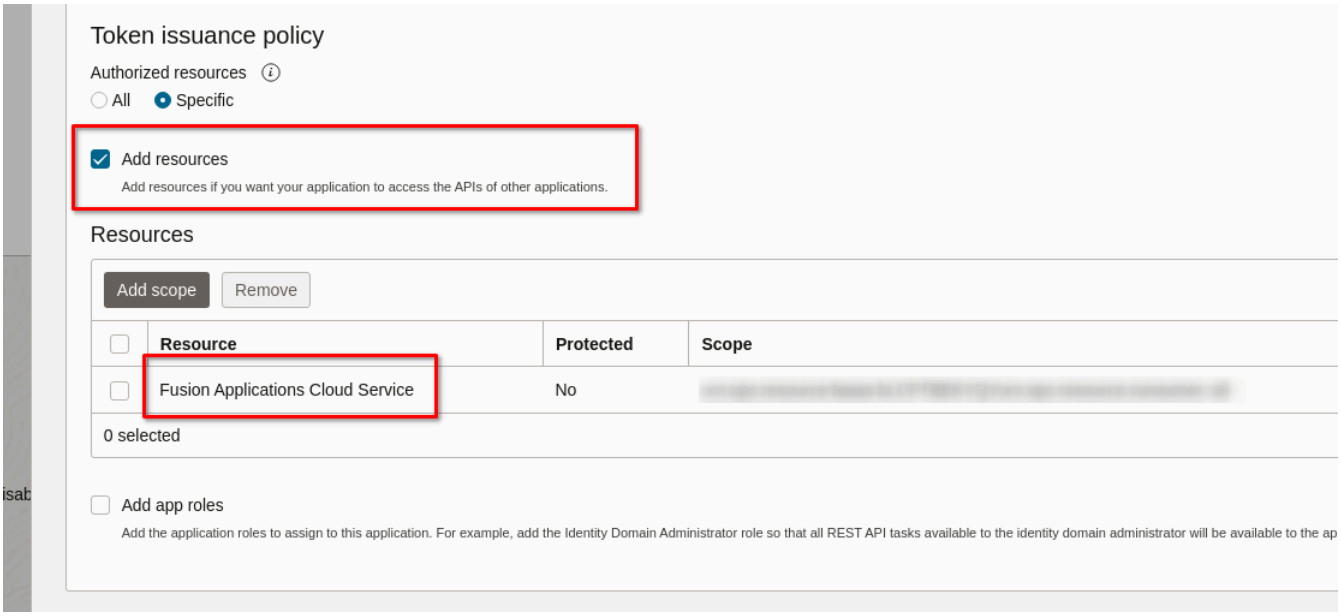
16. Click **Finish** to complete the creation of the integrated application.
17. After creation, ensure that you activate the newly created IDCS application.

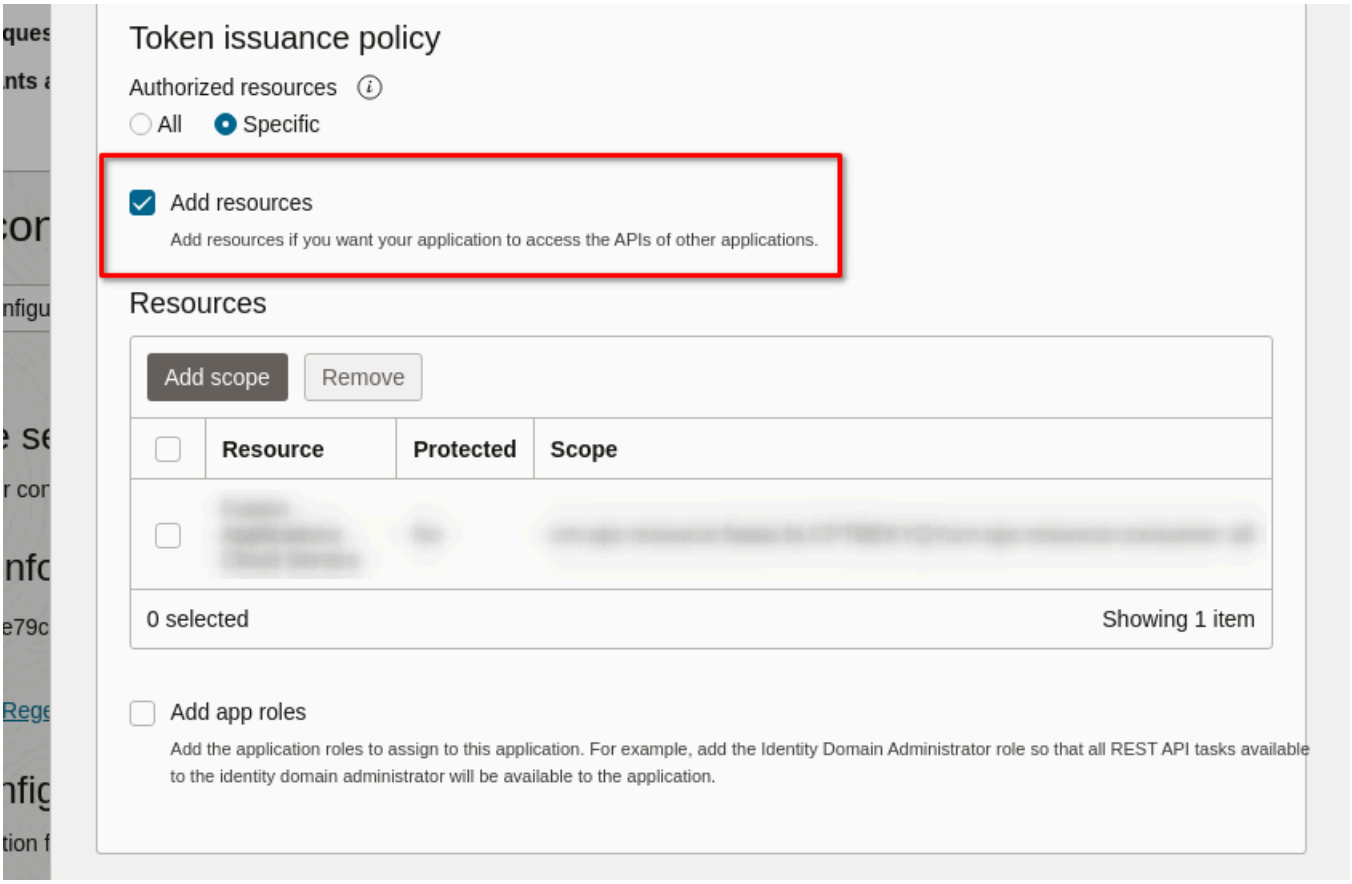
18. Once activated, your application appears as configured and ready for integration.






Enter the URL to be called during the logout process. When this URL is called, the resource owner session is terminated.
Client type
Trusted
Confidential
Certificate
Import certificate
1
Allowed operations
Introspect
On behalf of
ID token encryption algorithm
None
Select one of the available content encryption algorithms so that ID tokens passed through third parties, such as browsers, are encrypted.
Bypass consent
Turn on Bypass consent to overwrite the Require consent attribute for all the scopes configured for the application. Turning this option on means that no scope will require consent.
Client IP address
Anywhere
Restrict by network perimeter


Token issuance policy
Authorized resources
All
Specific
Add resources
Add resources if you want your application to access the APIs of other applications.
Resources
Add scope
Remove
Resource
Protected
Scope
Fusion Applications Cloud Service
No
0 selected
Add app roles
Add the application roles to assign to this application. For example, add the Identity Domain Administrator role so that all REST API tasks available to the identity domain administrator will be available to the ap



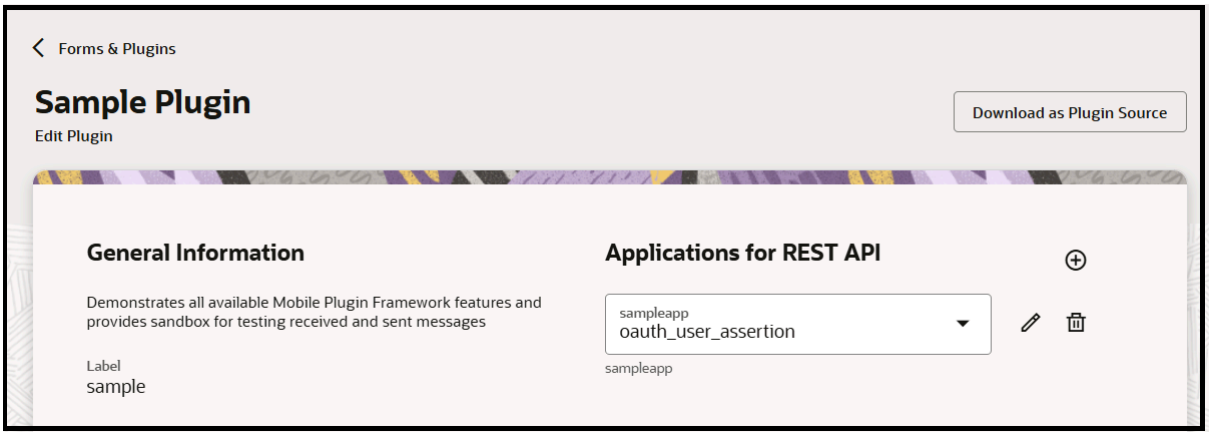
Plugin Configuration to Connect Oracle Fusion Field Service Application

This topic explains how to link the Oracle Fusion Field Service application configured for OAuth User Assertion to a plugin within Oracle Fusion Field Service.

To connect the Oracle Fusion Field Service Application to a plugin:

1. Navigate to **Configuration > Displays > Forms and Plugins**. Or, click the **Configurations >Users Type> Screens** and then select the required plugin.
2. Select the plugin from the available list that you would like to connect.
3. Click **Edit** from the **Action** menu. The **Edit Plugin** page appears.
4. In the **Applications for Rest API** section, click **Add**.
5. Select the Oracle Fusion Field Service application you created in *Oracle Fusion Field Service Application Configuration for OAuth User Assertion* from the drop-down list of available applications for a specific

application key (for example, "oauth_user_assertion_application"). This establishes the connection between the plugin and the configured application.



6. Click **Update**.

Sample Plugin to Obtain Access Token

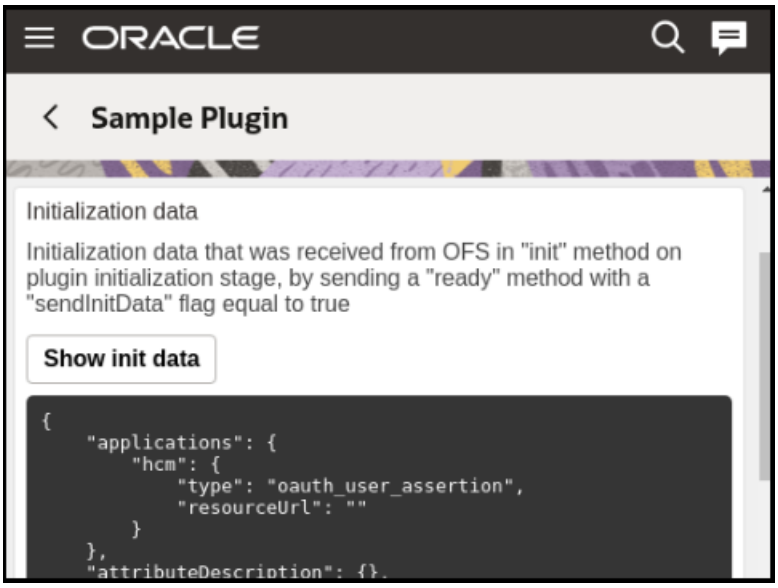
This topic describes how a plugin, such as the Sample Plugin, can use the configured Oracle Fusion Field Service application to obtain an access token using the getAccessToken procedure.

Initial Application List

Upon initialization, the plugin receives a message containing a list of available applications. This message includes the **resourceUrl** field for each application.

The **resourceUrl** is obtained from the plugin's configuration and is used by the Sample Plugin in subsequent API requests.

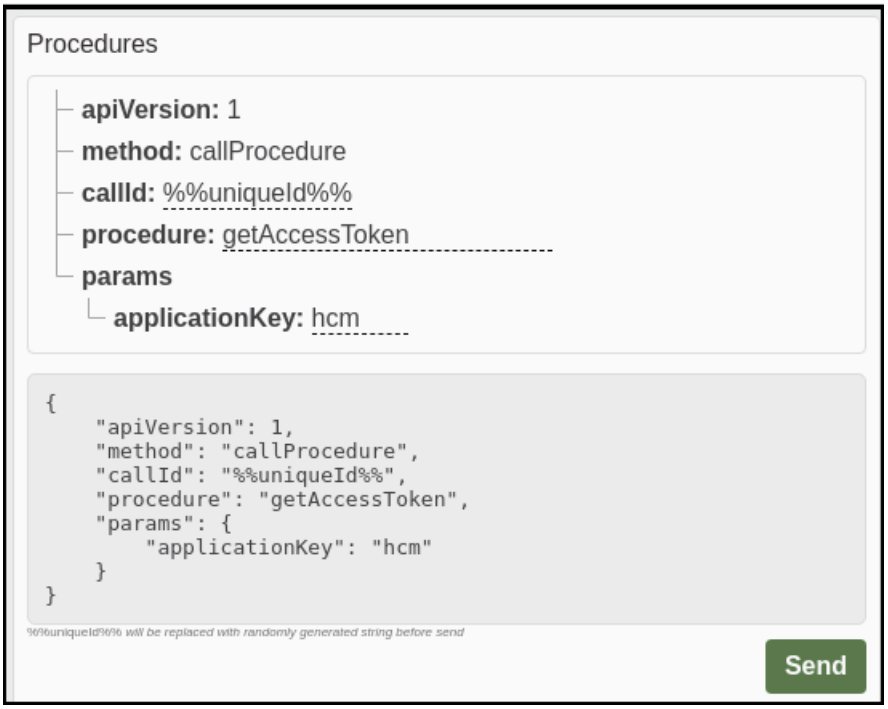
The following screenshot illustrates the Initialization data, showcasing how the plugin obtains and employs the resourceUrl field from its configuration during the initialization process.



Obtain an Access Token

To acquire an access token using the Sample Plugin, you need to utilize the **Procedures** functionality as shown in the screenshot below. You can

- Send a `callProcedure` method call to the plugin.
- Within the `callProcedure` call, specify `getAccessToken` as the desired procedure.



Retrieve the Token

Once the callProcedure with getAccessToken is processed, the resulting access token will be available in the `callProcedureResult` response method as shown below:

Validate the Token by Fetching Data

With the access token obtained, you can now fetch data from a REST API to verify its validity.

```
fetch("https://fusion-instance.com/hcmRestApi/resources/latest/workers", {
 headers: {
 Authorization: "Bearer <accessToken>"
 }
})
.then(resp => resp.json())
.then(json => console.log(json));
```

VBCS Application as an Oracle Fusion Field Service External Plugin

This topic provides information on how to create a VBCS application that acts as an Oracle Fusion Field Service plugin and obtains data from a REST API using the OAuth User Assertion flow.

First, you need to configure the IDCS application which gives you the ability to authorize using the OAuth Authorization Code flow.

1. Create a VBCS application.
2. Configure the plugin in Oracle Fusion Field Service.
3. Execute the complete workflow: open the VBCS plugin, authorize, and retrieve REST API data from Fusion.

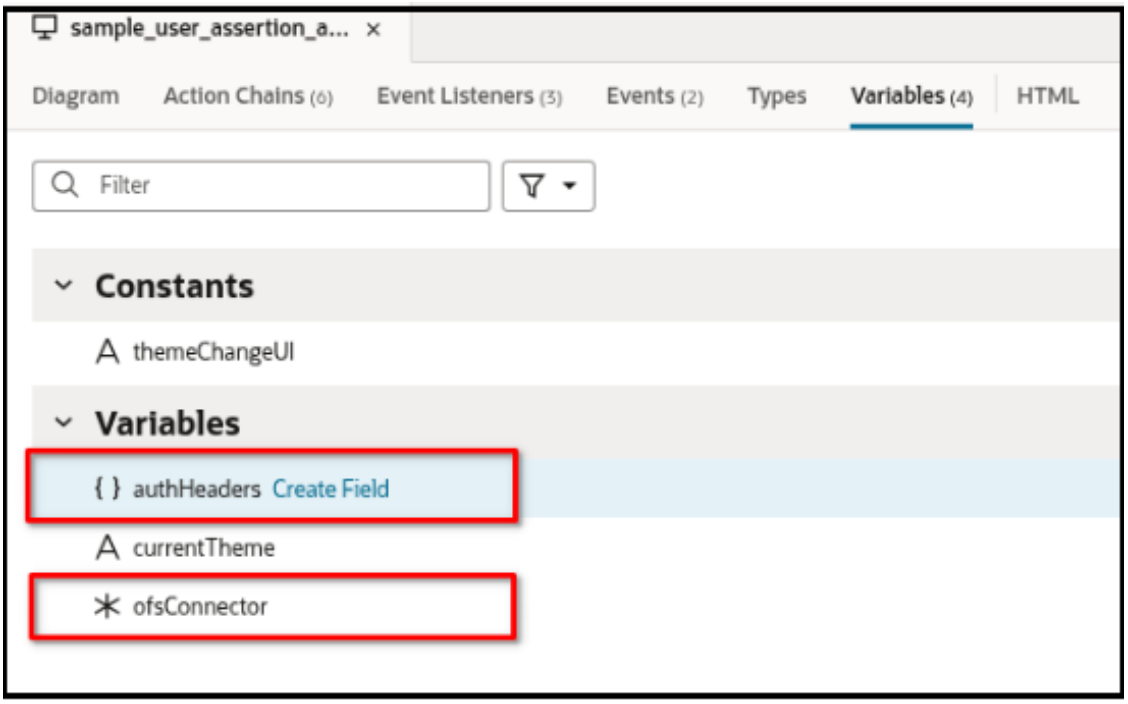
Create a VBCS Application

This topic provides a step-by-step guide to create a VBCS application that acts as an Oracle Fusion Field Service plugin and obtains data from a REST API using the OAuth User Assertion flow.

To create a new VBCS application:

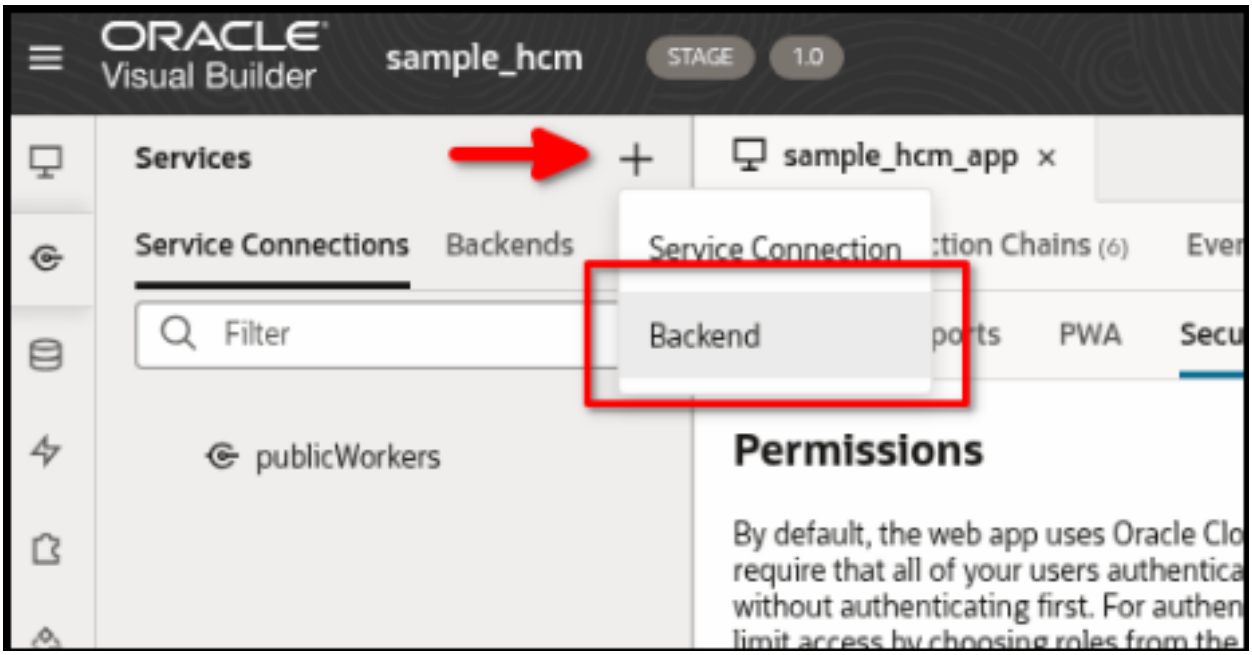
1. Open your VBCS environment.
2. Create a new VBCS application and specify a suitable name and ID for your application.
3. Within your newly created VBCS application, navigate to the **Variables** section.
    a. Create the necessary variables that your application will use during its operation. The specific variables required will depend on the logic of your plugin. Use the following code to add the required variables:

```
"variables": {
 "authHeaders": {
 "type": "object"
 },
 "ofsConnector": {
 "type": "any"
 },
 ...
}
```

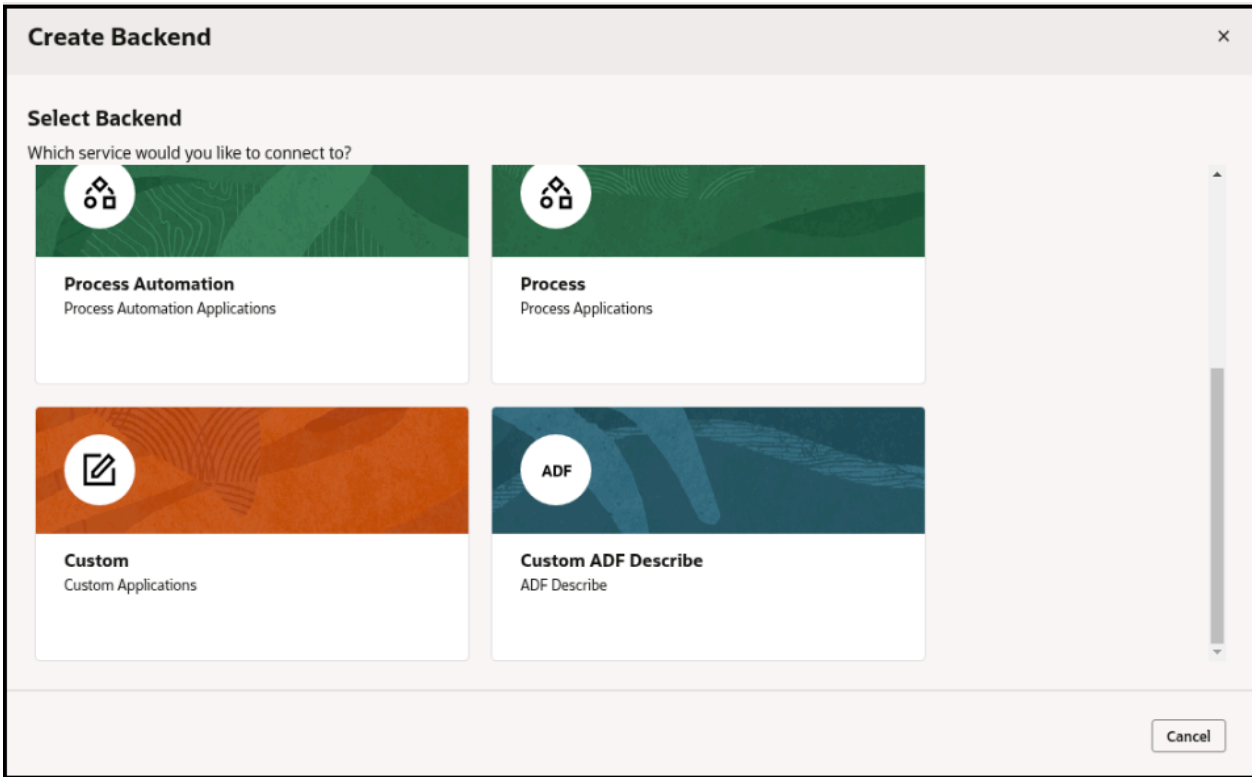


Add a Backend

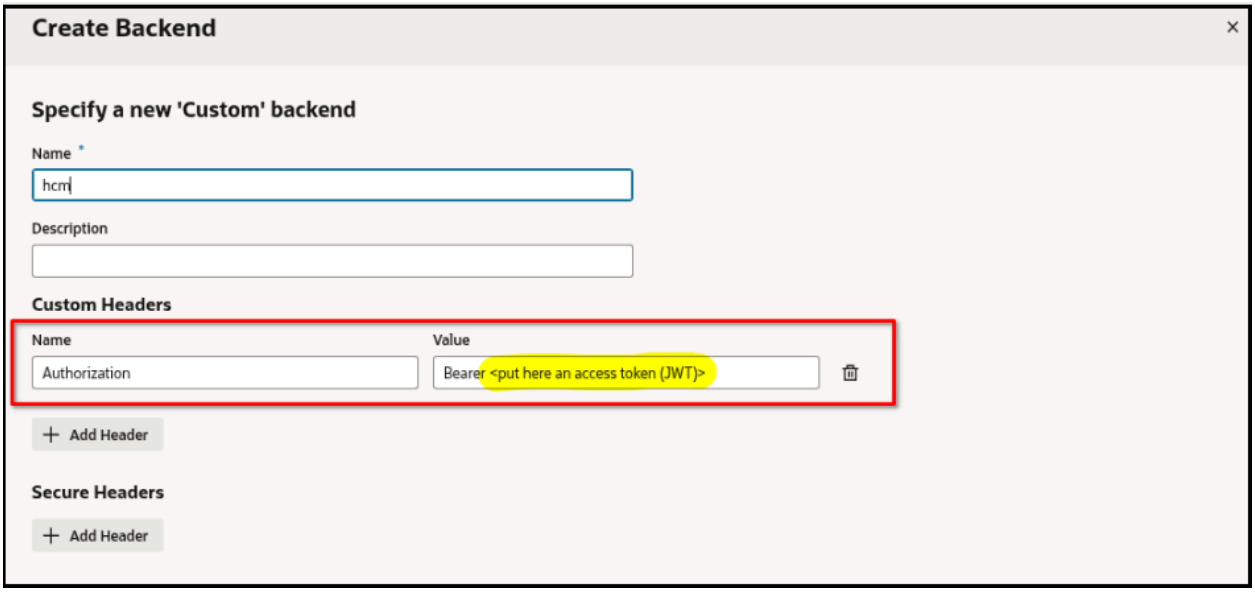
1. Navigate to the **Backend** section in VBCS.
2. Click the button to create a new backend.



3. Select **Custom** as the backend type.



4. Enter a name for your custom backend. For example, hcm.

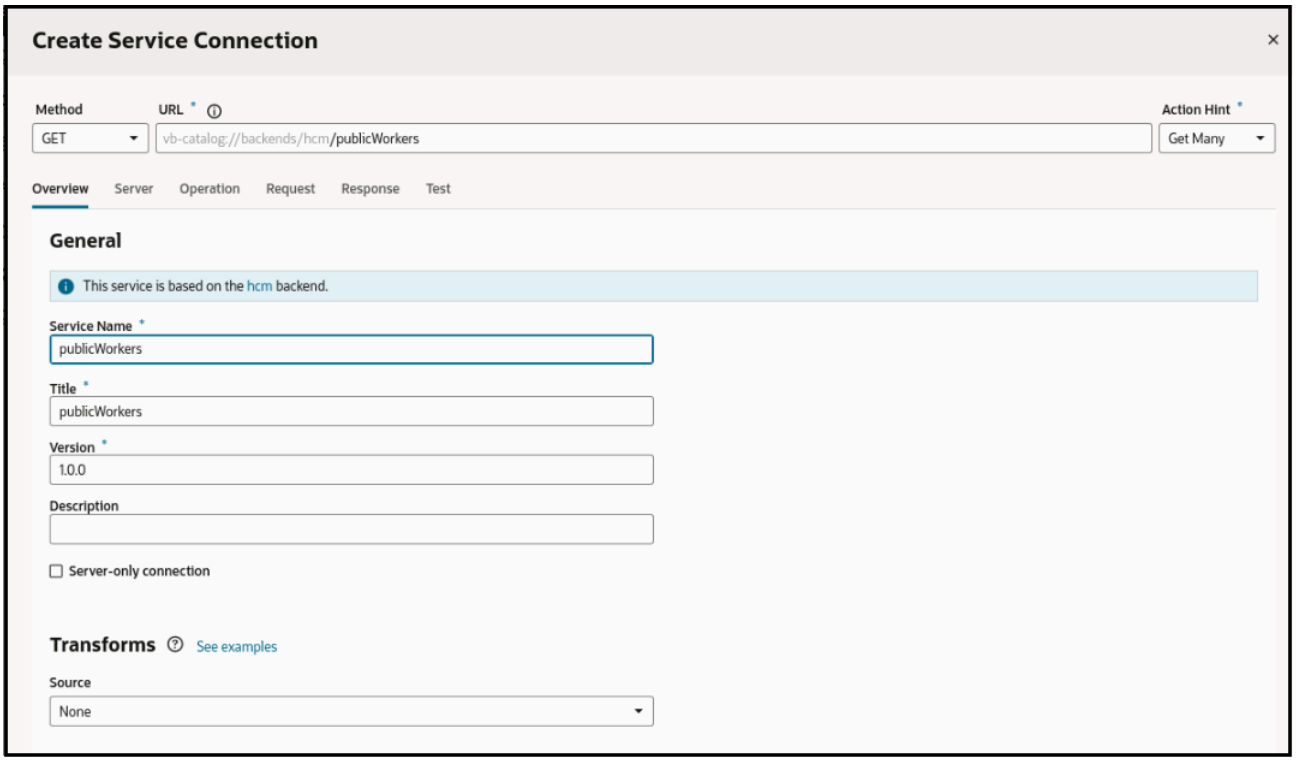


5. To allow VBCS to understand the structure of the REST API during development:
   a. Add a custom header named Authorization.
   b. Set the Value of this header to Bearer xxxx, where xxxx is a valid JWT (obtained through a tool like the Simple Authorization Plugin described in previous documentation).

   > **Note:** This header serves a temporary function to facilitate the design process and will be removed in a subsequent phase. The JWTs have a finite lifespan and will require periodic updates during development.

Add a 'Public Workers' Service Connection

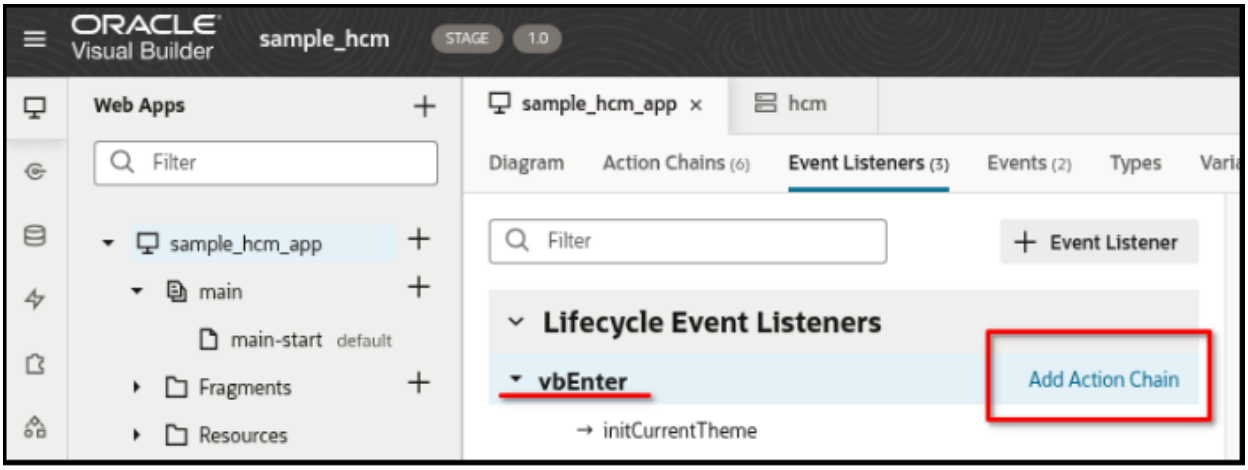
1. Next, navigate to the **Service Connections** section.
2. Click the button to add a new service connection.
3. Click **Select from Catalog**.
4. In the **Custom** section, select the custom backend you just created (e.g., hcm).
5. Select **Define by endpoint**.
6. Set the **Service Name** and **Title** to publicWorkers.



7. Set the URL suffix to /publicWorkers. This will be appended to the base URL of your HCM backend. In this example, the plugin will request data from Public Workers HCM REST API. For more information, see *https://docs.oracle.com/en/cloud/saas/human-resources/24c/farws/op-publicworkers-get.html*

Implement Plugin Framework Initialization

1. Navigate to the main page or the relevant flow where the plugin will be used.



2. Open the **Event Listeners** tab for the page.
3. Select the **vbEnter** event.
4. Click **Add Action Chain** to create a new Action Chain **_init** associated with this event.
5. Within this _init Action Chain, add the following code :

```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class _init extends ActionChain {

 /**
 * @param {Object} context
 */
 async run(context) {
 const { $application, $constants, $variables } = context;
 //---------------------------------------

 class OfsConnector {
 constructor(params) {
 const {
 apiVersion = 1,
 onInit = (jsonData) => {},
 onOpen = (jsonData) => {},
 onCallProcedureResult = (jsonData) => {},
 onError = (jsonData) => {
 alert(jsonData);
 }
 } = params;


 this.API_VERSION = apiVersion;

 this.onInit = onInit.bind(this);
 this.onOpen = onOpen.bind(this);
```

```
this.onCallProcedureResult = onCallProcedureResult.bind(this);
this.onError = onError.bind(this);

this.TRANSMIT_METHODS = {
METHOD_READY: 'ready',
METHOD_INIT_END: 'initEnd',
METHOD_CLOSE: 'close',
METHOD_UPDATE: 'update',
METHOD_CALL_PROCEDURE: 'callProcedure',
METHOD_SLEEP: 'sleep'
};

this.RECEIVE_METHODS = {
METHOD_ERROR: 'error',
METHOD_INIT: 'init',
METHOD_OPEN: 'open',
METHOD_UPDATE_RESULT: 'updateResult',
METHOD_CALL_PROCEDURE_RESULT: 'callProcedureResult',
METHOD_WAKEUP: 'wakeup'
};

window.addEventListener("message", this.onPostMessage.bind(this), false);
}

/**
* @param {Object} dataToSend
* @returns {void}
*/
sendMessage(dataToSend) {
const originUrl = this.constructor._getOriginUrl();
const origin = originUrl ? this.constructor._getOrigin(originUrl) : '*';

dataToSend.apiVersion = this.API_VERSION;

parent.postMessage(dataToSend, origin);
}


onPostMessage(event) {
// Accept only external messages
if (event.source === window) {
return;
}

const jsonData = event.data;

if (!jsonData) {
this.onError('Received message without data: ' + jsonData);

return;
}

switch (jsonData.method) {
case this.RECEIVE_METHODS.METHOD_INIT:
this.onInit(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_OPEN:
this.onOpen(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_CALL_PROCEDURE_RESULT:
this.onCallProcedureResult(jsonData);
break;

case this.RECEIVE_METHODS.METHOD_ERROR:
```

```
this.onError('Received message with error method: ' + jsonData);
break;

default:
this.onError('Received message with unsupported method: ' + jsonData);
}
}

static generateRandomString (length) {
return btoa(String.fromCharCode.apply(null, window.crypto.getRandomValues(new Uint8Array(length))))
.replaceAll('=', '')
.replaceAll('/', '')
.replaceAll('+', '').substr(0, length);
}

static _getOrigin(url) {
if (typeof url === 'string' && url !== '') {
if (url.indexOf("://") > -1) {
return (window.location.protocol || 'https:') + url.split('/')[2];
} else {
return (window.location.protocol || 'https:') + url.split('/')[0];
}
}

return '';
}

static _getOriginUrl() {
if (document.referrer) {
return document.referrer;
}

if (document.location.ancestorOrigins && document.location.ancestorOrigins[0]) {
return document.location.ancestorOrigins[0];
}

return null;
}
}

//---------------------------------------

const onInit = function(jsonData) {
if (jsonData.origin) {
localStorage.setItem('authRedirectOrigin', jsonData.origin || '');
}

this.sendMessage({
method: this.TRANSMIT_METHODS.METHOD_INIT_END
});
};

const onOpen = function(jsonData) {
let openParams = jsonData.openParams || {};
let securedData = jsonData.securedData || {};

// getCodeEndpoint
// Example: https://{idcsUrl}/oauth2/v1/authorize

// getTokenEndpoint
// Example: https://{idcsUrl}/oauth2/v1/token

$application.variables.authClientId = openParams.clientId || securedData.clientId || '';
$application.variables.authScope = openParams.scope || securedData.scope || '';
$application.variables.authGetCodeEndpoint = openParams.getCodeEndpoint || securedData.getCodeEndpoint
|| '';
```

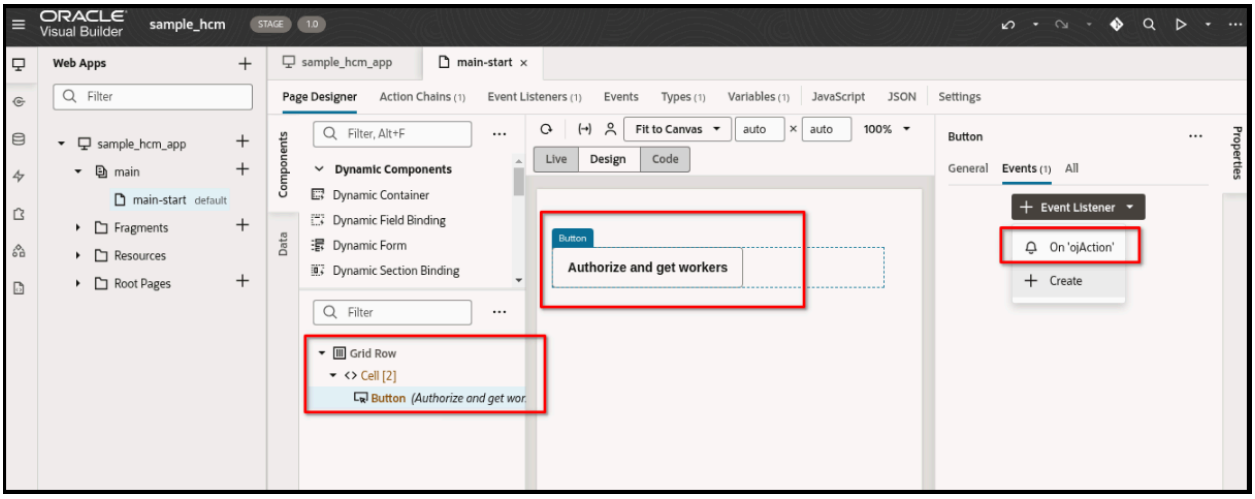
```
 $application.variables.authGetTokenEndpoint = openParams.getTokenEndpoint ||
 securedData.getTokenEndpoint || '';

 $application.variables.ofsGetCodeRedirectUri = localStorage.getItem('authRedirectOrigin') + '/plugin-
auth-redirect/';
 };

 $application.variables.ofsConnector = new OfsConnector({
 onInit: onInit,
 onOpen: onOpen
 });

 $application.variables.ofsConnector.sendMessage({
 method: $application.variables.ofsConnector.TRANSMIT_METHODS.METHOD_READY,
 sendInitData: true,
 sendMessageAsJsObject: true
 });

 //---------------------------------------

 }
 }

 return _init;
});
```

This code helps you to:

a. Include the Plugin Framework connector class into your VBCS application.
b. Store this connector class in an application variable.
c. Initiate the communication with the Field Service Plugin Framework

Create Authorization and Data Retrieval

1. On your VBCS page, place a **Button** with the label **'Authorize and get workers'**.



2. Create a new Action Chain named `_authorize` and associate it with the button's `ojAction` event listener.
3. Change code of the '_authorize' Action Chain, and add the following:

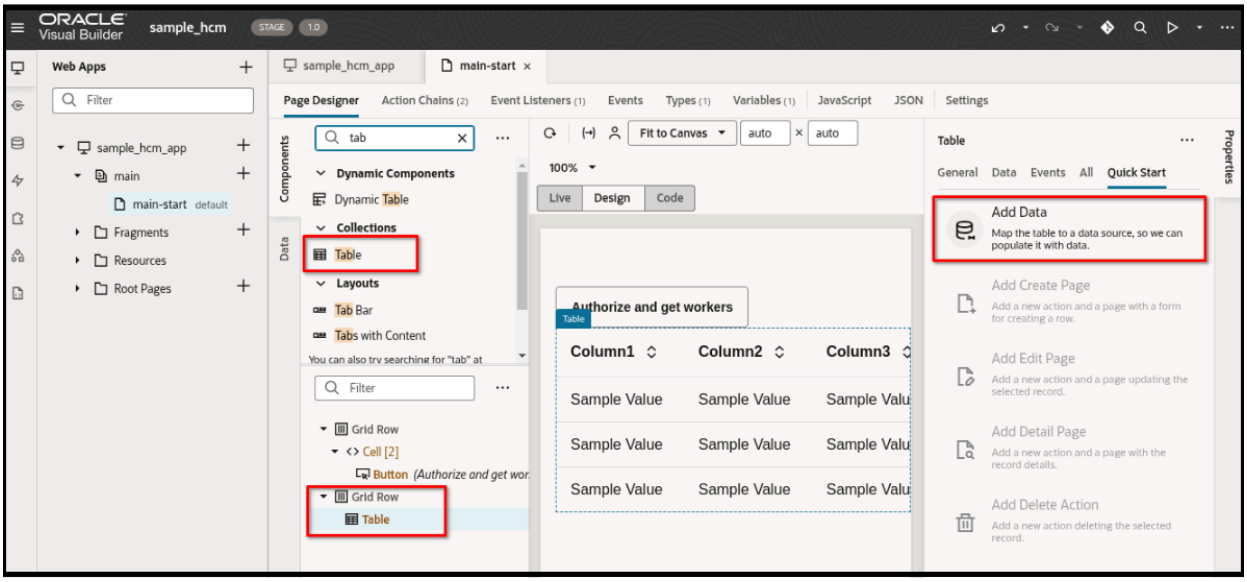
```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
```

```
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class _authorize extends ActionChain {

 /**
 * @param {Object} context
 */
 async run(context) {
 const { $application, $constants, $variables } = context;




 //---------------------------------------

 const callId = $application.variables.ofsConnector.constructor.generateRandomString(16);

 $application.variables.ofsConnector.onCallProcedureResult = async function(procedureResult) {
 if (!procedureResult.callId || procedureResult.callId !== callId) {
 return;
 }

 let token = procedureResult.resultData.token;

 if (token) {
 $application.variables.authHeaders = { 'Authorization': 'Bearer ' + token };
 }
 };

 $application.variables.ofsConnector.sendMessage({
 method: $application.variables.ofsConnector.TRANSMIT_METHODS.METHOD_CALL_PROCEDURE,
 callId: callId,
 procedure: "getAccessToken",
 params: {
 applicationKey: "oauth_user_assertion_application"
 }
 });

 //---------------------------------------




 }
 }

 return _authorize;
});
```

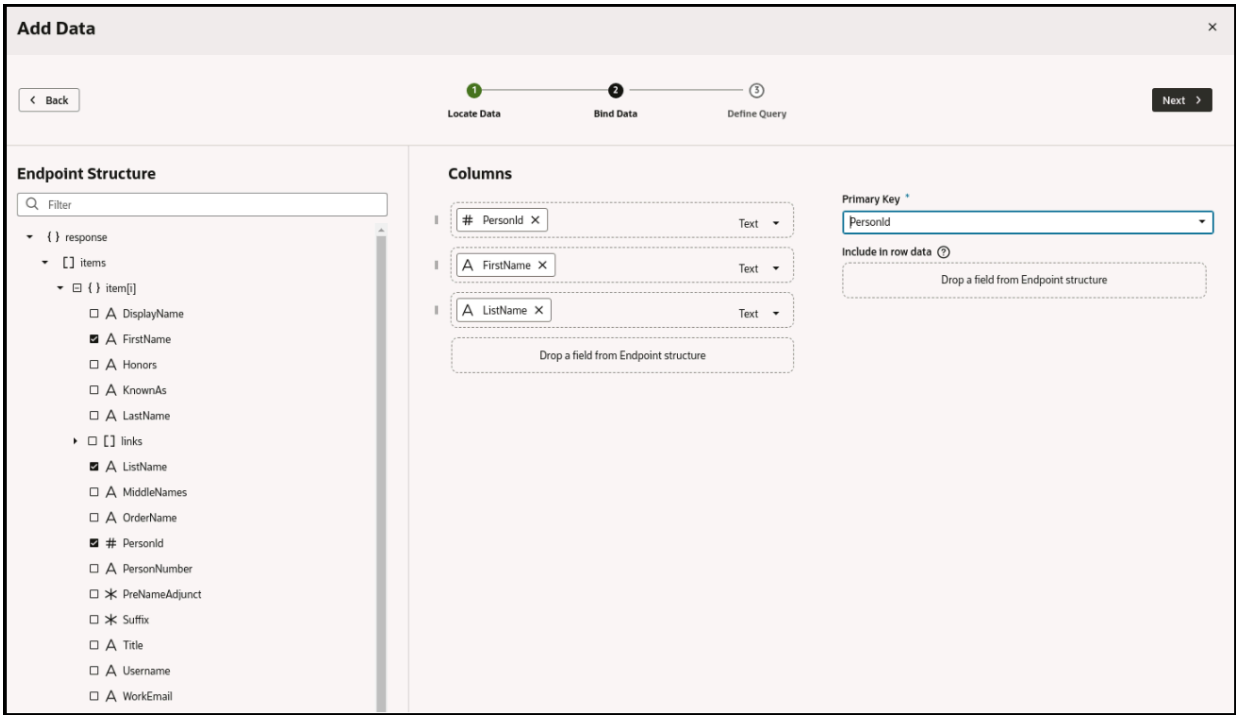
4. This Action chain will be triggered on Authorize button click. It will call the Plugin Framework function to get an authorization code. Next by code it will obtain a token and get the REST API data into the table.

Display Data in a Table

1. Drag and drop a Table component onto your VBCS page layout.

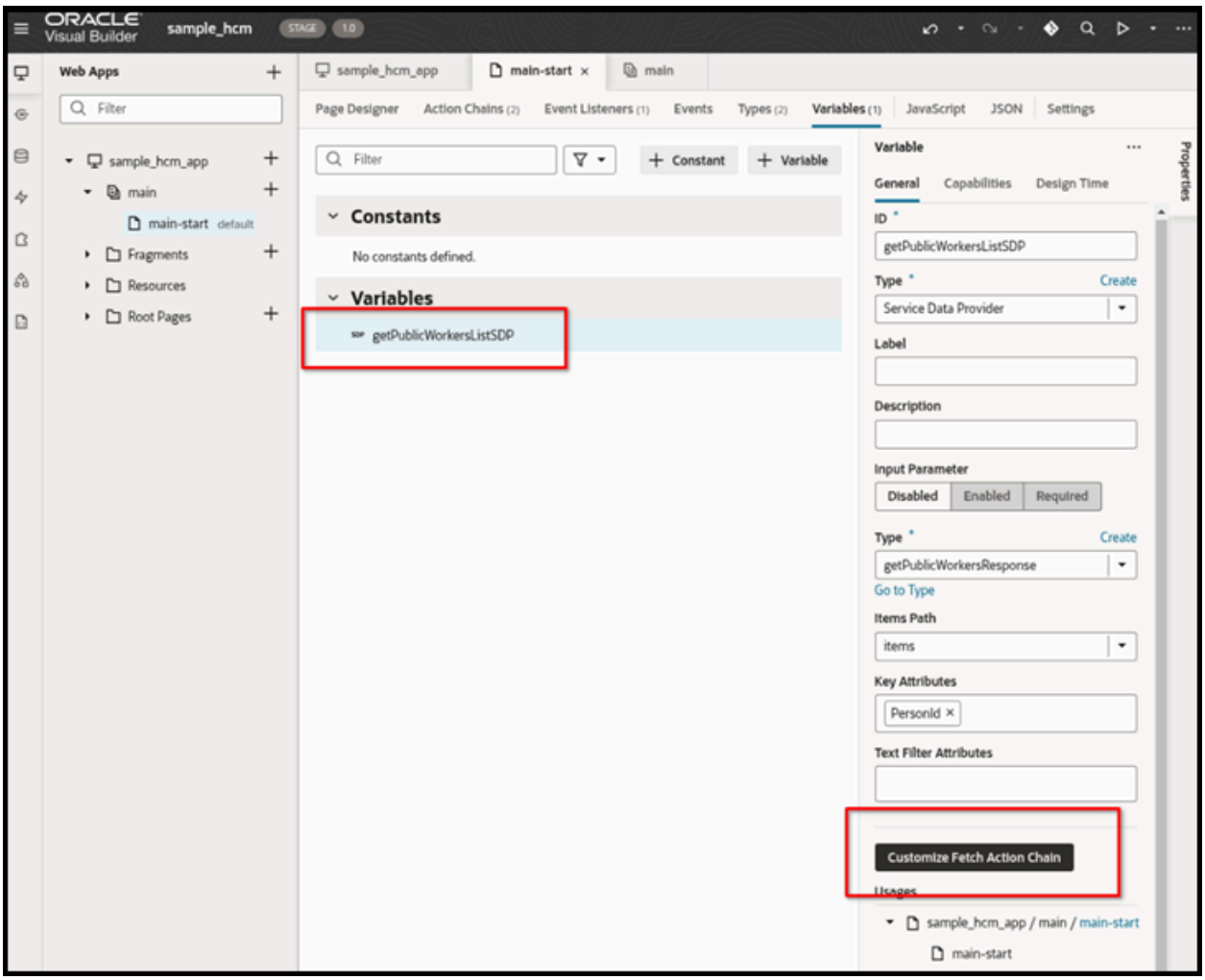


2. As the data source for the table, select the **Public Workers** service connection you created earlier.



3. Select the specific fields from the Public Workers endpoint structure that you want to display in the table . For example, PersonId, FirstName, LastName. Specify PersonId as the key field.

4. Navigate to the automatically created Service Data Provider variable, getPublicWorkersListSDP.



5. Click **Customize Fetch Action Chain**.
6. Within the getPublicWorkersFetch Action Chain, add the following code to the "Headers" section:

   ```
   headers: $application.variables.authHeaders,
   ```

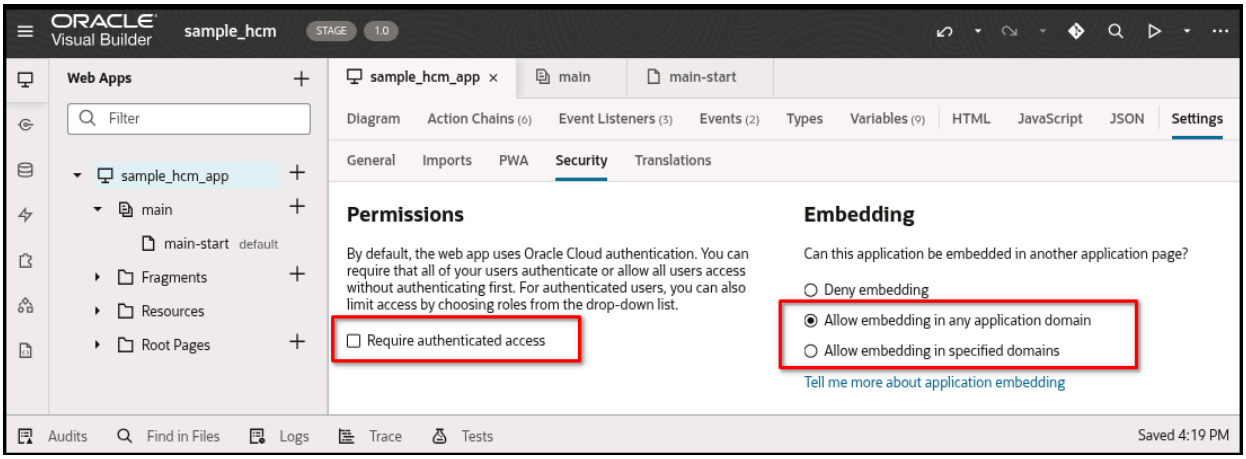
   This will ensure that the authorization header (containing the JWT obtained via the Plugin Framework) is included when fetching data for the table.
7. To list 'getPublicWorkersFetch' Action Chain, use the following code:

```
define([
 'vb/action/actionChain',
 'vb/action/actions',
 'vb/action/actionUtils',
], (
 ActionChain,
 Actions,
 ActionUtils
) => {
 'use strict';

 class getPublicWorkersFetch extends ActionChain {

 /**
```

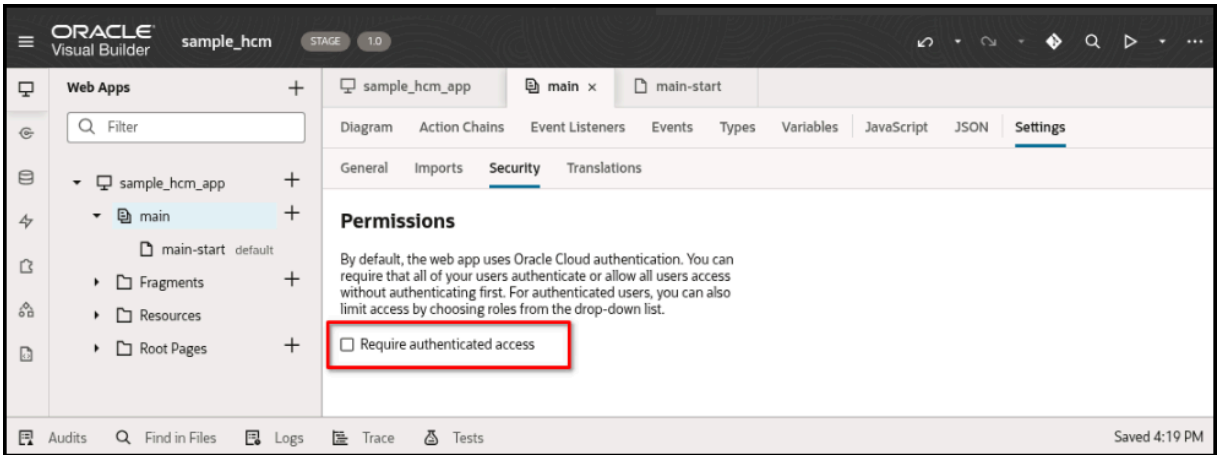
```
 * @param {Object} context
 * @param {Object} params
 * @param {{hookHandler:'vb/RestHookHandler'}} params.configuration
 */
 async run(context, { configuration }) {
 const { $page, $flow, $application, $constants, $variables } = context;
 const callRestEndpoint1 = await Actions.callRest(context, {
 endpoint: 'publicWorkers/getPublicWorkers',
 responseType: 'getPublicWorkersResponse',
 hookHandler: configuration.hookHandler,
 requestType: 'json',
 //-------------------------------------------------------------
 headers: $application.variables.authHeaders,
 //-------------------------------------------------------------
 });

 return callRestEndpoint1;
 }
 }

 return getPublicWorkersFetch;
});
```

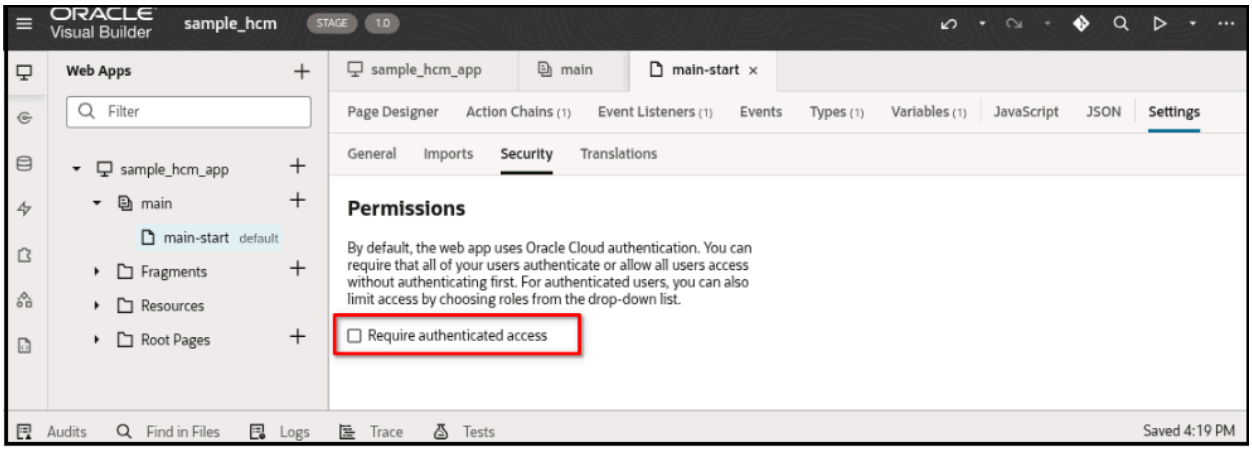
Configure Application Security

1. Navigate to the **Security** tab of your VBCS application.



2. Uncheck the authentication access requirement at all three levels: Application, Flows, and Pages. This is necessary because the plugin will be loaded in a context where the user might not be directly authenticated with the VBCS application itself.





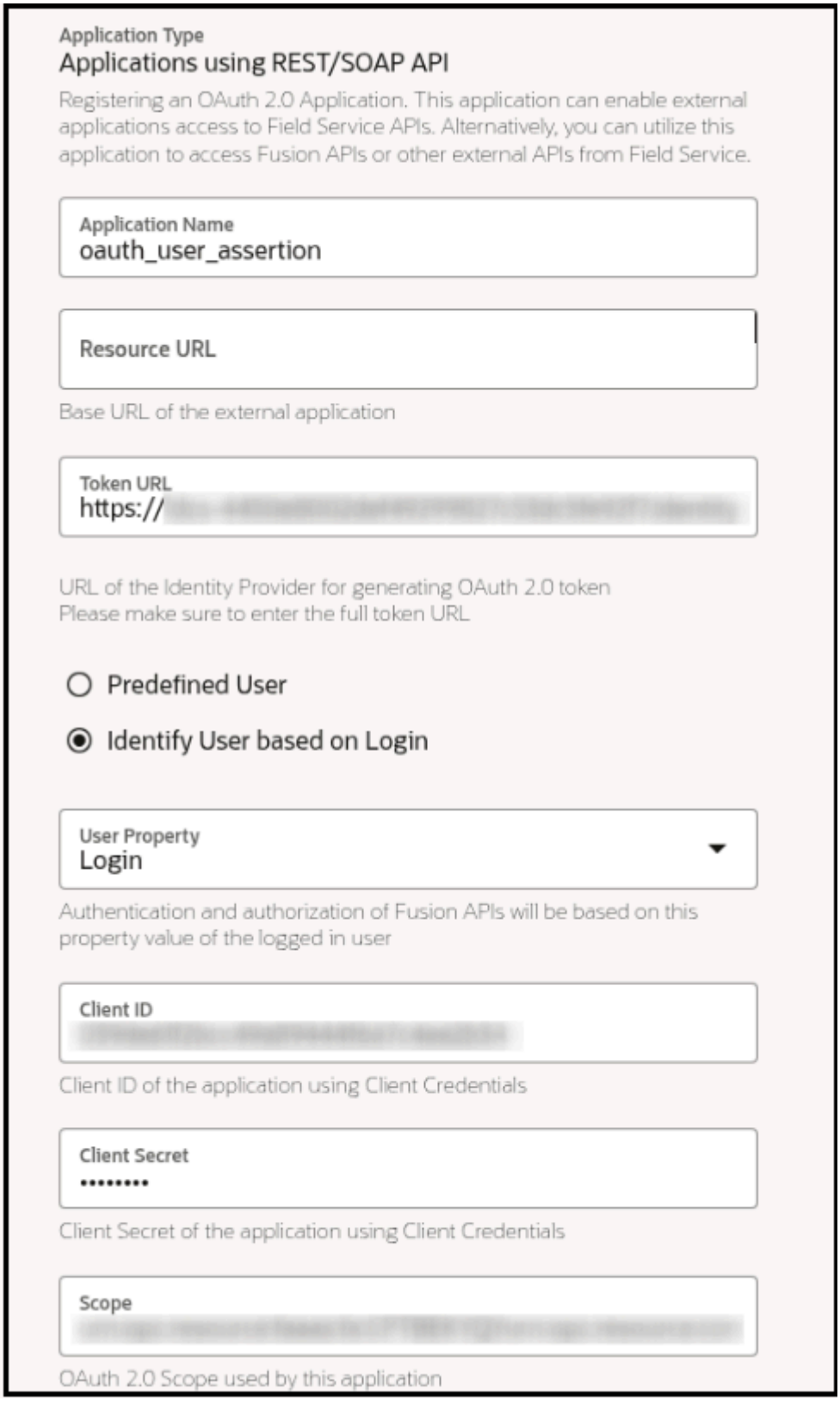
3. Configure the **Embedding** settings to allow embedding from any domain. This is crucial for the VBCS application to function correctly when embedded as an external plugin in Oracle Fusion Field Service.

Remove Temporary Authorization Header

1. Navigate to the **Backend** section and select the custom backend (hcm in this scenario) you created.
2. Remove the temporary Authorization header that you added in the Add an HCM Backend section. The authorization header is used only to help VBCS wizard to obtain data from the REST API and configure columns and primary key. This header is only for design-time assistance and is no longer needed as the authorization will be handled dynamically through the Plugin Framework at runtime.

Configure IDCS and Oracle Fusion Field Service Applications

Ensure that you use the credentials of the IDCS application to configure Oracle Fusion Field Service with Oauth User Assertion for authorization as shown below.

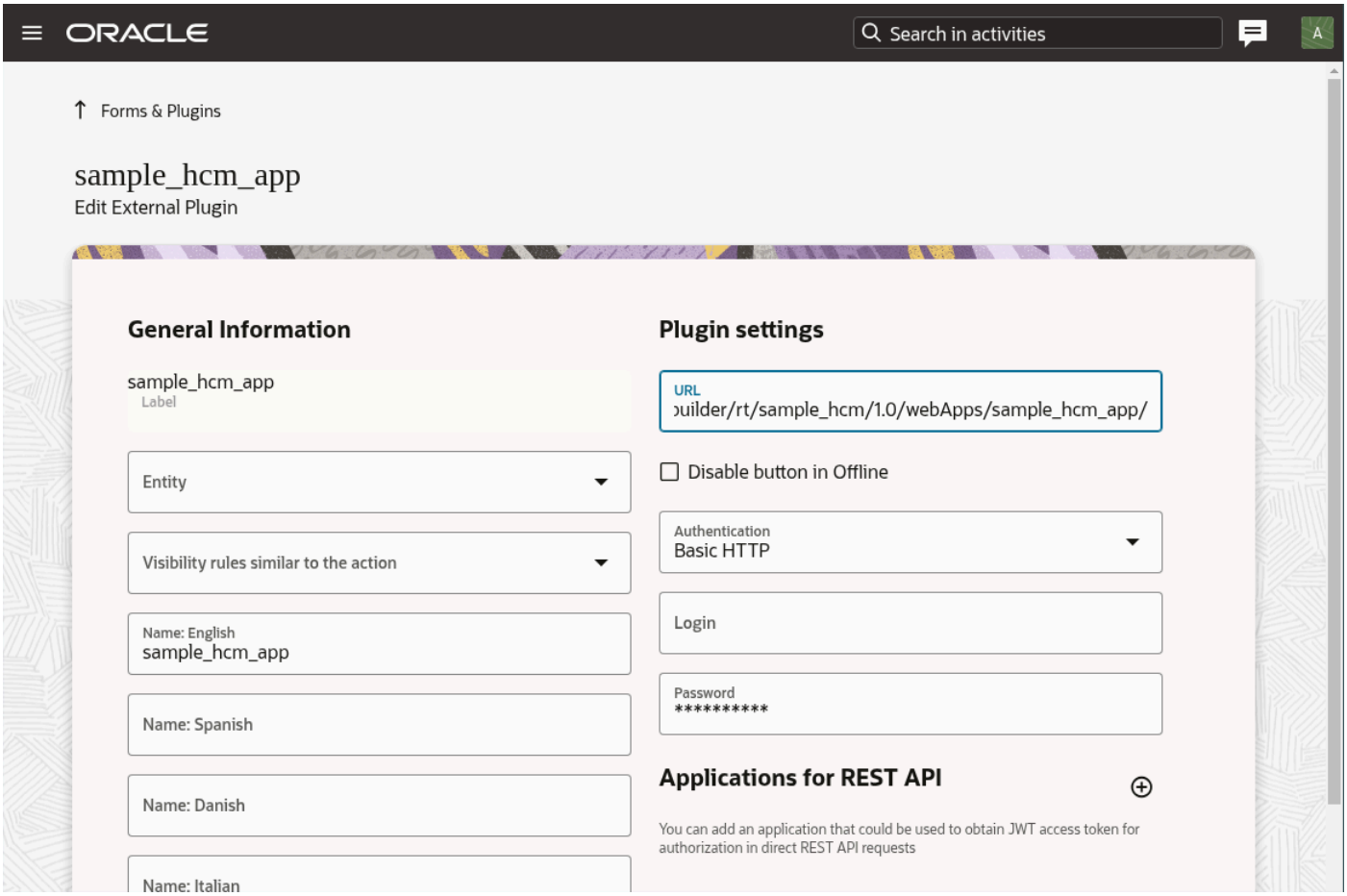

Application Type
Applications using REST/SOAP API
Registering an OAuth 2.0 Application. This application can enable external applications access to Field Service APIs. Alternatively, you can utilize this application to access Fusion APIs or other external APIs from Field Service.
Application Name
oauth_user_assertion
Resource URL
Base URL of the external application
Token URL
https://
URL of the Identity Provider for generating OAuth 2.0 token
Please make sure to enter the full token URL
Predefined User
Identify User based on Login
User Property
Login
Authentication and authorization of Fusion APIs will be based on this property value of the logged in user
Client ID
Client ID of the application using Client Credentials
Client Secret
••••••••
Client Secret of the application using Client Credentials
Scope
OAuth 2.0 Scope used by this application

Configure VBCS Application as an Oracle Fusion Field Service External Plugin

Once your VBCS application is developed and tested, stage it to generate a URL that can be used as an external plugin in Oracle Fusion Field Service.

To configure the VBCS application as Oracle Fusion Field Service external plugin:

1. Navigate to **Configuration → Forms & Plugins → Add Plugin → External Plugin**. The Add External Plugin page appears.
2. In the **Plugins settings** section, specify the stage URL of VBCS as plugin URL.



3. In the **Applications for REST API** section, click the + sign to add an application with the **oauth_user_assertion_application** key. Select the Oracle Fusion Field Service application you configured in the Configure IDCS and Oracle Fusion Field Service Applications section.
4. Click **Update**.

Troubleshooting

This section describes the troubleshooting tips that can help resolve common issues encountered when integrating your VBCS application with Oracle Fusion Field Service.

1. **HTTP 403 Error (Forbidden)**: If you encounter a "403 Forbidden" error, it might indicate that your user account in Fusion does not have the necessary permissions. Ensure that the ORA_HRC_HUMAN_CAPITAL_MANAGEMENT_INTEGRATION_SPECIALIST_JOB role is assigned to your Fusion user. You can check and assign this role in Fusion through **Tools -> Security Console -> Users**.
2. **CORS Error**: If you see a CORS (Cross-Origin Resource Sharing) error, the browser is preventing the VBCS application from making requests to the REST API due to security restrictions. Contact your Oracle Fusion Field Service administrator to verify the **ORA_CORS_ORIGINS** profile option code in Fusion: My Enterprise -> Setup and Maintenance → Tasks (visible in the Right Side of the screen) -> Search -> **Manage Administrator Profile Values** task -> Add full URL (**including "https"** prefix and **without trailing slash)** to Profile Value field. The value is a domain from which the plugin is loaded. For more information, see *https://docs.oracle.com/en/cloud/saas/fusion-service/facoe/c_chat_configure_for_cors.html*

   The domains that should be listed in the CORS policy field:

   a. for VBCS plugins - domain of VBCS application
   b. for standard plugins - **plugins-1-**{FS domain}, use environment name in the domain, for example, ofsc-******.test instead of alias for hosted plugins - **plugins-0-**{FS domain}, use environment name in the domain, for example, ofsc-******.test instead of alias
   c. for requests from browser's console - name of the domain where request is called

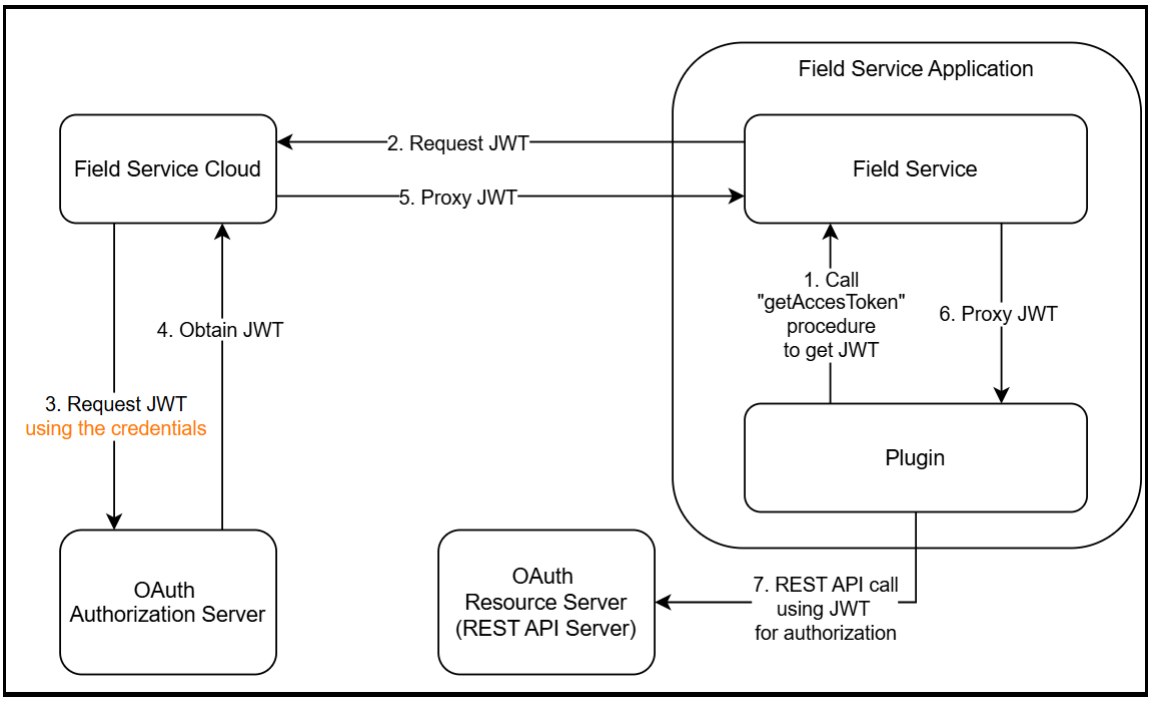
Obtain and Use a Refresh Token

Refresh tokens are not provided in the current version of the Plugin API. If your access token expires, you should request a new one instead.

OAuth Client Credentials Flow (getAccessToken procedure)

OAuth Client Credentials Flow is widely supported by numerous Identity Providers and is used when REST API calls are made on behalf of an application. In this setup, multiple users share the same application for REST API calls. However, this may be unsuitable in cases where user-specific API calls are required. The OAuth User Assertion or OAuth Authorization Code Grant Flows are recommended for such scenarios.

This diagram illustrates the OAuth Client Credentials Grant Flow, showcasing how applications can make REST API calls on behalf of multiple users while highlighting its suitability and limitations in different scenarios.



Advantages:

- Broadly used and supported by most OAuth Identity Providers.
- This is ideal when the plugin's access to the REST API is not tied to user-specific privileges. All users accessing the plugin share the same level of access, which is managed by the configuration on the Identity Provider side.

Process Workflow Summary:

1. Add the **OAuth Client Credentials** application to the **Configuration → Applications** page.
2. Configure an Identity Provider application that supports the OAuth Client Credentials Flow.
3. Enter the credentials (Client ID, Client Secret, Scope, Identity Provider endpoint) on the Oracle Fusion Field Service side.
4. Add the application to the plugin using the **Edit Plugin** page.
5. Call the "getAccessToken" procedure from the plugin with this application in the procedure parameters.
6. Obtain a JWT access token in the procedure response.
7. Use the JWT access token for REST API request authorization.

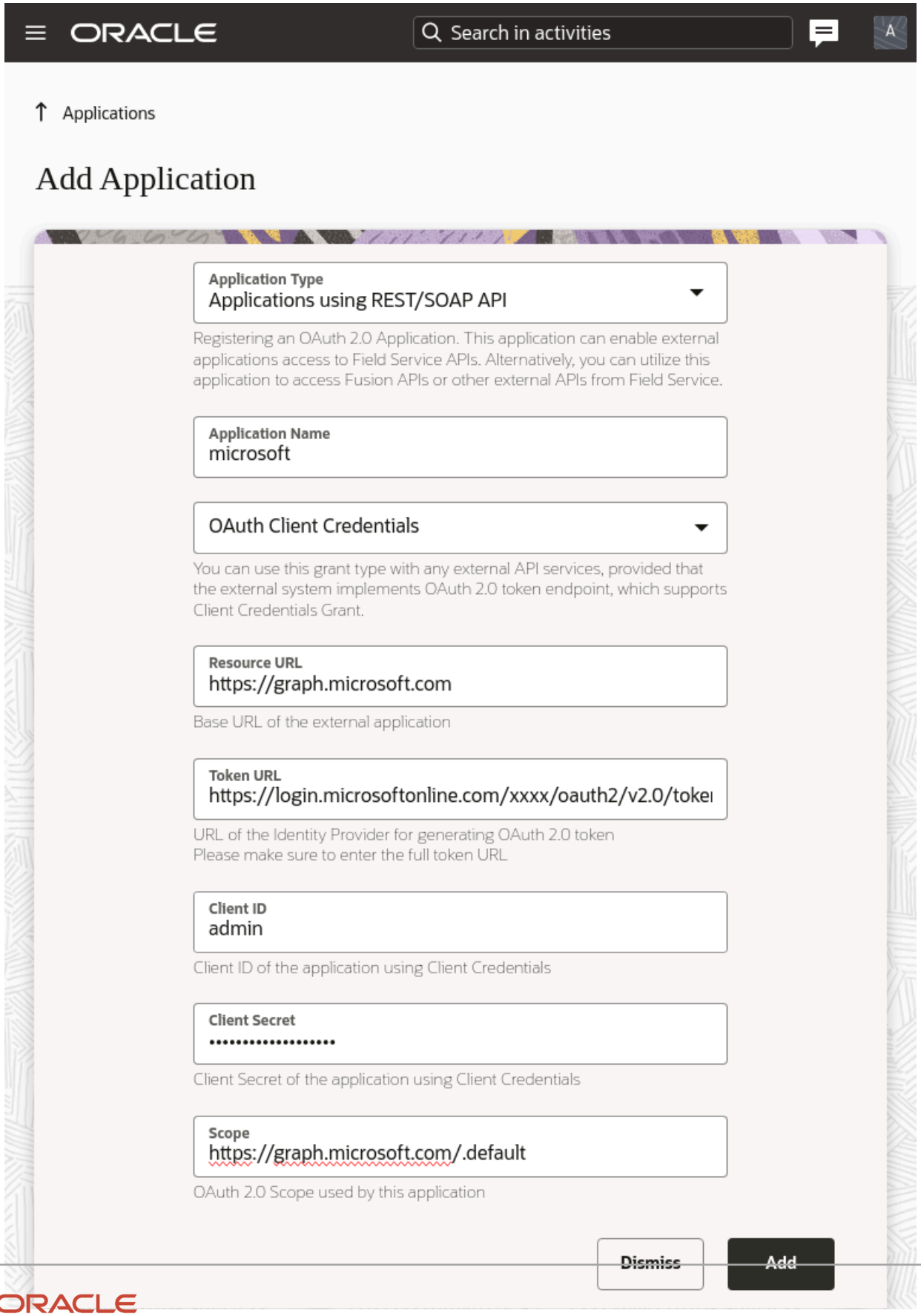
Oracle Fusion Field Service Application Configuration for OAuth Client Credentials Flow

This topic outlines the steps to create and configure a Oracle Fusion Field Service application using the OAuth Client Credentials flow to obtain an access token. This flow is suitable for scenarios where a user context is not required, but keep in mind that all users of the same plugin will share the same application and thus use the same access token.

To create an Oracle Fusion Field Service application:

1. Navigate to **Configuration → Applications**.

2. Click **Add Application**. The Add Application page appears.


ORACLE
Search in activities
A
Applications
Add Application
Application Type
Applications using REST/SOAP API
Registering an OAuth 2.0 Application. This application can enable external applications access to Field Service APIs. Alternatively, you can utilize this application to access Fusion APIs or other external APIs from Field Service.
Application Name
microsoft
OAuth Client Credentials
You can use this grant type with any external API services, provided that the external system implements OAuth 2.0 token endpoint, which supports Client Credentials Grant.
Resource URL
https://graph.microsoft.com
Base URL of the external application
Token URL
https://login.microsoftonline.com/xxxx/oauth2/v2.0/toke
URL of the Identity Provider for generating OAuth 2.0 token
Please make sure to enter the full token URL
Client ID
admin
Client ID of the application using Client Credentials
Client Secret
Client Secret of the application using Client Credentials
Scope
https://graph.microsoft.com/.default
OAuth 2.0 Scope used by this application
Dismiss
Add

3. Select **Applications using REST/SOAP API** as the application type. You will be prompted to enter a name for this application; provide a descriptive name.
4. From the available application security types, select **OAuth Client Credentials**. This indicates that this application will use its own credentials (Client ID and Client Secret) to obtain an access token from the Identity Provider without involving a specific user's context.
5. In the **Resource URL** (Optional) field, specify the URL that represents the OAuth 2.0 Resource Server (the server that provides REST data). It will be included in the "applications" section during the "init" method to avoid hardcoding the URL in the plugin.
6. In the **Token URL** field, enter the URL of the OAuth 2.0 Authorization Server (Identity Provider) that issues access tokens. For example, `https://{idcsUrl}/oauth2/v1/token` .

   > **Note:** In some cases, the OAuth 2.0 Resource Server and Authorization Server might share the same domain.

7. Enter **Client ID** Obtained from the Identity Provider configuration while creating the application in the Authorization Server.
8. Enter **Client Secret** Also retrieved from the Identity Provider configuration.
9. Specify **Scope** . This is a string (usually space-separated) that defines the permissions the Field Service application will request when obtaining an access token. The value is provided by the Identity Provider's configuration. For example: `urn:opc:resource:fusion:xxxxxxxx:field-service`
10. Click **Add**.

    Examples of credentials:
    - Resource Server URL: https://graph.microsoft.com/v1.0/users
    - Auth Server URL:https://login.microsoftonline.com/{tenant}/oauth2/v2.0/token
    - Client ID: "{clientId}"
    - Client Secret: "{clientSecret}"
    - Scope: https://graph.microsoft.com/.default

IDCS Configuration for OAuth Client Credentials Flow

This topic describes how to configure an integrated application within Oracle Identity Cloud Service (IDCS) to enable the OAuth Client Credentials flow and issue access tokens for use by Oracle Fusion Field Service.

To create an integrated application:

1. Navigate to the identity domain configuration in IDCS.
2. Select the **Integrated Applications** section from the left pane.
3. Click **Add Application**, which is available at the top of the page.
4. In the Add application dialog box, select **Confidential Application** as the application type, and then click **Launch Workflow** The **Add Confidential Application** dialog box appears.
5. Enter a descriptive name for your new integrated application.
6. Click **Submit**.
7. Next, click **Edit OAuth configuration** to proceed to the **Edit OAuth configuration** step.
8. In the **Client Configuration** section, select **Configure this application as a client now**.
9. In the **Authorization** section, select the **Client credentials** grant type. Ensure all other grant types are unchecked for this specific scenario, although real-world integrations might use multiple grant types for a single application.

   > **Note:** If your plugin intends to use refresh tokens functionality (although the Plugin API currently doesn't directly return them), you can optionally check the **Refresh token** grant type here.

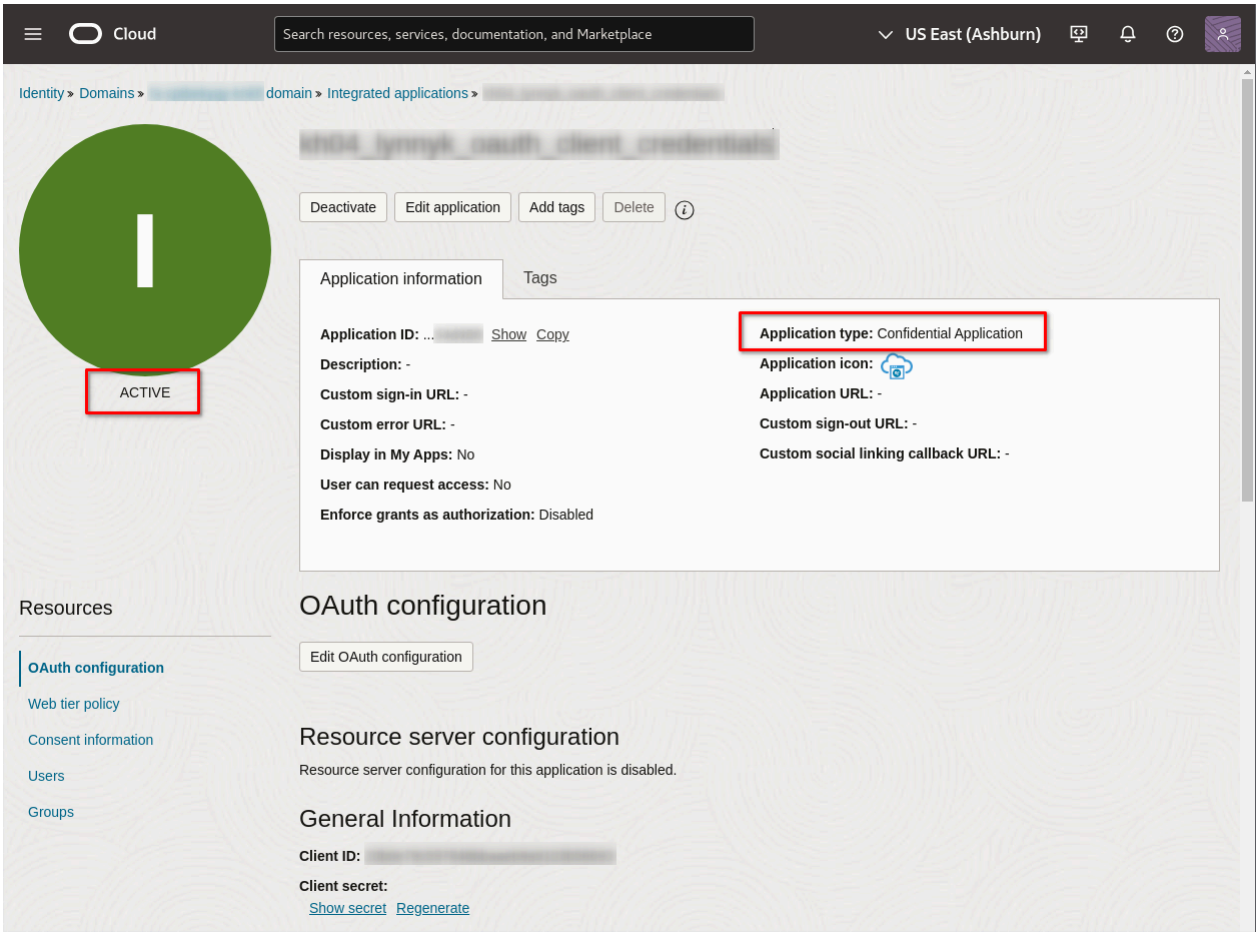
10. In the **Token Issuance Policy** section, select **Add resources** if you want your application to access the APIs of other applications.
11. In the **Resources** section, click **Add Scope**. A list of applications appears in the Add scope dialog box.
12. Select the Fusion Applications Cloud Service and then click **Add**.

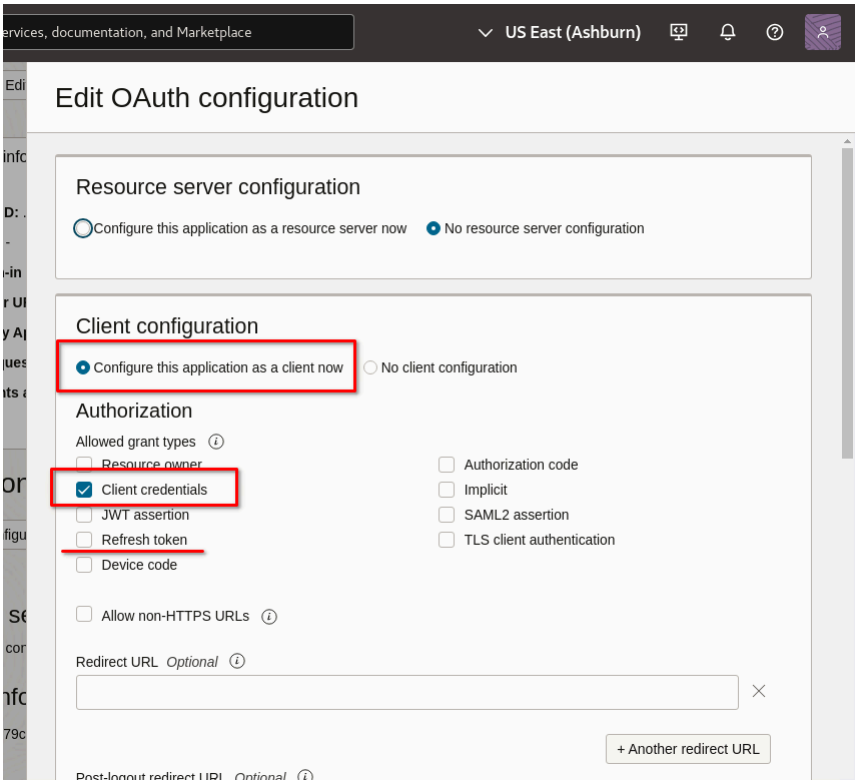
    The selected application is added to the Resource scope.

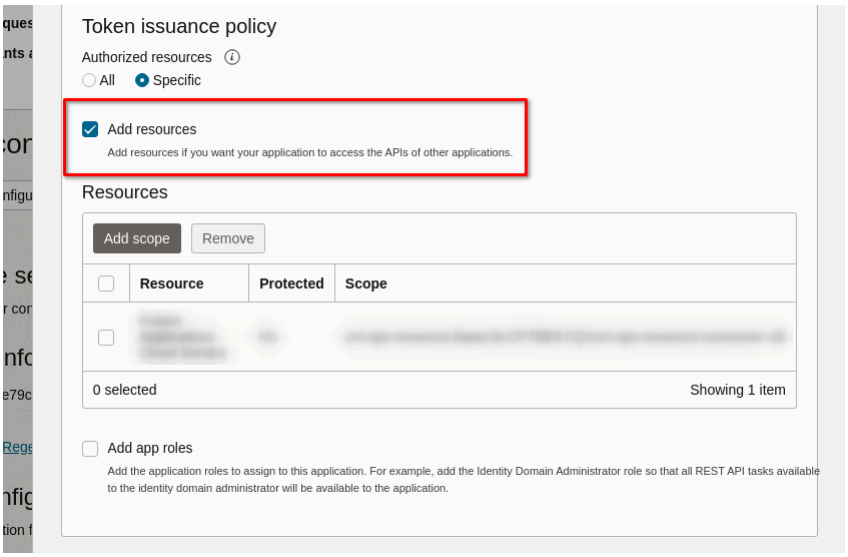
    > **Note:** If Fusion Applications Cloud Service is not listed as a resource, it indicates that Fusion Service is not linked to the IDCS domain. In this case, the integrated application must be created in the domain linked to Fusion Service.

13. Click **Next**.
14. Skip **Web tier policy**.
15. Click **Finish**. The newly created application is now listed on the Integrated Applications page and its status is **Inactive**.

16. Click the **Activate** icon to activate the application. Once activated, your application appears as configured and ready for integration.





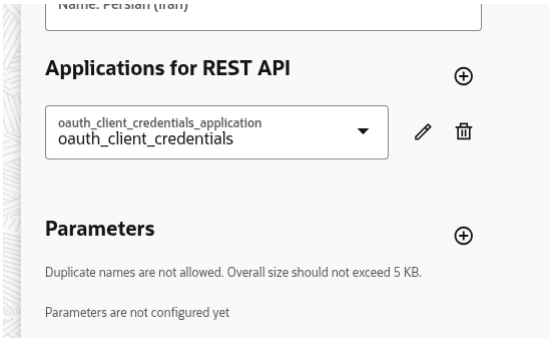


Plugin Configuration to Connect Oracle Fusion Field Service Application

This topic explains how to link the Oracle Fusion Field Service application configured for the OAuth Client Credentials flow to a plugin within Oracle Fusion Field Service. The steps are identical to those for the User Assertion flow.

To connect the Oracle Fusion Field Service Application to a plugin:

1. Navigate to **Configuration > Displays > Forms and Plugins**. Or, click the **Configurations >Users Type> Screens** and then select the required plugin.
2. Select the plugin from the available list that you would like to connect.
3. Click **Edit** from the **Action** menu. The **Edit Plugin** page appears.
4. In the **Applications for Rest API** section, click **Add**.
5. Select the Oracle Fusion Field Service application you created in *Oracle Fusion Field Service Application Configuration for OAuth Client Credentials Flow* from the drop-down list of available applications. This establishes the connection between the plugin and the configured application.



6. Click **Update**.

Sample Plugin to Obtain Access Token (Client Credentials)

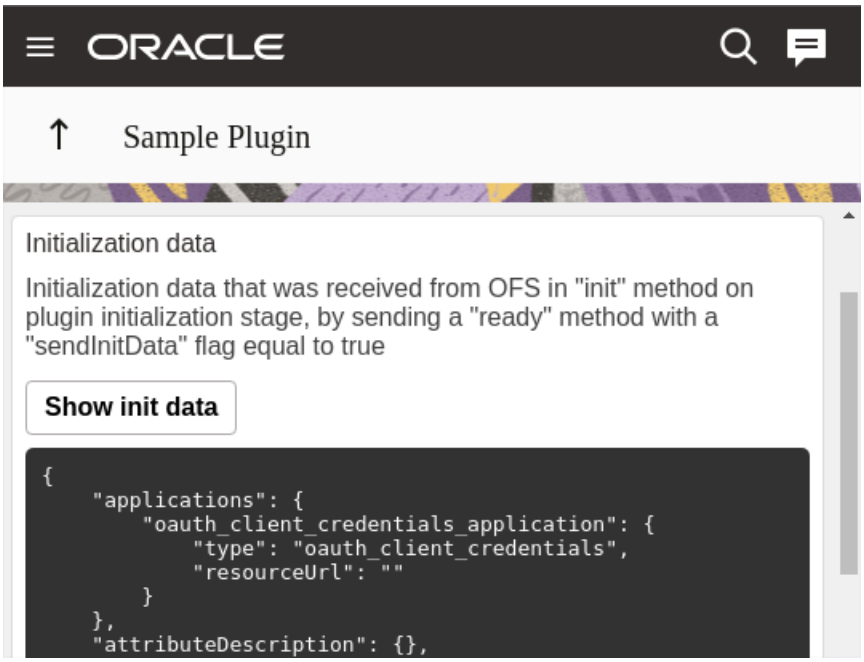
This topic outlines how a plugin, such as the Sample Plugin, can use the configured Oracle Fusion Field Service application (for Client Credentials flow) to obtain an access token using the "getAccessToken" procedure. The process is the same as for the User Assertion flow.

Initial Application List

Upon initialization, the plugin receives a message containing a list of available applications. This message includes the **resourceUrl** field for each application.

The **resourceUrl** is obtained from the plugin's configuration and is used by the Sample Plugin in its subsequent API requests.

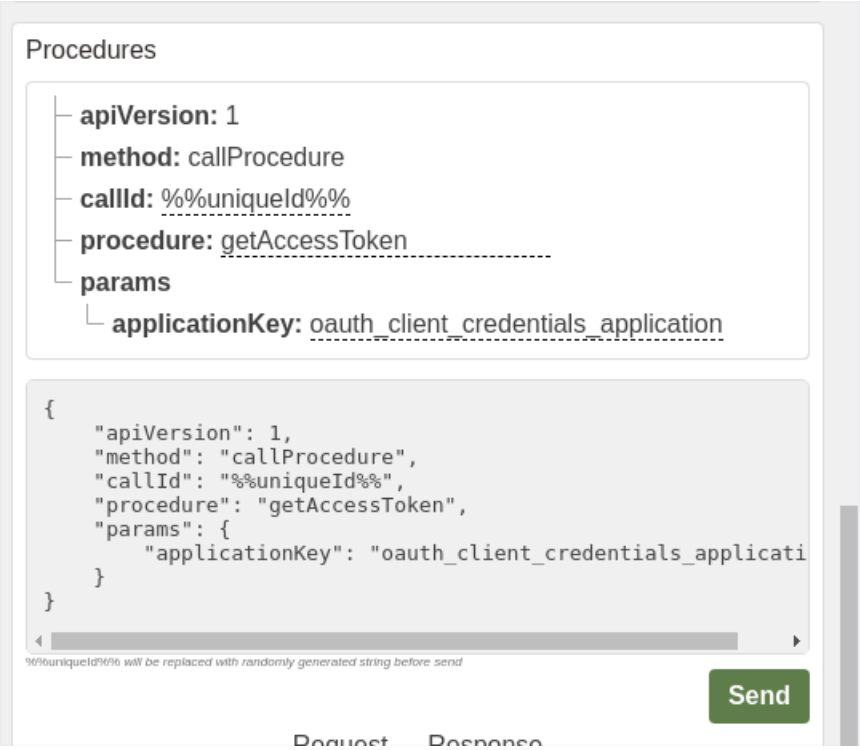
The following screenshot illustrates the Initialization data, showcasing how the plugin obtains and employs the resourceUrl field from its configuration during the initialization process.



Obtain an Access Token

To acquire an access token using the Sample Plugin, you need to utilize the **Procedures** functionality as shown below in the screenshot. You can

- Send a `callProcedure` method call to the plugin.
- Within the `callProcedure` call, specify `getAccessToken` as the desired procedure.



Retrieve the Token

Once the callProcedure with getAccessToken is processed, the resulting access token will be available in the `callProcedureResult` response method as shown below:

Validate the Token by Fetching Data

With the access token obtained, you can fetch data from a REST API to verify its validity.

```
fetch("https://rest-api-instance.com/rest/api", {
 headers: {
 Authorization: "Bearer <accessToken>"
 }
})
.then(resp => resp.json())
.then(json => console.log(json));
```

VBCS Application as Oracle Fusion Field Service External Plugin

The application for Client credentials flow is the same as for User Assertion Flow.

Obtain and Use a Refresh Token

Refresh tokens are not provided in the current version of the Plugin API. If your access token expires, you should request a new one instead.

getAccessToken Procedure Errors

Error Type	Error Code	Example	Possible Reason and Solution
TYPE_PROCEDURE_PARAM	CODE_PROCEDURE_PARAM_VALUE_INVALID		Application Identifier <applicationKey> is not a string, null or number,boolean, object, array, empty string is invalid. Check the applicationKey parameter.
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_GET_ACCESS_TOKEN_WRONG_APPLICATION_KEY		Procedure is called with Application Identifier <applicationKey> that is absent in Plugin's configuration. Do not call the procedure with an application differ from the applications that were received in the init method.
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_GET_ACCESS_TOKEN_APPLICATION_NOT_CONFIGURED		Application Identifier <applicationKey> is not chosen in Plugin's configuration, or it's type is not supported by getAccessToken procedure. Choose an Application on Plugin Configuration Screen.

Error Type	Error Code	Example	Possible Reason and Solution
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_GET_ACCESS_TOKEN_OFFLINE_NOT_SUPPORTED		The getAccessToken procedure is called while device is offline. Run procedure when the device is online.
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_GET_ACCESS_TOKEN_PROCEDURE_TIMEOUT		Field Service couldn't get access token from server in **15** seconds. Usually such long timeout caused problems with internet connection or problems of Authorization server. Normally it should respond in 4 sec.

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	<pre>"data": {
		"status": "no_access",
		"detail": "",
		"token": ""
		}</pre>
			Exceptional case. Contact administrator.

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	<pre>"data": {
		"status": "app_misconfigured",
		"detail": "",
		"token": ""
		}</pre>
			1. Application Identifier <applicationKey> is not chosen in Plugin's configuration.
			2. The OFS Application is not active.
			3. Chosen Applicaion is not found.
			4. Choose an Application on Plugin Configuration Screen.
			5. Activate an OFS Application.
			6. Exceptional case

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	<pre>"data": {
		"status": "unexpected_response",
		"detail": "",
		"token": ""
		}</pre>
			Field Service got an unexpected response from Identity Provider.
			
			Token URL has wrong suffix. For IDCS the right one is https://{idcsUrl}**/oauth2/v1/token**

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	<pre>"data": {
		"status": "cannot_infer_user_id",
		"detail": "",
		"token": ""
		}</pre>
			The field of user should be filled.

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	<pre>"data": {
		"status": "token_service_error",
		"detail": "invalid_request",</pre>
			Check the scope field is not empty, in case of Fusion it is mandatory.**Exceptional case. Contact administrator.**

Error Type	Error Code	Example	Possible Reason and Solution
		"token": "" }	

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": {
		"status": "token_service_error",
		"detail": "invalid_client",
		"token": ""
		}
			1. Activate an IDCS Application
			2. Check that Client ID / Client Secret is correct

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": {
		"status": "token_service_error",
		"detail": "invalid_grant",
		"token": ""
		}
			Check the username is correct
			
			Check that certificate is valid
			
			The client type (IDCS application configuration) should be "Trusted" not "Confidential" (to generate self-signed user assertions).

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": {
		"status": "token_service_error",
		"detail": "unauthorized_client",
		"token": ""
		}
			Check that OAuth client has authorization to use the requested grant.

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": {
		"status": "token_service_error",
		"detail": "unsupported_grant_type",
		"token": ""
		}
			Exceptional case. Contact administrator.

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": {
		"status": "token_service_error",
		"detail": "invalid_scope",
		"token": ""
		}
			Check the scope value

TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": { "status": "internal_error", "detail": "...", "token": ""	**Exceptional case. Contact administrator.**

Error Type	Error Code	Example	Possible Reason and Solution
		}	
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": { "status": "connection_error", "detail": "", "token": "" }	Field Service didn't get a valid response from Identity Provider. The reasons could be differ from timeout to wrong content. Check that Auth Server URL is correct.
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	CODE_PROCEDURE_FAILED	"data": { "status": "unexpected_response", "detail": "", "token": "" }	
TYPE_PROCEDURE_ERROR	CODE_PROCEDURE_UNKNOWN		Check the procedure name
TYPE_INTERNAL	CODE_UNKNOWNCODE_JSON_INVALIDCODE_METHOD_NOT_SUPPORTED		Check JSON syntax and method that is sent

Related Standards

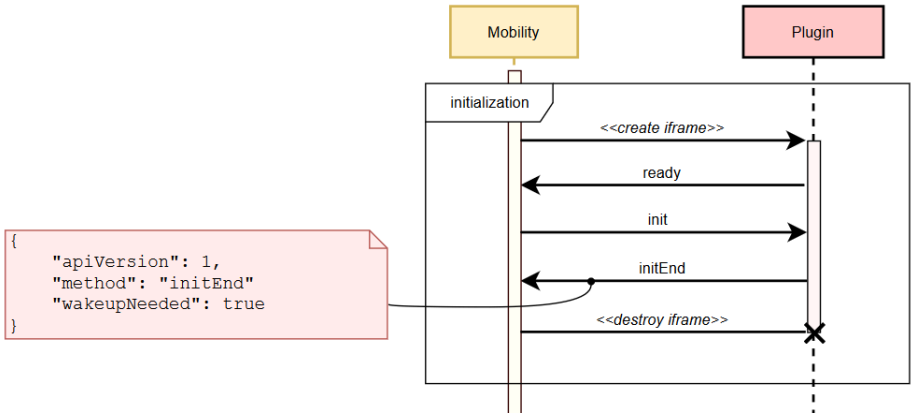
OAuth 2.0 Client Credentials Grant	*https://datatracker.ietf.org/doc/html/rfc6749#section-4.4*
JSON Web Token (JWT)	*https://datatracker.ietf.org/doc/html/rfc7519*
OAuth 2.0 User Assertion Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants	*https://datatracker.ietf.org/doc/html/rfc7521*
OAuth 2.0 Authorization Code Grant	*https://datatracker.ietf.org/doc/html/rfc6749#section-4.1*

Plugin Lifecycle

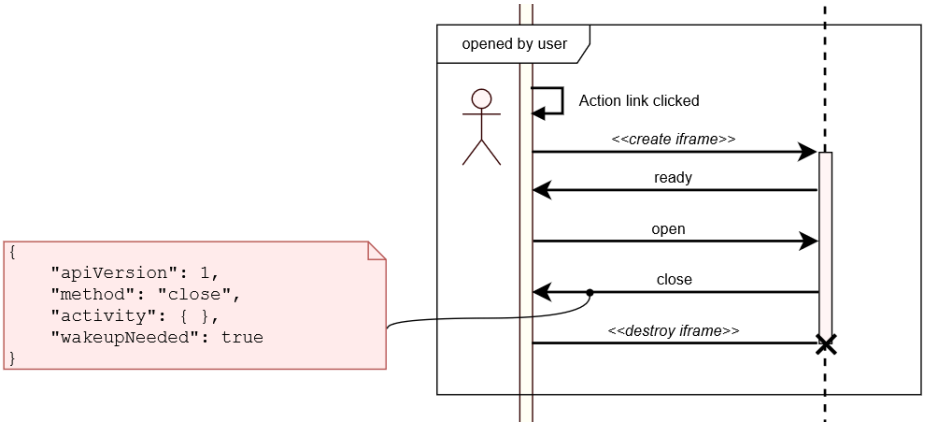
This topic provides the lifecycle diagram of a plugin.

The plugin lifecycle diagram is divided into four parts:

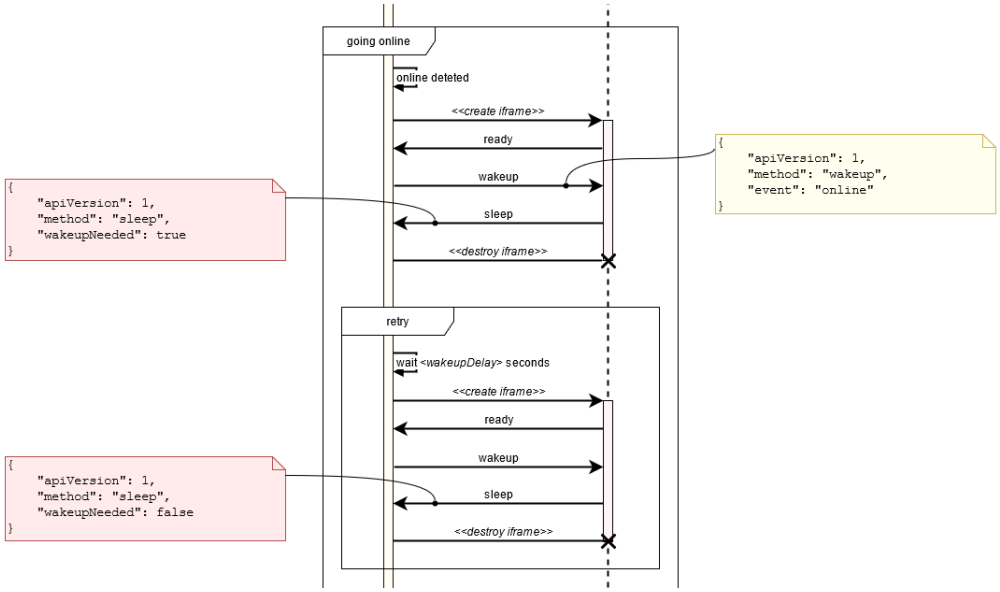
- Plugin is initialized: This diagram shows the initialization of the plugin:



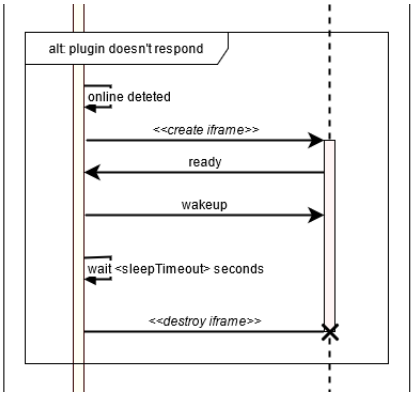
- Plugin is opened by a user: This diagram shows the flow when a user opens the plugin:



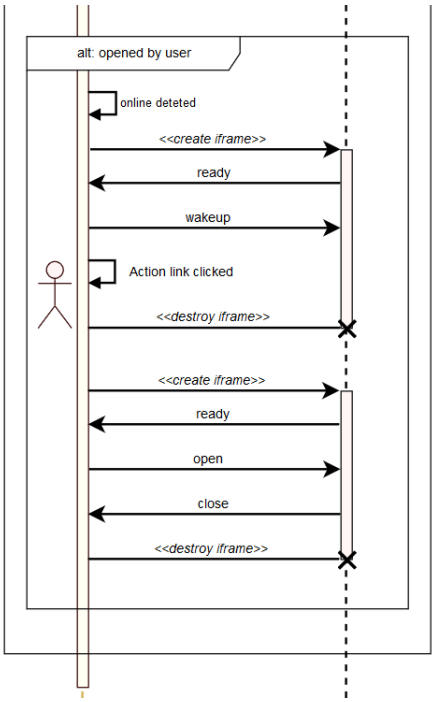
- Plugin switches online with retries: This diagram shows the flow of background synchronization performed by the plugin, when Oracle Fusion Field Service detects that the plugin is online:



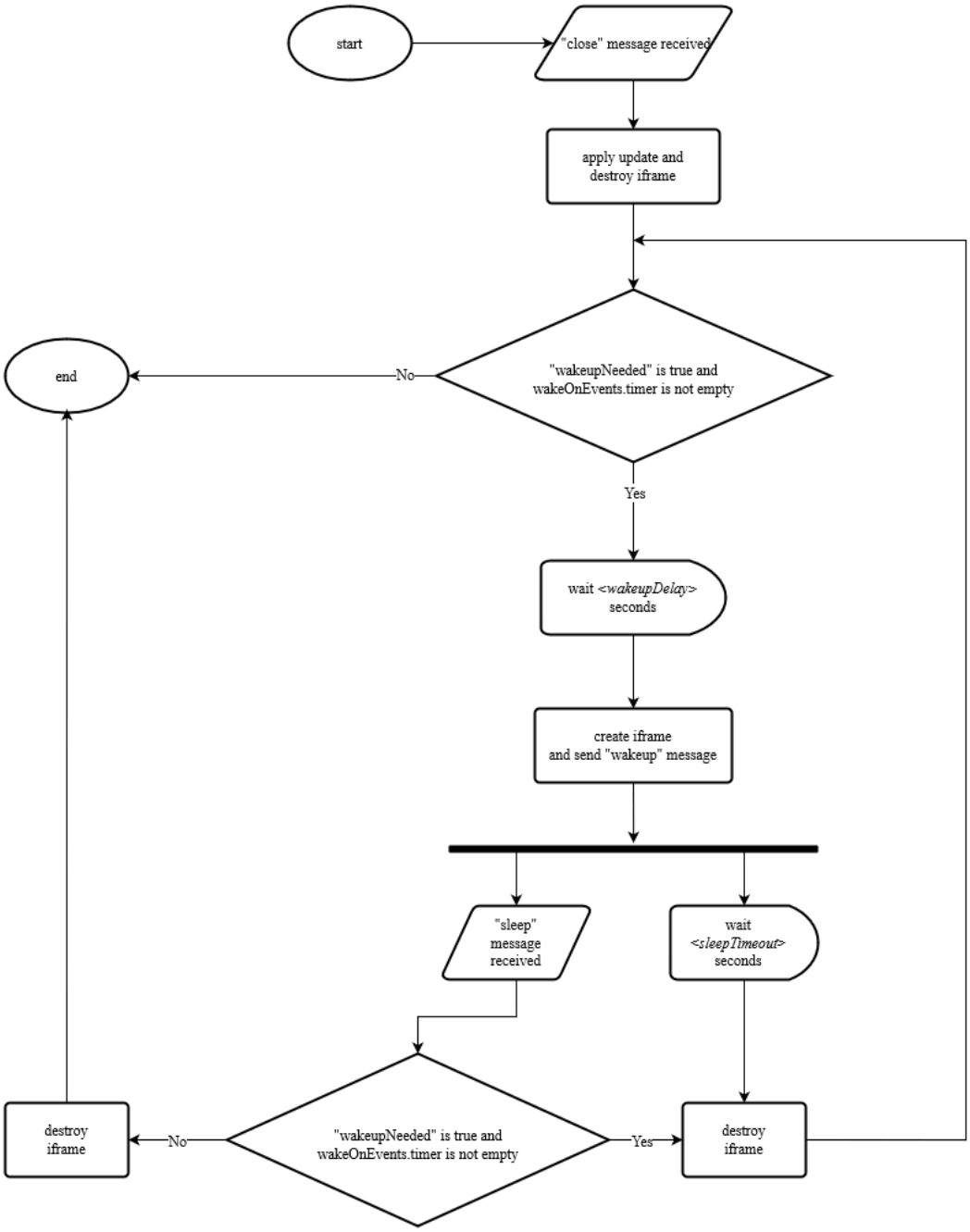
- Plugin doesn't respond: This diagram shows the flow when the plugin doesn't send the *sleep* message within two minutes:



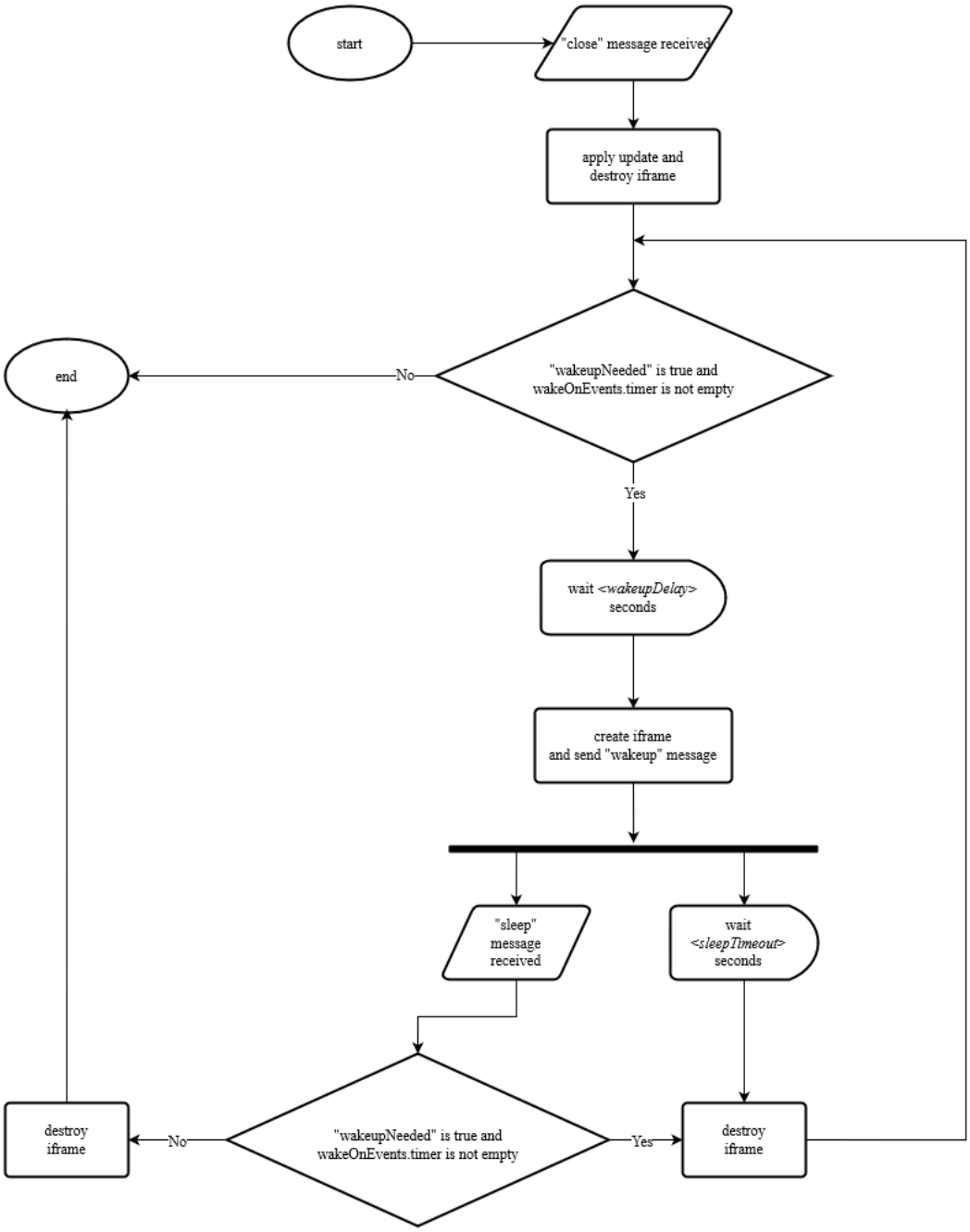
- Plugin is opened by the user: This diagram shows the flow when the plugin is opened by the user when the background synchronization is still in progress:



This flowchart shows the background synchronization of the plugin for the *online* event:



This flowchart shows the background synchronization of the plugin for the *timer* event:



Plugin Rules and Guidelines

You create a plugin on the **Forms & Plugins** page and add it as a button on the required page. You can also add buttons on the activity hint. You can configure the plugin during creation or editing of an Action link.

Here are some rules and guidelines that you must follow, while creating a plugin:

- **Plugin URL** must point to the main page of the plugin's source files. The plugin needs the HTTPS protocol. Main page of the plugin must be accessible via configured **Plugin URL**. Its content is a valid HTML/XHTML page, that should load JavaScript code sources, static resources (images, .css stylesheets) or contain them itself.
- For a plugin to be treated as a valid plugin, the main page must run the JavaScript code that interacts with Oracle Fusion Field Service through the **Plugin API**.
- To make a plugin accessible offline, it must use a Service Worker that loads its resources for offline use. However, you must be aware of their availability in different versions of browsers and operating systems.
- A plugin can save data for offline locally on the user's device using cookies, the localStorage property, or the IndexedDB object.
- The plugin’s URL is loaded into iframe, and the URL points outside the Oracle Fusion Field Service domain. Therefore the plugin’s application cache, cookies, the localStorage property, and the IndexedDB object are separated from that of Oracle Fusion Field Service. These elements can't interfere with one another, according to the *Same origin* policy, described on the https://www.w3.org website. Most properties of the parent window (window.parent property) are also unavailable for the plugin's JavaScript. So, the only way to interact with Oracle Fusion Field Service is through the **Plugin API**
- The plugin API is based on messages. Oracle Fusion Field Service sends the messages and the plugin receives them. Similarly, the plugin sends the messages and Oracle Fusion Field Service receives them. JavaScript code uses the window.postMessage() method to send the messages, and the receiver receives by subscribing to the window.addEventListener() method.

Transport

Plugin API is based on **postMessages - *https://developer.mozilla.org/ru/docs/Web/API/Window/postMessage*** , that can be sent by OFS and received by Plugin and vice versa.

postMessages are sent by JavaScript code using *window.postMessage()* method, and received by subscribing to messages using *window.addEventListener()*.

Debugging

Use:

```
setOfscDebugModes(true, 'pluginApi');
```

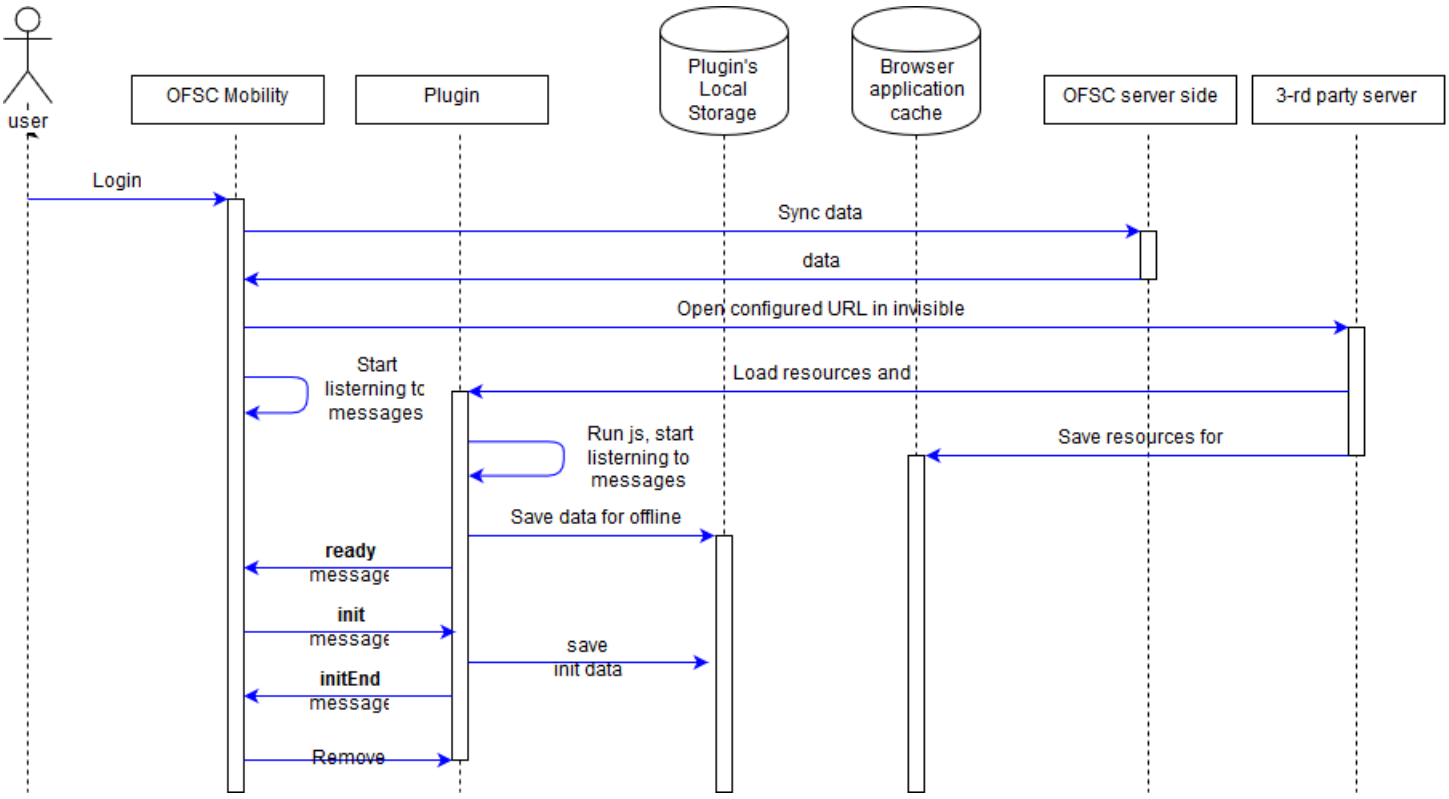
in Browser Console to enable debugging mode. It will show information about stages of initialization and all post messages between platform and plugin. To disable the debugging mode use:

```
setOfscDebugModes(false, 'pluginApi');
```

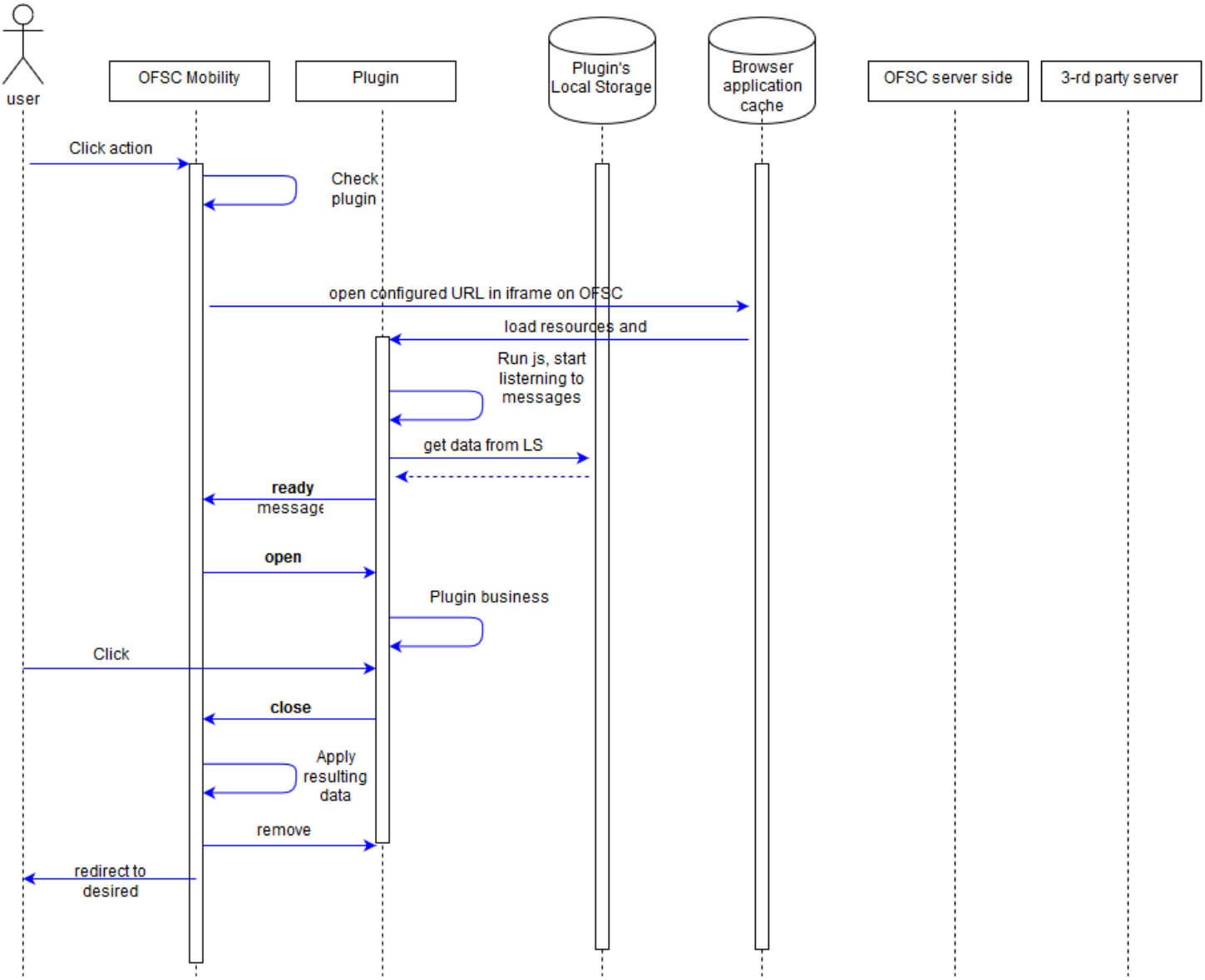
Flowcharts

This topic provides the flowcharts that show how the plugin framework works.

This flowchart shows the plugin’s initialization flow:



This flowchart shows the plugin's main flow:


user
OFSC Mobility
Plugin
Plugin's Local Storage
Browser application cache
OFSC server side
3-rd party server
Click action
Check plugin
open configured URL in iframe on OFSC
load resources and
Run js, start listerning to messages
get data from LS
ready message
open
Plugin business
Click
close
Apply resulting data
remove
redirect to desired

2 Plugin API Messages

Message Formats

You can send a message to a plugin as a string, containing serialized JSON data, or as a raw JavaScript object.

Here's an example of a message that's sent as a string containing serialized JSON data:

```
window.parent.postMessage('{"apiVersion":1,"method":"close","activity":{"cname":"John"}}', targetOrigin);
```

Here's an example of a message that's sent as a raw JavaScript object:

```
window.parent.postMessage({
 apiVersion: 1,
 method: 'close',
 activity: {
 cname: 'John'
 }
}, targetOrigin);
```

You can update file properties only by using a JavaScript object as message data. See File properties for details. Similarly, the plugin must process the messages that it receives. Oracle Fusion Field Service always sends the data to the plugin as a serialized JSON string and never as a raw object. For example:

```
function getPostMessageData(event)
{
 var data = JSON.parse(event.data);
 switch (data.method)
 {
 case 'open':
 pluginOpen(data);
 break;
 default:
 showError();
 }
};

window.addEventListener("message", _getPostMessageData, false);
```

JSON data is an object (hash) of a defined format, and contains common fields (that describe the message itself) and fields that are specific for different 'methods' (for example, that hold Oracle Fusion Field Service entities data), for example:

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activity",
 "resource": {
 "pid": 5000038
 },
 "inventoryList": {
 "20997919": {
 "invid": 20997919,
 "inv_pid": 5000038,
 }
 }
}
```

Where:

- **apiVersion, method**: Common fields.
- **entity**: Name of the Oracle Fusion Field Service entity that's to be processed by the plugin. Available only for the 'open' method.
- **resource, inventoryList**: Entity data collections. Available only for 'open' and 'close' methods.

Common Fields

- **apiVersion:** Version of the plugin API that's used for interaction between and Oracle Fusion Field Service and the plugin. Available methods and data depend on it. This is a required parameter. You must include this parameter in the message for the plugin to be processed without any errors.
- **method:** Describes the action initiated by Oracle Fusion Field Service or the plugin, and the actions that should be performed by other side.

Related Topics

- Does the plugin apply limits to property values?

Available Methods

Initiated by Oracle Fusion Field Service:

- *init*: The plugin is loaded when Oracle Fusion Field Service is initialized, and the initialization data can be stored by the plugin.
- *open*: The plugin content is to be shown on the page in Oracle Fusion Field Service.
- *error*: Data submitted by the plugin is invalid, or some internal errors have occurred.
- *wakeup*: Oracle Fusion Field Service detected that a connection to a server is available and the plugin has requested to be activated on this event.
- *callProcedureResult*: Oracle Fusion Field Service returns the result of running the procedure (RPC).

Initiated by the plugin:

- *ready*: The plugin is loaded and is ready to receive messages.
- *initEnd*: The Plugin finished processing the initialization data.
- *close*: The plugin submits data and its window will be closed if data is valid.
- update: The plugin updates OFS entities data without closing its window.
- *sleep*: Lets the plugin finish the background activity (started by “wakeup” method) when all data is synchronized with the server. If the plugin is not able to synchronize with its own server because network connectivity is not available, it can notify Oracle Fusion Field Service to wake it up in the background when the connectivity is available.
- *callProcedure*: The plugin requests the Oracle Fusion Field Service to run the procedure (RPC) without closing the plugin's window.

ready Method

A message that includes the *ready* method indicates that all the resources needed for the plugin's functioning are loaded; the plugin has started listening to the messages from Oracle Fusion Field Service Core Application and is ready to process them.

The plugin sends the *ready* message every time it's loaded. When Oracle Fusion Field Service Core Application is being loaded or a page is being refreshed, the message, *'Preparing data for offline'* is shown to the user. The message is displayed until every plugin sends the *ready* message. If a plugin doesn't send the *ready* message within 120 seconds, Oracle Fusion Field Service Core Application marks the plugin as *'Failed to init'* and shows the corresponding message to the user (*"This plugin has not been loaded: ..."*). If a plugin is marked as *'Failed to init'*, Oracle Fusion Field Service Core Application tries to re-initialize it when the user opens the plugin by clicking a button. When a user opens the plugin through a button, the message, *'Screen is loading. Please wait.'* is shown. This message is shown until the plugin sends the *ready* message. If a plugin doesn't send the *ready* message within 120 seconds, Oracle Fusion Field Service Core Application displays the message, *'The plugin has not loaded.'*.

Message Format

Messages of this method have this format:

```
{
 "apiVersion": 1,
 "method": "ready",
 "sendInitData": true,
 "showHeader": true,
 "enableBackButton": true
}
```

Initialization

If you have set the field '**sendInitData**' to **true**, and Oracle Fusion Field Service Core Application has loaded the plugin while initializing (not when the user opens the plugin), Oracle Fusion Field Service Core Application sends an additional *init* message with initialization data to the plugin such as attribute description. It destroys the plugin's iframe only when the plugin sends the **initEnd** message. If you have not set the field '**sendInitData**' or set it as not equal to true, and Oracle Fusion Field Service Core Application loads the plugin while initializing Oracle Fusion Field Service Core Application (not when the user opens the plugin), Oracle Fusion Field Service Core Application destroys the plugin's iframe immediately after the '**ready**' message received. If the user opens the plugin, the field '**sendInitData**' is ignored.

Amount of Data Sent to Plug-In

Oracle Fusion Field Service Core Application sends the *open* message with all data available for entity collections, according to the context layout, where the plugin's button is placed and depending on how the Available Properties section is configured. For example, if a plugin is opened from the Activity List page, the data of all the activities for the selected day's queue is sent with the data of all non-scheduled activities of the selected resource. Oracle Fusion Field Service Core Application collects and serializes the data, and the plugin un-serializes it, thereby increasing the loading time for the plugin. To reduce the amount of this data, the optional parameter, " ***dataItems*** " is supported for the *ready* message. The value of this parameter defines the items that are present in the entity collections. Using this parameter, a plugin can prevent Oracle Fusion Field Service Core Application from sending certain items in the available entity collections, but it can't broaden the set of entity collections that is sent to the plugin. This set is predefined and depends on the page from which the plugin is opened.

Format of dataItems

dataItems is an array, where each item is a label of a certain data subset. If the item with the label of some subset is absent in this array, Oracle Fusion Field Service Core Application does not send the corresponding items in the entity

collections of the open message. If dataItems is not set in the *ready* message, no filtering is applied and the full data set is sent to the plugin. Here is the list of available keys and data subsets:

Key	Affected Collections	Description
team	team	Information about assistants and resources who the current resource is assisting to
resource	resource	Properties of the current selected resource
scheduledActivities	activityList	Activities, scheduled (belongs to the queue) for the selected date
nonScheduledActivities	activityList	Non-scheduled activities that don't belong to any date's queue
resourceInventories	inventoryList	Inventories in the "provider" pool
installedInventories	inventoryList	Inventories in the "install" pool
deinstalledInventories	inventoryList	Inventories in the "deinstall" pool
customerInventories	inventoryList	Inventories in the "customer" pool

Example of a Message with "dataItems"

```
{
 "apiVersion": 1,
 "method": "ready",
 "sendInitData": true,
 "dataItems": ["team",
 "resource",
 "scheduledActivites",
 "nonScheduledActivites",
 "resourceInventories",
 "installedInventories",
 "deinstalledInventories",
 "customerInventories"
 ]
}
```

Restriction of Navigation with the ready Method

Users can click **Back** on the header of the page or the browser to exit the plugin page. Some business flows require mobile users to stay on the required action page until the action is finished. Further, users may be required to go to a specific page after leaving the current page. For example, during the completion flow, a mobile user is required to fill a custom checklist and only after filling it that the regular completion page is displayed. To support such business flows, the parameters "showHeader" and "enableBackButton" are available for the ready method. Here are the details of the "*showHeader*" and "*enableBackButton*" parameters:

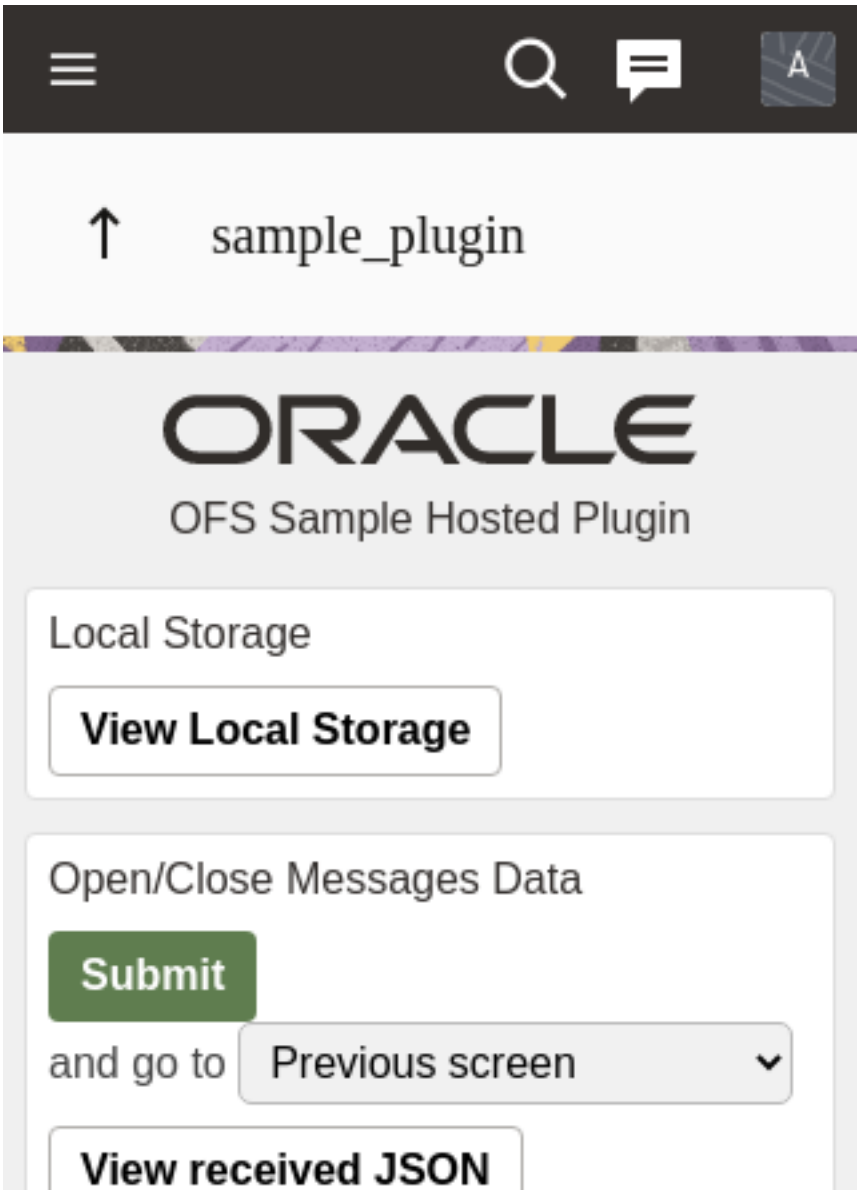
Param Name	Mandatory	Default Value	Type	Description
enableBackButton	No	true	boolean	If the flag value is set to true, then the Oracle Fusion Field Service Core Application "< Back" button is shown and its navigation is not locked. If the flag value is set to false, then the Oracle Fusion Field Service Core Application "< Back" button is hidden and the navigation is

Param Name	Mandatory	Default Value	Type	Description
				locked. (The browser's native Back and Forward buttons are blocked, notification is shown if the user uses the native Back button).
showHeader	No	true	boolean	If the flag value is set to true, then the Oracle Fusion Field Service Core Application header is shown. If the flag value is set to false, then the header is hidden.

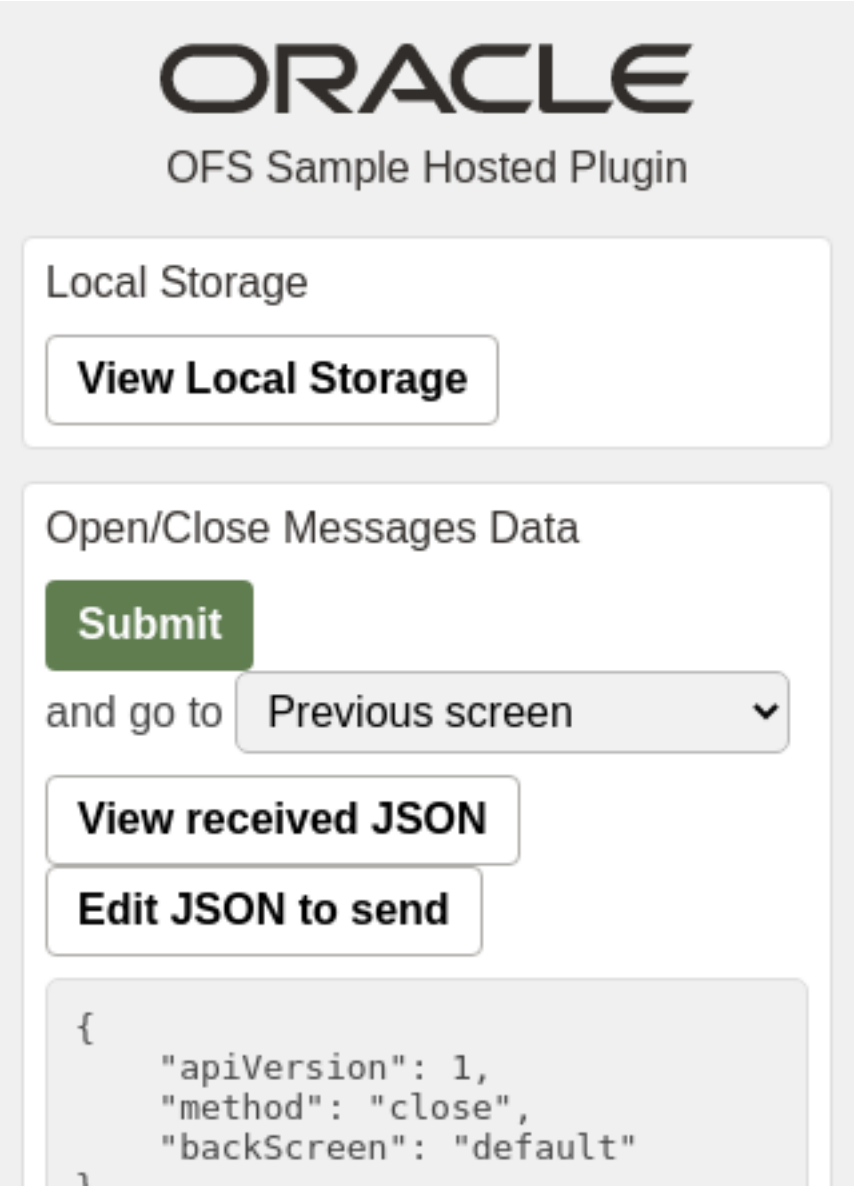
Users can be redirected to a specific page after the plugin is closed. See the *close* method for details. This table shows the behavior of the plugin for different values of the showHeader param.

enableBackButton Value	showHeader = True	showHeader = False
enableBackButton = True	In this scenario: • Navigation is possible • Global header is visible • Page header is visible	In this scenario: • Navigation is possible • Global header is not visible • Page header is not visible
enableBackButton = False	In this scenario: • Navigation is not possible • Global header is not visible • Page header is visible	In this scenario: • Navigation is not possible • Global header is not visible • Page header is not visible

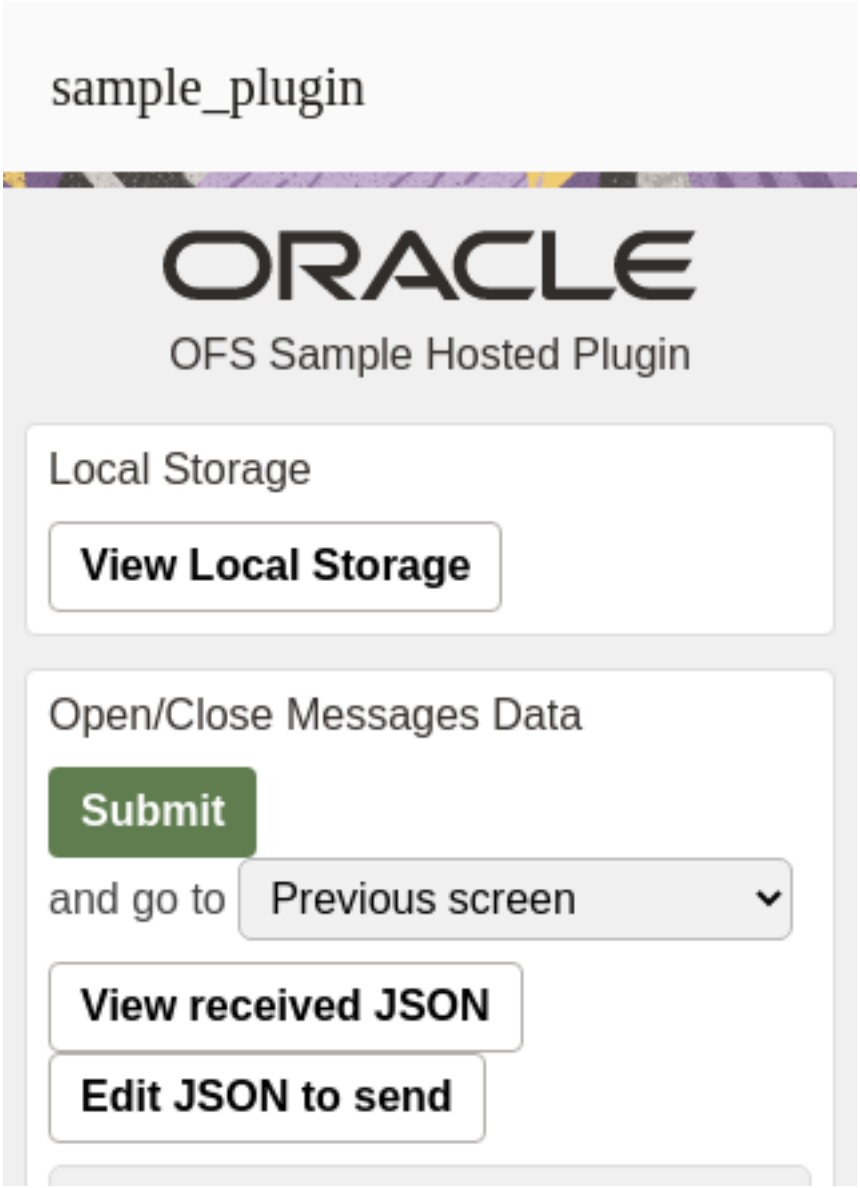
This screenshot shows the scenario when **enableBackButton** is *True* and **showHeader** is *True*.



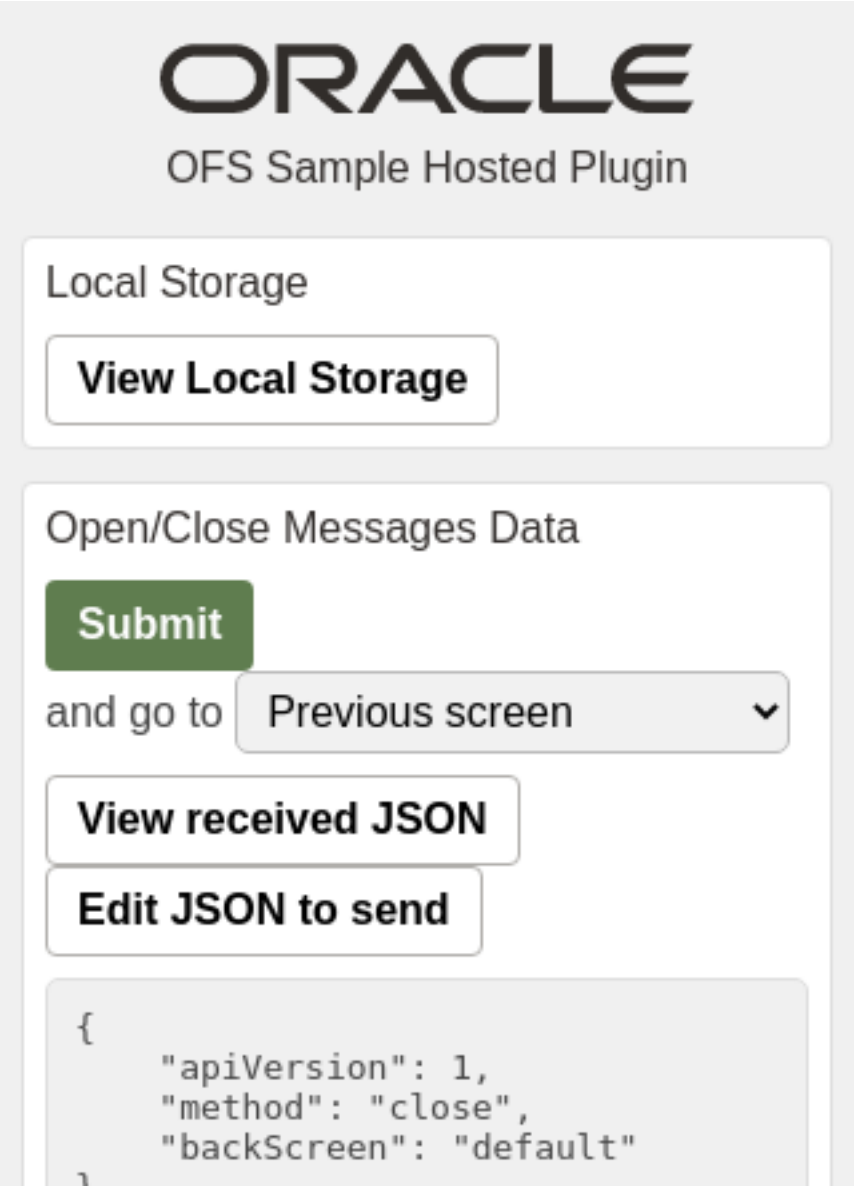
This screenshot shows the scenario when **enableBackButton** is *True* and **showHeader** is *False*.



This screenshot shows the scenario when enableBackButton is False and showHeader is True.



This screenshot shows the scenario when **enableBackButton** is *False* and **showHeader** is *False*.



Examples of the "ready" Message

```
// the header is shown but the "back" button is hidden:
{
 "apiVersion": 1,
 "method": "ready",
 "showHeader": true,
 "enableBackButton": false
}

// the header is hidden but the user can go back using browser's back button:
{
 "apiVersion": 1,
 "method": "ready",
```

```
 "showHeader": false,
 "enableBackButton": true
}

// the header is hidden and the user can leave the plugin only when the plugin sends the "close" message via
 Plugin API (the browser's back button does not work):
{
 "apiVersion": 1,
 "method": "ready",
 "showHeader": false,
 "enableBackButton": false
}
```

sendMessageAsJsObject - use JSON or JS object in transfer

PostMessage technology on early stage had some limitations in some browsers to type of data that is transferred. So on very beginning of Plugin API was decided to use JSON string as simple type of data to communication between Plugin and OFS. During some time OFS start to support receiving of JS object instead of JSON string, but responses still were sent as JSON string.

"TakePhoto" procedure which requires to send photo as a file from OFS to plugin as a result of procedure calling. So in 23.5 was added functionality to send data as JS object from OFS to Plugin. In order to keep backward compatibility it was added "sendMessageAsJsObject" flag that by default is equal to true.

Benefits that brings using of JS object:

- ability to send files from OFS to Plugin
- faster transfer, because it is absent parsing and validation of JSON format

It is highly recommended to use **"sendMessageAsJsObject": true** in all modern plugins

OFS keep information about choosen variant of transfer in browser's memory. So if "**sendMessageAsJsObject":true** flag was sent, OFS will transfer data as a JS object until the "**sendMessageAsJsObject": false** will be sent or while the browser tab will be opened.

Examples of the "ready" message

```
{
 "apiVersion": 1,
 "method": "ready",
 "sendMessageAsJsObject": true
}
```

Related Topics

- How do I use the open method in the Plugin API?
- close Method

init Method

A message with the init method indicates that Oracle Fusion Field Service Core Application has started initializing the plugin.

The init message contains these fields:

```
{
 "apiVersion": 1,
```

```
 "method": "init",
 "attributeDescription": {}
}
```

attributeDescription

The attributeDescription field is an object that contains the descriptions of all the properties that are configured for the plugin.

List of all buttons that are configured for a plugin is sent to the plugin in the 'buttons' field of the 'init' message. This field is a list of objects that contains the 'buttonId' and 'params' fields. buttonId is the 'context layout item id' of the button. 'params' is an object that represents the parameters that are configured for the corresponding context layout item. Each enum items of the "aworktype" (Activity type) property contains the "features" object. If the "Enable segmenting and extended duration" option is enabled for the activity type, the "features" object contains the "isSegmentingEnabled" field with a value equal to true.

Example of the init method

```
{
 "apiVersion": 1,
 "method": "init",
 "attributeDescription": {
 "pid": {
 "entity": "ENTITY_PROVIDER",
 "fieldType": "field",
 "gui": "text",
 "label": "pid",
 "title": "ID",
 "type": "string",
 "access": "READ_ONLY"
 },
 "pname": {
 "entity": "ENTITY_PROVIDER",
 "fieldType": "field",
 "gui": "text",
 "label": "pname",
 "title": "Name",
 "type": "string",
 "access": "READ_ONLY"
 },
 "invtype": {
 "entity": "ENTITY_INVENTORY",
 "enum": {
 "NT": {
 "label": "NT",
 "text": "Internet"
 },
 "DT": {
 "label": "DT",
 "text": "Digital Telephony"
 },
 "AT": {
 "label": "AT",
 "text": "Analog Telephony"
 }
 },
 "fieldType": "field",
 "gui": "combobox",
 "label": "invtype",
 "title": "Inventory Type",
 "type": "enum",
 "access": "READ_WRITE"
 },
 "invpool": {
 "entity": "ENTITY_INVENTORY",
```

```
"fieldType": "field",
"gui": "text",
"label": "invpool",
"title": "Inventory pool",
"type": "string",
"access": "READ_WRITE"
},
"invsn": {
"entity": "ENTITY_INVENTORY",
"fieldType": "field",
"gui": "text",
"label": "invsn",
"title": "Serial Number",
"type": "string",
"access": "READ_WRITE"
},
"aid": {
"entity": "ENTITY_ACTIVITY",
"fieldType": "field",
"gui": "text",
"label": "aid",
"title": "Activity ID",
"type": "string",
"access": "READ_ONLY"
},
"aworktype": {
"entity": "ENTITY_ACTIVITY",
"enum": {
"27": {
"label": "27",
"text": "Multi-type Unwired Installs"
},
"55": {
"label": "55",
"text": "Maintenance"
},
"WH": {
"label": "WH",
"text": "Warehouse Activity"
},
"installation": {
"label": "installation",
"text": "Installation"
},
},
"fieldType": "field",
"groups": [
{
"label": "task",
"text": "Task",
"items": [
"55"
]
},
{
"label": "teamwork",
"text": "Teamwork",
"items": [
"teamwork_type"
]
},
{
"label": "internal",
"text": "Internal",
"items": [
"WH"
```

```
]
},
{
"label": "customer",
"text": "Customer",
"items": [
"27",
"installation"
]
}
],
"gui": "grouped-combobox",
"label": "aworktype",
"title": "Activity type",
"type": "enum",
"access": "READ_ONLY"
},
"no_ports": {
"entity": "ENTITY_ACTIVITY",
"fieldType": "property",
"gui": "text",
"label": "no_ports",
"title": "# Ports",
"type": "int",
"access": "READ_WRITE"
},
"CANCEL_REASON": {
"entity": "ENTITY_ACTIVITY",
"enum": {
"1": {
"text": "M1 REQUESTED BY ISP"
},
"2": {
"text": "M2 PC INADEQUATE"
},
"3": {
"text": "M8 - TAP CHANGEOUT REQUIRED"
},
"4": {
"text": "SS - STORM SERVICE CALL FOLLOW-UP"
},
"5": {
"text": "W1 - JUST BROWSING AT THIS TIME"
}
},
"fieldType": "property",
"gui": "combobox",
"label": "CANCEL_REASON",
"title": "Cancellation Reason",
"type": "enum",
"access": "READ_WRITE"
}
}
}
```

Activity type features in "init" message

The "attributeDescription" field of "init" message contains "features" object inside enum items of the "aworktype" (activity type) property. If the "Enable segmenting and extended duration" option is enabled for the activity type, the "features" object contains "isSegmentingEnabled" with value equal to true.

Example of "init" message

```
{
 "apiVersion": 1,
```

```
"method": "init",
"attributeDescription": {
"WO_COMMENTS": {
"fieldType": "property",
"entity": "ENTITY_ACTIVITY",
"gui": "text",
"label": "WO_COMMENTS",
"title": "WO Comments",
"type": "string",
"access": "READ_WRITE",
"lines": 4
},
"aworktype": {
"fieldType": "field",
"entity": "ENTITY_ACTIVITY",
"gui": "grouped-combobox",
"label": "aworktype",
"title": "Activity type",
"type": "enum",
"access": "READ_WRITE",
"enum": {
"5": {
"label": "5",
"text": "Inv Pick up",
"features": {}
},

"VH": {
"label": "VH",
"text": "Vehicle Maintenance",
"features": {}
},
"Multiday": {
"label": "Multiday",
"text": "Multi-day activities",
"features": {
"isSegmentingEnabled": true
}
},
},
"groups": [
{
"label": "internal",
"text": "Internal",
"items": [
"VH"
]
},
{
"label": "customer",
"text": "Customer",
"items": [
"5",
"Multiday"
]
}
]
}
},
"buttons": [
{
"buttonId": "20348",
"params": {}
}
]
```

```
}
```

initEnd Method

The messages with this method indicate that the plugin has finished processing the initialization data.

When Oracle Fusion Field Service Core Application receives the initEnd message from a plugin, it destroys the plugin's iframe. The message may also contain the optional 'wakeupNeeded' field, which allows to continue background activity of the plugin after a page is reloaded. See the description of the wakeup method for details. You can change the appearance of the plugin tile (the icon image), status text, and color using the optional iconData parameter. See the corresponding section for details.

Example of the initEnd message

```
{
 "apiVersion": 1,
 "method": "initEnd",
 "wakeupNeeded": false,
 "iconData": {
 "text": "89"
 }
}
```

Oracle Fusion Field Service ignores all other fields.

Related Topics

- wakeup Message

open Method

When a user opens a plugin through a button, a message with the open method is sent to the plugin after Oracle Fusion Field Service receives the ready message. The response of the 'open' method contains the 'user' item, includes the 'main_resource_id' field that represents the resource which is referenced to the current user. Similarly, the response of the 'open' method includes the 'team' item, which contains information about teamwork. The response of the 'open' method also contains the 'queue' key with current queue state (activated, not activated, or deactivated). The 'resource' key contains the time-related fields such as the current resource's time, resource's time zone difference, and the difference between a device's clock and UTC.

The open message contains entity collections, for example, data of available Oracle Fusion Field Service entities such as activities and inventories. See Available entities and data collections for details. The 'dataItems' option of the 'ready' method controls the availability of the 'team' item. The 'team' item is not sent if the plugin is opened from the Main menu. The response of the 'open' method is extended with the activity and inventory lists when they are available.

Example of open message

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activity",
 "user": {
 "uid": 38,
 "ulogin": "rayner",
 "uname": "RAYNER, Faye",
```

```
"format": {
date: "m/d/y",
long_date: "l, F jS, Y",
time: "h:i A",
datetime: "m/d/y h:i A"
},
"su_zid": 4,
"week_start": 0,
"ulanguage": 1,
"languageCode": "en",
"design_theme": 11,
"allow_vibration": 1,
"allow_desktop_notifications": 0,
"sound_theme": 11,
"providers": [
"5000038",
"5000039"
],
"main_resource_id": 5000038
},
"team": {
"assistingTo": {
"3000001": [
"3000008",
"3000037"
],
"3000015": []
},
"assistingMe": [
"3000003",
"3000008"
],
"teamMembers": {
"3000001": {
"uid": 1000001,
"external_id": "resource_1",
"pname": "Resource 1",
"pactive": 1
},
"3000003": {
"uid": 1000003,
"external_id": "resource_13",
"pname": "Resource 3",
"pactive": 1
},
"3000008": {
"uid": 1000008,
"external_id": "resource_8",
"pname": "Resource 8",
"pactive": 1
},
"3000015": {
"uid": 1000015,
"external_id": "resource_15",
"pname": "Resource 15",
"pactive": 1
},
"3000037": {
"uid": 1000037,
"external_id": "resource_37",
"pname": "Resource 37",
"pactive": 1
}
}
},
"resource": {
```

```
"pid": 5000038,
"pname": "RAYNER, Faye",
"external_id": "55038",
"gender": "1"
},
"activityiList": {
"3956534": {
"WO_COMMENTS": "AUTOMATIC TRANSFER WORK ORDER\n\n",
"astatus": "started",
"aid": 3956534
}
},
"activity": {
"WO_COMMENTS": "AUTOMATIC TRANSFER WORK ORDER\n\n",
"astatus": "started",
"aid": 3956534
},
"inventoryList": {
"20997919": {
"invid": 20997919,
"inv_aid": 3956534,
"inv_pid": 5000038,
"invpool": "install",
"invsn": "SABDFWKNZ"
},
"20998078": {
"invid": 20998078,
"inv_aid": 3956532,
"invpool": "customer",
"invsn": "5CTBME4AW090379"
},
"20998080": {
"invid": 20998080,
"inv_aid": 3956533,
"invpool": "customer",
"invsn": "SABGZTWGM"
}
}
}
```


Example of the open message for a plugin opened from the Main menu (only 'user' collection is available)


```
{
"apiVersion": 1,
"method": "open",
"entity": "user",
"user": {
"uid": 2315,
"ulogin": "admin",
"uname": "Admin",
"format": {
"date": "m/d/y",
"long_date": "l, F jS, Y",
"time": "h:i A",
"datetime": "m/d/y h:i A"
},
"week_start": 0,
"ulanguage": 1,
"language": "en",
"design_theme": 1,
"allow_vibration": 0,
"allow_desktop_notifications": 0,
"sound_theme": 0,
"providers": [
2
]
```

```
 }
}
```

Example of the open message for a plugin opened from the Parts Catalog

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "partsCatalogItem",
 "team": {
 "assistingTo": {},
 "assistingMe": [],
 "teamMembers": {}
 },
 "user": {
 "uid": 2315,
 "ulogin": "admin",
 "uname": "Admin",
 "format": {
 "date": "m/d/y",
 "long_date": "l, F jS, Y",
 "time": "h:i A",
 "datetime": "m/d/y h:i A"
 },
 "week_start": 0,
 "ulanguage": 1,
 "languageCode": "en",
 "design_theme": 1,
 "allow_vibration": 0,
 "allow_desktop_notifications": 0,
 "sound_theme": 0,
 "providers": [
 2
 ]
 },
 "partsCatalogItem": {
 "catalogId": 2,
 "label": "a5123-df"
 },
 "resource": {
 "pid": 5000038,
 "pname": "RAYNER, Faye",
 "external_id": "55038",
 "gender": "1"
 },
 "activityiList": {
 "3956534": {
 "WO_COMMENTS": "AUTOMATIC TRANSFER WORK ORDER\n\n",
 "astatus": "started",
 "aid": 3956534
 }
 },
 "inventoryList": {
 "20997919": {
 "invid": 20997919,
 "inv_aid": 3956534,
 "inv_pid": 5000038,
 "invpool": "install",
 "invsn": "SABDFWKNZ"
 }
 }
}
```

Structure of the 'team' collection in the ‘open’ method when teamwork is not set:

```
"team": {
 "assistingTo": {},
```

```
 "assistingMe": [],
 "teamMembers": {}
}
```

Structure of the 'team' collection in the ‘open’ method when teamwork is set:

```
"team": {
 "assistingTo": { - object with list of resources who I am assisting to
 "3000001": [ - array with list of additional resources who is assisting to user who I am assisting to
 (current user 3000035 is absent in this list!)
 "3000008", - resource ID who is also assisting to resource 3000001
 "3000037"
 ],
 "3000015": []
 }
 "assistingMe": [ - array with list of resources who is assisting me
 "3000003", - resource ID who is assisting to me
 "3000008"
 ]
 "teamMembers": { - object with information of all team members
 "3000001": {
 "uid": 1000001, - the resource is main resource for this user ID
 "external_id": "resource_1", - resource external ID
 "pname": "Resource 1", - resource name
 },
 "3000003": {
 "uid": 1000003,
 "external_id": "resource_13",
 "pname": "Resource 3"
 },
 "3000008": {
 "uid": 1000008,
 "external_id": "resource_8",
 "pname": "Resource 8"
 },
 "3000015": {
 "uid": 1000015,
 "external_id": "resource_15",
 "pname": "Resource 15"
 },
 "3000037": {
 "uid": 1000037,
 "external_id": "resource_37",
 "pname": "Resource 37"
 }
 }
}
```

Example of "open" message for Plugin opened via deep link

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "resource": {},
 "team": {
 "assistingTo": {},
 "assistingMe": [],
 "teamMembers": {}
 },
 "user": {
 "uid": 2315,
 "ulogin": "admin",
 "uname": "Admin",
 "format": {
```

```
"date": "m/d/y",
"long_date": "l, F jS, Y",
"time": "h:i A",
"datetime": "m/d/y h:i A"
},
"week_start": 0,
"ulanguage": 1,
"languageCode": "en",
"design_theme": 1,
"allow_vibration": 0,
"allow_desktop_notifications": 0,
"sound_theme": 0,
"providers": [
2
]
},
"activityList": {
"4225438": {
"aid": "4225438"
},
"4225439": {
"aid": "4225439"
}
},
"inventoryList": {
"21064417": {
"invid": "21064417"
},
"21064418": {
"invid": "21064418"
}
},
"openParams": {},
"externalData": {
"zipcodes": ["35801", "06101", "62701"],
"status": "completed"
}
}
```


Available Entities and Data Collections

The field 'entity' and entity data collections are available only for 'open' and 'close' methods. The value of the special 'entity' field depends on the Oracle Fusion Field Service Core Application page from which the user opens the plugin. Availability of entity data collections that are sent within the message data, depends on the value of 'entity'. this table gives the available entities and data collections for the *open* method:

Page	Entity Field Value	Available Collections
Main menu Team Map	user	user
Activity List Route Map	activitiyList	user team queue resource activitiyList
Activity List -> Inventory List	inventoryList	

Page	Entity Field Value	Available Collections
		inventoryList
Activity List -> Activity Details	activity	user
Activity List -> Activity Details -> Inventory List	activityInventoryList	team queue resource activityList activity inventoryList
Activity List -> Inventory List -> Inventory Details	inventory	user team queue resource activitiyList inventoryList inventory
Activity List -> Activity Details -> Inventory List -> Inventory Details	activityInventory	user team queue resource activityList activity inventoryList inventory
Inventory Search -> Parts Catalog Item Details	partsCatalogItem	user team

Page	Entity Field Value	Available Collections
		queue resource activitiyList inventoryList partsCatalogItem

Entity Data Collections

- **team**: Information about assistants and resources who are assisting to the current resource
- **resource**: Element in the resource tree representing a defined company asset
- **activity**: Entity of Oracle Fusion Field Service that represents any time-consuming activity of the resource
- **activityList**: Activity list
- **inventory**: Equipment that can be installed or deinstalled during an activity
- **inventoryList**: Inventory list
- **user**: User who has currently logged in to Oracle Fusion Field Service Core Application and opens the plugin
- **partsCatalogItem**: Information that identifies the parts catalog item, so it can be retrieved using the getParts procedure

Note: The 'team', 'resource', 'user', and 'partsCatalogItem' collections can't be updated through the plugin API and are ignored if they're sent with the 'close' message.

Availability of activity, inventory, and resource properties depends on the configuration of the plugin. See Available Properties for details.

Available Fields for 'user' Entity Collection

The available properties for this entity are predefined and do not depend on the configuration of the plugin. this table provides the available fields for the ‘user’ entity collection:

Field	Type	Example Value	Description
uid	Number	2315	Internal id of user
ulogin	String	admin	Login
uname	String	Admin	Name
format	Object<String, String>	`{ "date": "m/d/y", "long_date": "l, F jS, Y", "time": "h:i A", "datetime": "m/d/y h:i A" }`	Collection of date format strings in the PHP's style

Field	Type	Example Value	Description
su_zid	Number	2	Time Zone id
week_start	Number	0	Week start day (0-6) 0 - Sunday, 1 - Monday
ulanguage	Number	1	Language id (1 - English)
languageCode	String	en	Two-letter code for the language
design_theme	Number	1	Design theme ID
allow_vibration	Number	0	1 - Vibration on mobile devices is allowed, 0 - disallowed
allow_desktop_notifications	Number	0	1 - Browser desktop notifications are allowed, 0 - disallowed
sound_theme	Number	0	Sound notification settings 0 - Off, 1 - Quiet, 2 - Loud, 3 - Persistent
providers	Array<Number>	[38, 3000001]	List of resources, that are visible to user, excluding their descendants
main_resource_id (optional)	Number	1111	Resource ID, which is set as the main resource

Available Fields for 'activity' Entity Collection

This table provides the available fields for the ‘activity’ entity collection:

Field	Description
cname	Name
caddress	Address
ccity	City
czip	ZIP/Postal Code
cstate	State
customer_number	Account Number
c_zid	Time Zone
cphone	Phone
cemail	Email

Field	Description
ccell	Cellular Phone
atype	Activity Type
position_in_route	Position in Route
aworktype	Activity type
time_slot	Time Slot
service_window	Service Window
appt_number	Work Order
clanguage	Message Language
cmessagetime	Reminder
activity_workskills	Work Skill
length	Duration
ETA	Start
astatus	Activity status
aid	Activity ID
end_time	End
delivery_window	Delivery Window
acoord_status	Coordinate Status
acoord_x	Coordinate X
acoord_y	Coordinate Y
travel	Traveling Time
sla_window_start	SLA Start
sla_window_end	SLA End
atime_of_booking	Activity Time of Booking
atime_of_assignment	Activity Time of Assignment
activity_flow	Activity workflow

Non-Available Fields for 'activity' Entity Collection

This table provides the fields that are not available for the ‘activity’ entity collection:

Field	Description
aworkzone	Work Zone
time_delivered	Time Notified
eta_end_time	Start - End
date	Date
pid	Resource ID
apoints	Points
atravelarea	Travel Area
activity_capacity_categories	Capacity Categories
activity_alerts	Alerts
activity_compliance	Compliance Alerts
auto_routed_to_provider_id	Auto-Routed to Resource
auto_routed_to_date	Auto-Routed to Date
first_manual_operation	First Manual Operation
first_manual_operation_user_id	First Manual Operation Performed by User
first_manual_operation_interface	First Manual Operation Interface
auto_routed_to_provider_name	Auto-Routed to Resource (Name)
first_manual_operation_user_name	First Manual Operation Performed by User (Name)
first_manual_operation_user_login	First Manual Operation Performed by User (Login)
access_hours	Access Hours
access_schedule	Access Schedule

'activity_flow'

The "activity_flow" field can be set from the plugin, or it will be calculated dynamically based on the workflow conditions created. The "activity_flow" field can be set for only those activities in status '*Pending*' or '*En route*'. Also, the "activity_flow" field can be set when the activity status is changed from '*Pending*' or '*En route*' to '*Started*'. The dynamic calculation works when the activity status is changed from '*Pending*' or '*En route*' to '*Started*' and "activity_flow" field was not ser previously; the application associates the workflow with the activity permanently, saves the workflow ID into the "activity_flow" field, and stops recalculation. It is prohibited to change the "activity_flow" field after an activity is started; a corresponding error message will be displayed.

'masterActivityId'

Each segment of a segmentable activity has a field called "masterActivityId". It's and id of the main activity which the segment belongs to. This field may be usable in a scenario when a plugin needs to deinstall inventory from a segmentable activity because customer pool inventory is assigned to the main segmentable activity (via "inv_aid" field) which is absent in the activity list. Using the "masterActivityId" activity field, the plugin can filter inventories from the "inventoryList" which belong to the main segmentable activity.

Example of an "open" message which contains segments of a segmentable activity and customer pool inventory


```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "user": {
 "allow_desktop_notifications": 1,
 "allow_vibration": 1,
 "design_theme": 11,
 "format": {
 "date": "m/d/y",
 "long_date": "l, F jS, Y",
 "time": "h:i A",
 "datetime": "m/d/y h:i A"
 },
 "providers": [
 2
 ],
 "sound_theme": 2,
 "su_zid": 15,
 "uid": 2315,
 "ulanguage": 1,
 "languageCode": "en",
 "ulogin": "admin",
 "uname": "Admin",
 "week_start": 0
 },
 "resource": {
 "pid": 3000037,
 "currentTime": "2021-08-30 05:21:23",
 "deviceUTCDiffSeconds": 0,
 "timeZoneDiffSeconds": -14400
 },
 "team": {
 "teamMembers": {},
 "assistingTo": {},
 "assistingMe": []
 },
 "queue": {
 "date": "2021-08-30",
 "status": "notActivated",
 "isActual": true
 },
 "activityList": {
 "4227119": {
 "acoord_status": null,
 "acoord_x": null,
 "acoord_y": null,
 "aworktype": "LU",
 "appt_number": null,
 "astatus": "pending",
 "aid": "4227119"
 },
 "4227120": {
 "acoord_status": null,
 "acoord_x": null,
 "acoord_y": null,
 "aworktype": "4",
```

```
"appt_number": null,
"astatus": "pending",
"aid": "4227120"
},
"4227122": {
"acoord_status": null,
"acoord_x": null,
"acoord_y": null,
"aworktype": "Multiday",
"appt_number": null,
"astatus": "pending",
"aid": "4227122",
"masterActivityId": 4227121
},
"4227123": {
"acoord_status": null,
"acoord_x": null,
"acoord_y": null,
"aworktype": "Multiday",
"appt_number": null,
"astatus": "pending",
"aid": "4227123",
"masterActivityId": 4227121
},
"4227133": {
"acoord_status": null,
"acoord_x": null,
"acoord_y": null,
"aworktype": "Multiday",
"appt_number": null,
"astatus": "pending",
"aid": "4227133",
"masterActivityId": 4227132
}
},
"inventoryList": {
"21258560": {
"invpool": "customer",
"invid": "21258560",
"inv_aid": 4227121,
"inv_pid": null,
"invsn": null,
"invtype": "EC",
"quantity": 42
},
"21258561": {
"invpool": "customer",
"invid": "21258561",
"inv_aid": 4227132,
"inv_pid": null,
"invsn": null,
"invtype": "HD12",
"quantity": 100
}
},
"buttonId": "20360",
"openParams": {}
}
```


Available Fields for 'inventory' Entity Collection

This table provides the available fields for the ‘inventory’ entity collection:

Field	Description
invsn	Serial Number
invpool	Inventory pool
invtype	Inventory Type
invid	Inventory Id
inv_aid	Activity Id
inv_pid	Resource Id
inv_change_invid	Changed Inventory ID
quantity	Quantity

Available Fields for 'resource' Entity Collection

This table provides the available fields for the 'resource' entity collection:

Field	Description
email	Email address
external_id	External ID
pdate_fid	Date format
pactive	Status
pid	ID
planguage	Message Language
pname	Name
pphone	Phone
ptime_fid	Time format
ptype	Resource type
time_zone	Time zone
currentTime	Current time in "YYYY-MM-DD hh:mm:ss" format in the resource's time zone at the time of generating the "open" message.
deviceUTCDiffSeconds	Difference between browser's time and UTC (server time) in seconds. A plugin can calculate the actual UTC time using this formula: UTC = Math.round(new Date().getTime() / 1000) - deviceUTCDiffSeconds.

Field	Description
timeZoneDiffSeconds	Provider's timezone diff in seconds at the time of generating the "open" message.

Unavailable Fields for 'resource' Entity Collection

This table provides the fields that are not available for the ‘resource’ entity collection:

Field	Description
alerts	Alerts
calendar	Calendar
oncall_calendar	On-call Calendar
organization_id	Organization
p_rprid	Routing profile
pcapacity_bucket	Use as Capacity Area
pending	Pending
pinitial_ratio	Initial Ratio for Activity Duration
queue_status	Queue status
reactivated	Reactivated
resource_capacity_categories	Capacity Categories
resource_effective_workskills	Effective Work Skills
resource_time_slots	Time slots
resource_workskills	Work Skills
resource_workzones	Work Zones
skip_days_for_stats	Working days left for reported data to start impacting duration estimations
total	Total

Available Fields for "partsCatalogItem" Entity Collection

This table provides the available fields for the ‘partsCatalogItem’ entity collection:

Field	Example Value	Description	Mandatory
catalogId	17	A unique identifier of a catalog which contains the item. Is returned by the getPartsCatalogsStructure procedure and is required by getParts procedure.	Yes

Field	Example Value	Description	Mandatory
label	a5123-df	A unique identifier of a part within a catalog. Is required by getParts procedure.	Yes

Available Fields for "user" Entity Collection

This table provides the available fields for the ‘user’ entity collection:

Field	Type	Example Value	Description
allow_desktop_notifications	Number	0	1 - Browser desktop notifications are allowed,0 - disallowed
allow_vibration	Number	0	1 - Vibration on mobile devices is allowed,0 - disallowed
design_theme	Number	1	Design theme ID

format	Object<String, String>	{
		"date": "m/d/y",
		"long_date": "l, F jS, Y",
		"time": "h:i A",
		"datetime": "m/d/y h:i A"
		}
			Collection of date format strings in the PHP's style

main_resource_id (optional)	Number	1111	Resource ID which is set as main resource
providers	Array<Number>	[38, 3000001]	List of resources, that are visible to user, excluding their descendants
sound_theme	Number	0	Sound notification settings.0 - Off, 1 - Quiet, 2 - Loud, 3 - Persistent
su_zid	Number	2	Time Zone id
uid	Number	2315	Internal id of user
ulanguage	Number	1	Language id (1 - English)
ulogin	String	admin	Login
uname	String	Admin	Name
week_start	Number	0	Week start day (0-6).

Field	Type	Example Value	Description
			0 - Sunday, 1 - Monday

Available Fields for "team" Entity Collection

This table provides the available fields for the ‘team’ entity collection:

Field	Description
assistingTo	Object with resources to assist (each item is an array with other resources who assists)
assistingMe	Array with IDs of assistants
teamMembers	Object as plain collection with information about assistants. The information consists of the following four fields: • external_id • pactive • pname • uid

close Method

The message format for the *close* method is similar to the *open* method. When Oracle Fusion Field Service receives a message from a plugin with the *close* method, it validates all the entity properties and their values, and applies the update only if no rules are violated. After updating, it closes the plugin. If there are violations, Oracle Fusion Field Service sends a message with the *error* method, which includes a list of errors (see example).

There's no need to send all the collections and properties received through the *open* method. The plugin can send only those entities and properties that have to be updated. The remaining entities and properties are left unchanged. Oracle Fusion Field Service Core Application updates only those entity collections that were sent with the *open* message. If the plugin sends extra entities with the *close* message, they're ignored. Oracle Fusion Field Service Core Application updates only the properties that are configured for the plugin. If the plugin sends extra properties with the *close* message, they're ignored. The property values that are sent are validated according to the type and attributes of the updated property. See Error codes for details.

Related Topics

- How do I use the open method in the Plugin API?

Error Types and Error Codes

This topic describes the error types and error codes related to the *close* message.

The error types related to the *close* message are:

Type	Occurs When	Available Message Fields
TYPE_ENTITY_ACTION	Requested action is inapplicable for the specified entity	• entity • entityId
TYPE_ENTITY_PROPERTY	Value of one of the properties, submitted by the plugin to be updated, has an invalid value or violates a business rule for the given entity and conditions	• entity • entityId • propertyLabel
TYPE_INTERNAL	Oracle Fusion Field Service is unable to process message due to: • Invalid format or contents of message, or • Unexpected internal error	NA

These error message fields are available:

- **entity**: Data for the listed entity is invalid ("activity" or "inventory")
- **entityId**: Id of the entity, for which the data is invalid (such as aid for activity and invid for inventory, for example, "10028719")
- **propertyLabel**: Label of the property, for which the value is invalid, for example, "customer_number", "WO_TYPE"

These error codes are available for the *close* message:

Code	Occurs When
TYPE_ENTITY_ACTION	
CODE_ACTION_ON_PAST_DATE_NOT_ALLOWED	The requested action is forbidden for the entity, if it's assigned for an archived (past) queue: • The plugin tries to update the activity properties that are in the past and overnight or overtime limit is elapsed • The plugin tries to update the inventory properties in the customer, installed, or deinstalled pool of activity, which is in the past and overnight or overtime limit is elapsed
TYPE_ENTITY_PROPERTY	
CODE_PROPERTY_VALUE_TOO_LARGE	Any of these: • Property type is 'field' and length of its value exceeds 119 UTF-16 codepoints • Property type is 'file', its GUI type is 'signature' and length of its value exceeds 102400 UTF-16 codepoints • Property is neither field nor signature and length of its value exceeds 32767 UTF-16 codepoints See the Property Value Length and Limits section for details.
CODE_MANDATORY_PROPERTY_EMPTY	Any of these:

Code	Occurs When
	For activity • 'astatus' value is empty For inventory • 'invpool' is 'install', 'deinstall' or 'customer' and 'inv_aid' value is empty • 'invpool' is 'install', 'deinstall' or 'provider' and 'inv_pid' value is empty • 'invpool' value is empty
CODE_ACTIVITY_STATUS_INVALID	Any of these: • 'astatus' of the activity doesn't equal one of: 'pending', 'started', 'enroute', 'complete', 'suspended', 'notdone', 'cancelled' • 'astatus' of the activity is 'enroute' and 'Enroute Support' option is disabled on the 'Business Rules' page. • Transition from the current activity status to the new one, specified in 'astatus', is not allowed. See the Possible transitions between activity statuses graph for details.
CODE_INVENTORY_POOL_INVALID	Any of these: • 'invpool' of inventory doesn't equal one of: 'customer', 'install', 'deinstall', 'provider' • Transition from the current inventory pool to the new one, specified in 'invpool', is not allowed See the Possible transitions between inventory pools graph for details.
CODE_INVENTORY_AID_INVALID	Any of these: • 'invpool' of inventory is 'provider' and 'inv_aid' value is not empty • 'invpool' of inventory is 'customer' or 'deinstall' and submitted 'inv_aid' value doesn't equal current value of 'inv_aid' • 'inv_aid' doesn't equal 'aid' of the started activity in the same queue and submitted 'inv_aid' value doesn't equal the current value of 'inv_aid'
CODE_INVENTORY_PID_INVALID	Any of these: • 'invpool' of the inventory is 'customer' and 'inv_pid' value is not empty • 'invpool' of the inventory is 'deinstall' and 'inv_pid' value doesn't equal the current value of 'inv_pid' and doesn't equal the 'pid' of the currently selected resource or the resource's teammates • 'invpool' of the inventory is 'provider', 'install', or 'deinstall' and submitted 'inv_pid' value doesn't equal the current value of 'inv_pid'
CODE_ACTIVITY_STATUS_INVALID_FOR_FUTURE	'astatus' is not 'pending' or 'cancelled' and activity is assigned for the day in future relative to the current date in the provider's time zone
CODE_ACTIVITY_STATUS_STARTED_ALREADY_IN_QUEUE	'astatus' is 'started' or 'enroute' and there is another started activity in the same queue
CODE_ACTIVITY_STATUS_ENROUTE_ALREADY_IN_QUEUE	'astatus' is 'started' or 'enroute' and there is another en route activity in the same queue
CODE_ACTIVITY_STATUS_INVALID_FOR_INACTIVE_QUEUE	'astatus' is not 'pending' or 'cancelled' and the activity is assigned for a queue that is not activated or is deactivated

Code	Occurs When
CODE_ACTIVITY_STATUS_INVALID_FOR_NON_TRAVEL_ACTIVITY	'astatus' is 'enroute' and the activity doesn't support "calculate travel" feature (is a non-travel activity)
CODE_ACTIVITY_STATUS_REORDERING_IS_NOT_ALLOWED	'astatus' is 'enroute', activity is ordered and is not first in the route, and "Allow activity reorder inside the route" option is not selected for the user type of the current user
TYPE_INTERNAL	
CODE_UNKNOWN	Oracle Fusion Field Service is unable to process the message due to: • Invalid format or contents of message, or • Unexpected internal error occurred
TYPE_MESSAGE_FORMAT	
CODE_METHOD_NOT_SUPPORTED	Any of these: • Wrong value in the method field • Broken sequence of the method calls (for example, something is sent after the 'close' message, or before the 'update' method is processed)

Related Topics

- Activity Status and Inventory Pool Changes

Property Value Length and Limits

Limits are applied to property values that are submitted by the plugin through the Plugin API for update. If a value length exceeds the limit, Oracle Fusion Field Service returns an error message as part of the message with the *error* method.

Fields (property type is 'field')

Maximum un-formatted data to store is 239 bytes. JavaScript uses UTF-16 for strings, so one Unicode character may take up 2 to 4 bytes. But, the String.length property uses UTF-16 code points for counting, which is 2 bytes. This means, the length of the string containing one 4-byte UTF-16 char is 2. So, only ceil(239/2) = 119 code points can be stored without truncating.

Signatures (property type is 'file' and GUI option is 'Signature')

We assume that the value contains only MIME-type and correct base64 string. So, each character takes up 2 bytes as JavaScript uses UTF-16. To avoid overflow of LocalStorage, each signature is limited with 200 KB (1024*200/2 = 102400 characters).

File Properties (property type is 'file' and GUI is not 'Signature')

Maximum allowed length for a file property value depends on the "File size limit" attribute configured for the property, but it can't exceed 20 MB (20971520 bytes) in any case.

Properties (any other property type)

Maximum amount of data to store is 65 535 bytes (2^16 - 1). Oracle Fusion Field Service internally uses the UTF-8 encoding, so the value is converted to UTF-8 representation before checking against the limit. One Unicode character (code point) may take up 1 to 4 bytes in UTF-8. But, JavaScript uses UTF-16 for strings, so one character takes up 2 to 4 bytes. The String.length property uses UTF-16 code units for counting, which is 2 bytes. So the length of the string that contains one 1 or 2-byte Unicode char is 1. The length of the string that contains one 3 or 4-byte Unicode char (code point) is 2. There's also a range of 2-byte Unicode code points (U+0800 - U+10000) that take up 2 bytes (1 code unit) in UTF-16 (e.g. ¿ - \u20AC), but require 3 bytes in UTF-8. So, only 65535/3 = 21845 code units are always under limit. If the length of the string is grater that 21845, it may or may not pass the validation depending on its contents. To know whether the property value is of valid length, it must be converted to UTF-8, for example:

```
function isPropertyLengthValid(value) {

 if (('' + value).length <= }}{{21845) {
 return true;
 }

 if (('' + value).length > }}{{65535) {
 return false;
 }

 var utf8Encoder = }}{{new TextEncoder();
 var utf8BytesArray = utf8Encoder.encode(value);

 utf8Encoder = }}{{null;

 if (utf8BytesArray.length <= }}{{65535) {
 return true;
 }

 return false;
}
```

File Properties

Plugin API supports updating of file properties. The plugin sends the values of file properties with the regular properties in the entity collections or inventory actions as part of the *close* message. Due to performance limitations, it's not rational to send the file contents using JSON strings, so the Plugin API accepts raw JS objects as the value for PostMessage data. File properties can be updated only using JS objects as message data. The value of the file property in the PostMessage data must be an object that has two properties:

- fileName: Name of the file, that will be shown on the Oracle Fusion Field Service user interface
- fileContents: Blob object that contains the file contents. It can be constructed and filled with the data generated by JS code in runtime, or just obtained from the file input and sent to Oracle Fusion Field Service Core Application without any transformation, as the File object inherits the Blob.

Contents of the file property value is validated against these rules:

- Length of the file must be less than or equal to the configured File size limit
- MIME type of the file must be equal to one of the configured Allowed MIME types

Examples of close Method with File Properties

Sending Uploaded Content

```
var file = document.querySelector('input[type=file]').files[0];
```

```
window.parent.postMessage
(
{
apiVersion: 1,
method: 'close',
activity:
{
aid:132,
ccity: 'Cleveland',
door_photo:

{ fileName: 'DCIM_20170425_203115.jpg', fileContents: file }
}
},
targetOrigin
);
```

Sending Generated Content

```
var text =
 '<?xml version="1.0" encoding="UTF-8"?>' +
 '<test></test>';

var blob = new Blob([text], { type: 'text/xml' });

window.parent.postMessage
(
 {
 apiVersion: 1,
 method: 'close',
 activity:
 {
 ccity: 'Cleveland',
 XML_DATA_PROP:
 {
 fileName: 'test_data.xml',
 fileContents: blob
 }
 }
 },
 targetOrigin
);
```

Activity Status and Inventory Pool Changes

You can change the activity status (such as start, suspend, complete activity) by simply updating the field 'astatus' taking into account the available status transitions. You can perform actions (such as install, deinstall inventory, undo install) on the inventory pool for serialized inventory by simply updating the field 'pool' taking into account the available pool transitions, and update the required fields for the pool (for example, inv_aid for install pool).

Order of Applying Changes to Entity Data Collections

If a plugin sends a few collections such as, 'activityList', 'activity', 'inventoryList', and 'inventory' in the 'close' method, the application tries to apply the changes in this order:

1. 'activityList'
2. 'activity'
3. 'inventoryList'

4. 'inventory'

If a plugin receives the same activity changes in the 'activityList' and 'activity' entity data collections, only the changes from the 'activity' entity data collection are applied. The changes from the 'activityList' entity data collection are ignored. However, the current activity in the 'activityList' can be changed, if the 'activity' entity data collection is not sent to the plugin. This example shows the activity changes that can and cannot be applied:

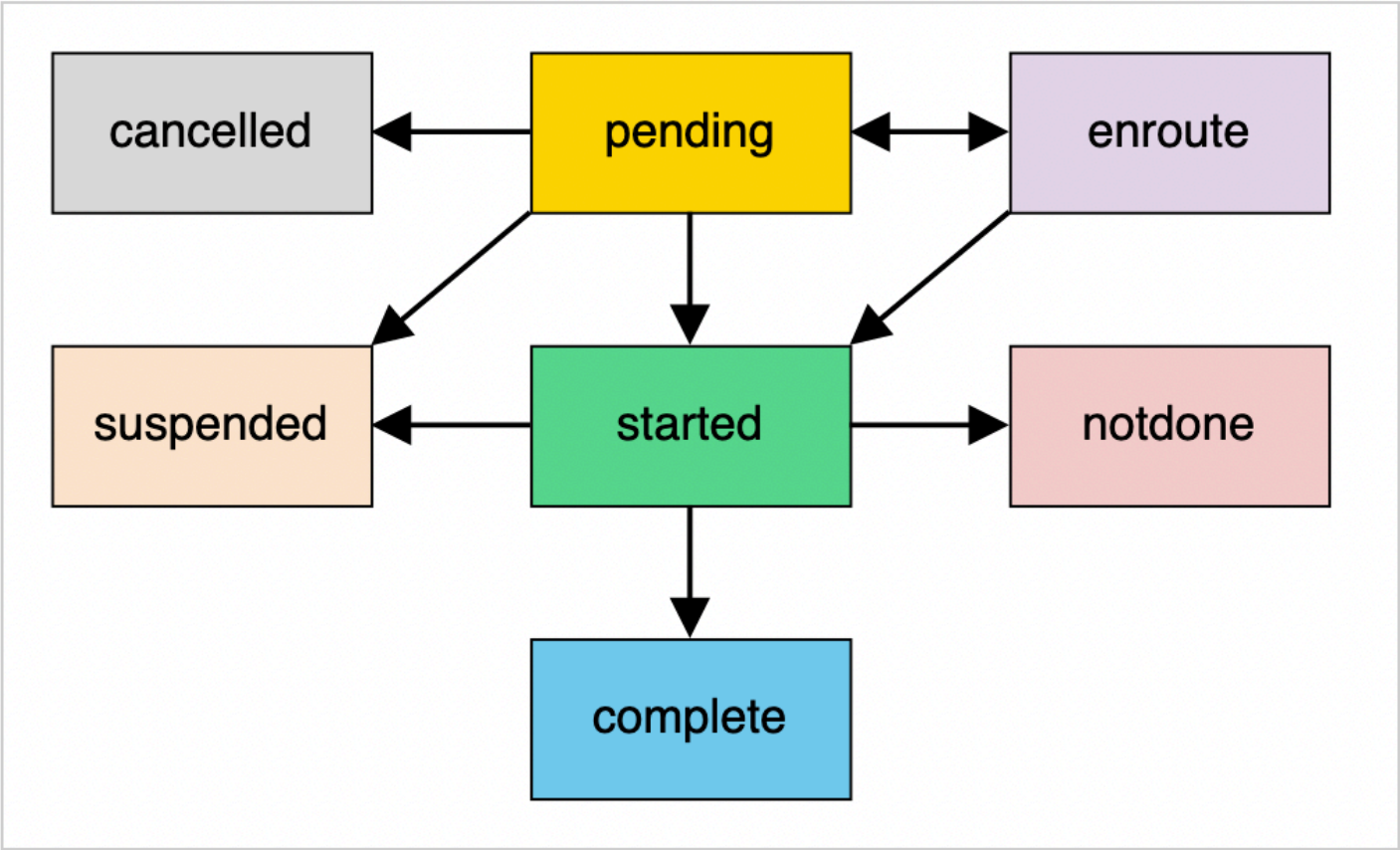
```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "default",
 "wakeupNeeded": false,
 "activity": {
 "aid": "8761055",
 "ACTIVITY_NOTES": "new changes 1" <--- these changes will be applied
 },
 "activityList": {
 "8761054": {
 "ACTIVITY_NOTES": "another activity"
 },
 "8761055": {
 "ACTIVITY_NOTES": "new changes 2" <--- these changes won't be applied, they will be ignored
 }
 }
}
```

If a plugin receives the same activity changes in the 'inventoryList' and 'inventory' entity data collections, only the changes from the 'inventory' entity data collection are applied. The changes from the 'inventoryList' entity data collection are ignored. However, the current inventory in the 'inventoryList' can be changed, if the 'inventory' entity data collection is not sent to the plugin. This example shows the inventory changes that can and cannot be applied:

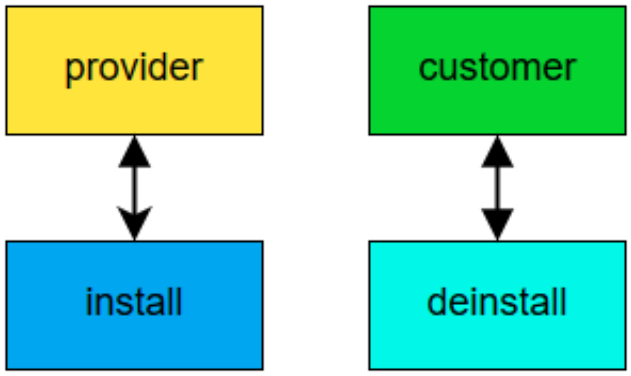
```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "default",
 "wakeupNeeded": false,
 "inventory": {
 "invid": "1055",
 "INVENTORY_NOTES": "new changes 1" <--- these changes will be applied
 },
 "inventoryList": {
 "1054": {
 "INVENTORY_NOTES": "another inventory"
 },
 "1055": {
 "INVENTORY_NOTES": "new changes 2" <--- these changes won't be applied, they will be ignored
 }
 }
}
```

Changing of Activity status

Activity status can be changed (e.g. start, suspend, complete activity) by simple update of field '*astatus*' regarding the available status transitions. This image shows the possible transitions between activity statuses:



This image shows the possible transitions between inventory pools:



Inventory Glossary

This table provides the meaning of commonly used inventory terms:

Term	Description
Inventory	The term inventory describes equipment that is used – or in the language of inventory – consumed by activities. Inventory items can be located at the customer's home or business or carried in a resource's truck. Modems, faceplates, wire, cable, and electrical tape are all examples of inventory. Inventory includes both serialized and non-serialized items. Serialized inventory consists of individual pieces with serial numbers that identify both the type of inventory and the manufacturer/distributor. Non-serialized inventory, such as faceplates, wire, and electrical type don't have serial numbers. This type of inventory is generic. One manufacturer's supply can be exchanged for another based on a model number. Non-serialized inventory is often accounted for in bulk by units of measure, such as feet, pounds, dozen, and so on. These items are usually carried in the resource's truck, although the amounts required for individual activities are recorded with serialized inventory on the Activity Details page and on the Inventory List in the resource's Oracle Fusion Field Service.
Inventory type	Inventory type is used to identify the business logic linked with inventory of such type: • whether inventory of the type is serialized or non-serialized • whether inventory of the type can use additional model property • if inventory of the type is non-serialized then what Unit of measure is used to count them You can change the inventory type after creating inventory. Further, you can change one inventory type to another any time.
Default inventory type	For compatibility with the companies that don't need inventory type when a new inventory is created and if there are no "Inventory type" configured in the application, then: • New inventory is created without setting inventory type. (The id of the type is set to 0). • Inventory is serialized and its quantity is forced to be 1. • Model is empty. • Such inventory cannot be listed in required inventories, because there is no defined inventory type.

Non-Serialized Inventory Update

You cannot install or deinstall non-serialized inventory through a simple update of properties in the 'inventory' or 'inventoryList' entity data collection. This is because, updating non-serialized inventory requires creating new inventory in the target inventory pool. You can process non-serialized inventory only using Inventory Actions.

You can add decimal values for the "quantity" field of inventory in the "inventory" and "inventoryList" collections and for the "quantity" param of all of inventory actions (create, delete, install, deinstall, undo_install, and undo_deinstall).

Oracle Fusion Field Service rounds off the value before applying the update or action. Therefore, it won't have more digits after the decimal point than what is configured for the inventory type (the "Quantity precision" option). The value for "quantity" must satisfy these requrements:

- Is a Number or a String for which the parseFloat() function returns the Number (not NaN)
- Is less than 999999999.99999
- Is greater that -999999999.99999 for "inventory" and "inventoryList" collections
- Is greater that 0 for inventory actions

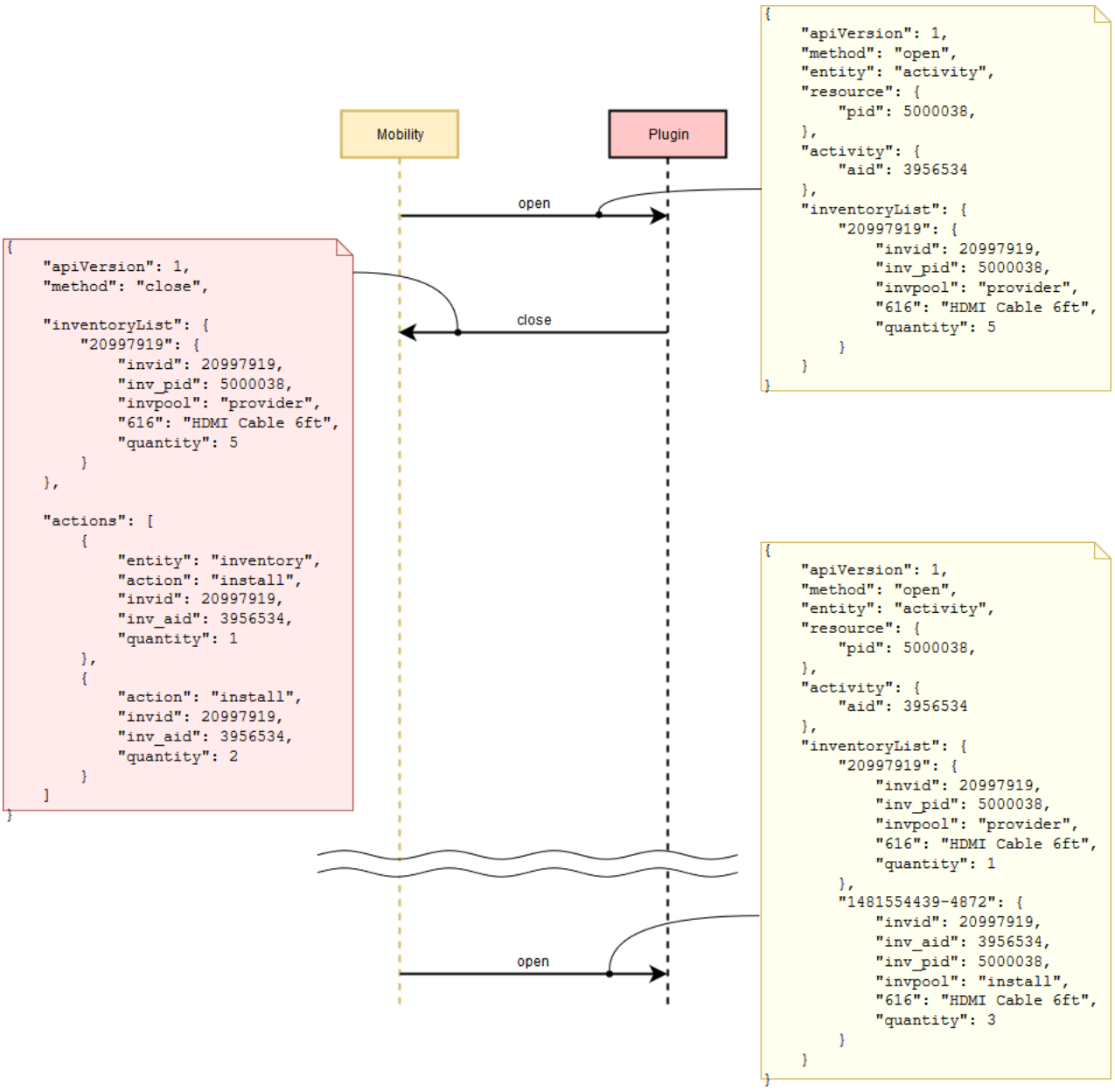
If these requirements are not satisfied, these error messages are shown:

- For "inventory" and "inventoryList" collections, the error type is TYPE_ENTITY_PROPERTY, the code is CODE_PROPERTY_VALUE_INVALID
- For inventory actions the error type is TYPE_ACTION_PARAM and the code is CODE_ACTION_PARAM_VALUE_INVALID

Inventory Actions

To support complex business flows that require the creation of activities, or deal with non-serialized inventory types, the optional field "actions" is supported for the *close* method. This field contains a list of objects, whose fields represent the parameters of actions (such as "install", "create") that are applied by Oracle Fusion Field Service on both the client and server side.

Example of a Message Sequence This diagram shows an example of a message sequence:



Order of Processing of Actions

Actions are applied in the same order as sent by the plugin in the "actions" array.

Actions are run after applying all the data collection updates that are sent by plugin in the same *close* message. No actions are applied if errors occur during the validation of data collections and all actions. All validation errors are sent to a plugin within the *error* message. If some actions fail to run, the remaining actions are applied anyway, and the

processing errors are sent to the plugin within the *error* message. Each error contains the id of action that is failed, or the id of the action that doesn't pass validation. Id is the order number of the action in the actions list, sent by the plugin.

Here is how Oracle Fusion Field Service processes the *close* message:

1. Validate entity data collections.
2. Validate actions.
3. If there are any validation errors, send the *error* message to the plugin; if not, proceed to the next step.
4. Apply entity data collections update.
5. Apply actions.
6. If there are any errors during the update of data collections or running of actions, send the *error* message to the plugin; if not, proceed to the next step.
7. Close the plugin window.

Inventory Action Parameters

Each action is an object, whose fields are the action parameters. Every action must contain at least two fields (parameters) - ‘entity’ and ‘action’.

This table describes the ‘entity’ and ‘action’ parameters for inventory:

Parameter	Mandatory	Type	Description
entity	Yes	String	Must be either "activity" or "inventory"
action	Yes	String	Must be equal to one of the supported inventory actions or supported activity actions (such as "install", "create")

Parameters that are specific for each action are described in the Supported Inventory Actions and Supported Activity Actions sections. Parameters that contain the ids of Oracle Fusion Field Service entities (invid, inv_aid, inv_pid) are of the type "string" and not "number". This is because, entities that are created on the client side have ids similar to "1234567890-1234", before they're synchronized with the server.

Labels and values of all parameters are case-sensitive, for example, all these parameters are invalid:

```
{
 ACTION: "INSTALL"
 entity: "Inventory",
 Inv_Aid: ""
}
```

Supported Inventory Actions

The Plug-in API supports install, deinstall, undo install, undo deinstall, create, and delete actions for inventory.

install

This table describes the parameters supported for the install inventory action:

Parameter	Mandatory	Type	Description
invid	Yes	String	Id of an existing inventory that is in the "provider" pool of the current resource or the resource's teammates.
inv_aid	Yes	String	Id of the started activity. Inventory will be installed to its "install" pool. Must contain the id of started segment for segmentable activities.
quantity	Yes/No	Number	Is mandatory and must be > 0 for non-serialized inventory. Is forbidden for serialized inventory.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Fusion Field Service inventory properties to be updated. Properties are validated and processed according to the plugin configuration.

deinstall

This table describes the parameters supported for the deinstall inventory action:

Parameter	Mandatory	Type	Description
invid	Yes	String	Id of an existing inventory that is in the "customer" pool of a started activity.
inv_aid	Yes	String	Id of the current resource or the resource's teammates. Inventory will be deinstalled to its "deinstall" pool.
quantity	Yes/No	Number	Is mandatory and must be > 0 for non-serialized inventory. Is forbidden for serialized inventory.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Fusion Field Service inventory properties to be updated. Properties are validated and processed according to the plugin configuration.

undo_install

This table describes the parameters supported for the undo-install inventory action:

Parameter	Mandatory	Type	Description
invid	Yes	String	Id of an existing inventory that is in the "install" pool of a started activity.
quantity	Yes/No	Number	Is mandatory and must be > 0 for non-serialized inventory.

Parameter	Mandatory	Type	Description
			Is forbidden for serialized inventory.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Fusion Field Service inventory properties to be updated. Properties are validated and processed according to the plugin configuration.

undo_deinstall

This table describes the parameters supported for the undo-deinstall inventory action:

Parameter	Mandatory	Type	Description
invid	Yes	String	Id of an existing inventory that is in the "deinstall" pool of the current resource or the resource's teammates.
quantity	Yes/No	Number	Is mandatory and must be > 0 for non-serialized inventory. Is forbidden for serialized inventory.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Fusion Field Service inventory properties to be updated. Properties are validated and processed according to the plugin configuration.

create

This table describes the parameters supported for the create inventory action:

Parameter	Mandatory	Type	Description
invtype	Yes	String	Label of one of the Inventory Types, configured for Oracle Fusion Field Service (for example, "NT")
invpool	Yes	String	Inventory pool in which the inventory is created. It can be one of: "customer", "install", "deinstall", "provider".
inv_aid	Yes/No	String	Id of the started activity. Inventory will be created in its pool. Must contain the id of the started segment for segmentable activities. Is mandatory if invpool is one of: "customer", "install", or "deinstall". Is forbidden for invpool equal to "provider".

Parameter	Mandatory	Type	Description
inv_pid	Yes/No	String	Id of the current resource or the resource's teammates. Inventory will be created in the resource's pool. Is mandatory if invpool is one of: "provider", "install", "deinstall". Is forbidden for invpool equal to "customer".
quantity	Yes/No	Number	Is mandatory and must be > 0 for non-serialized inventory. Is forbidden for serialized inventory. **Note:** If the quantity is not present in the **Add Plugin** or **Modify Plugin** page, **Available properties** section or present and set to "Read-only" and if it's not configured as available for the plugin, then it is set to "1" for non-serialized inventory by Oracle Fusion Field Service Core Application itself.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Fusion Field Service inventory properties to be updated. Properties are validated and processed according to the plugin configuration.

delete

This table describes the parameters supported for the delete inventory action:

Parameter	Mandatory	Type	Description
invid	Yes	String	Id of an existing inventory that is in the "provider" pool of the current resource or the resource's teammates, or in "install", "deinstall" or "customer" pool of the started activity. **Note:** There is no quantity parameter for the delete action. The entire record with any quantity is deleted from the corresponding pool.

Example of the close Message

```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "default",
 "actions":
 [
 // INSTALL
 {
 "entity": "inventory",
```

```
"action": "install",
"invid": 21258426,
"inv_aid": 4224031,
"properties": // Properties can be updated too
{
"PORT_INFO": "A0|1|1|0|7.9|QF9-0719537",
"EQUIPMENT_ETHERNET": "08:00:27:ea:d1:bd"
}
},
{
"entity": "inventory",
"action": "install",
"invid": 21229417,
"inv_aid": 4224031,
"quantity": 12, // Install only 12 pieces of NSI

"properties": // Model should be set if needed
{
"inventory_model": "RG6 - BLK"
}
},

// DEINSTALL
{
"entity": "inventory",
"action": "deinstall",
"invid": 21064418,
"inv_pid": 3000001
},

// CREATE
{
"entity": "inventory",
"action": "create",
"invtype": "NT",
"invpool": "installed",
"inv_aid": 4224031,
"inv_pid": 3000001,
"quantity": 100,

"properties":
{
"inventory_model": "RG6 - BLK"
}
},

// DELETE
{
"entity": "inventory",
"action": "delete",
"invid": "1484311067004-9891"
}
]
}
```

Error Types for Inventory Actions

You can encounter validation, run, or internal type of errors when your plugin performs an action.

Validation Errors

If the *error* message contains only validation type of errors, it means that no actions are applied and no entity collections are updated. So, the *close* message can be sent again with the same actions, corrected to eliminate the errors. This table describes the validation errors:

Type	Occurs When	Available Message Fields
TYPE_ACTION_ERROR	Actions have an invalid format or are inapplicable	• actionId • entity • entityId
TYPE_ACTION_PARAM	Action param has an invalid value or mandatory param is missing	• actionId • entity • entityId • paramName
TYPE_ACTION_PROPERTY	Value of a property submitted by the plugin to be updated in the "properties" param, has an invalid value	• actionId • entity • entityId • propertyLabel

Run Errors

Run errors are errors that are generated when actions are applied. As actions are applied one after the other, some may fail and some may be applied successfully. The *error* message contains run errors, for failed actions. This means that the entity collections may be updated after some actions are applied, and it didn't produce any run errors (or there are no errors, which contain their index). So, actions which don't fail, must not be sent again. This table describes the run errors:

Type	Occurs When	Available Message Fields
TYPE_ACTION_FAILED	Action is rejected due to incorrect value of action params, which can't be checked at the validation stage	• actionId • entity • entityId

Internal Errors

The error message contains errors of these types, if the validation has passed successfully, but entity collection update, or some actions have failed. This means that the entity collection update may not be applied, so as actions which didn't cause errors . In this case, the entities are in a state that is unknown to the plugin, so the *close* message containing any updates, must not be sent. This table describes the internal errors:

Type	Occurs When	Available Message Fields
TYPE_INTERNAL	Oracle Fusion Field Service is unable to process the message due to an unexpected change of the system's state. Doesn't contain information about the action, which caused it.	NA

Type	Occurs When	Available Message Fields
	Both actions and entity collection updates may lead to such errors.	

Available Message Fields

- entity: Entity type on which the action is performed ("inventory")
- entityId: Id of the entity, which is updated by action (equals invid for inventory, for example, "10028719")
- actionId: Zero-based order number of erroneous action in the actions list, sent by the plugin. For example, 0, 1, 17.
- paramName: Name of the action parameter, whose value is invalid, for example, "entity", "inv_aid".
- propertyLabel: Label of the property, whose value is invalid, for example, "customer_number", "WO_TYPE".

Error Codes for Inventory Actions

This table describes the errors that are available for inventory-related actions:

Code	Caused by Action	Cause
TYPE_ACTION_ERROR		
CODE_ACTION_NUMBER_LIMIT_EXCEEDED	create	Number of items in the "actions" field of *close* or *update* message is greater than 10,000.
CODE_ACTION_ON_PAST_DATE_NOT_ALLOWED	• install • deinstall • undo_install • undo_deinstall • create • delete	Any of these: • "inv_aid" param of "install", "deisntall", "undo_install" or "undo_deinstall" action is equal to id of activity that is assigned for a past date • "inv_aid" param of "create" or "delete" action is equal to id of the activity that is assigned for a past date, and "invpool" is "customer", "install" or "deinstall"
CODE_ENTITY_ID_INVALID	• install • deinstall • undo_install • undo_deinstall • create • delete	"invid" param is not equal to the id of any inventory in available pools
CODE_ACTION_UNKNOWN	—	"action" param is not equal to any of the supported inventory actions (for example, "install", "create")
CODE_ACTION_ENTITY_UNKNOWN	—	"entity" param is not equal to "inventory"
TYPE_ACTION_PARAM		

Code	Caused by Action	Cause
CODE_ACTION_INVENTORY_AID_INVALID	create	"inv_aid" param is sent for the "create" action, and "invpool" is "provider"
CODE_ACTION_INVENTORY_PID_INVALID	• deinstall • create	Any of these: • "inv_pid" param value is not equal to id of current resource or his teammates • "inv_pid" param is sent for "create" action, and "invpool" is "customer"
CODE_ACTION_INVENTORY_POOL_INVALID	create	"invpool" param value is not equal to one of: "customer", "install", "deinstall", "provider"
CODE_ACTION_INVENTORY_TYPE_INVALID	create	"invtype" param value is not equal to the label of any of the Inventory Types, configured for Oracle Fusion Field Service
CODE_ACTION_MANDATORY_PARAM_EMPTY	• install • deinstall • undo_install • undo_deinstall • create • delete	Any of these: • "invid" param is not sent or its value is empty for "install", "deinstall", "undo_install", "undo_deinstall" or "delete" action • "invpool" param of "create" action is not sent or is empty • "inv_aid" param of "install" action is not sent or is empty • "inv_pid" param of "deinstall" action is not sent or is empty • "inv_aid" param of "create" action is not sent or is empty, and "invpool" is "customer", "install" or "deinstall" • "inv_pid" param of "create" action is not sent or is empty, and "invpool" is "provider", "install" or "deinstall" • "quantity" is not sent or is empty for inventory of non-serialized type
CODE_ACTION_PARAM_VALUE_INVALID	• install • deinstall • undo_install • undo_deinstall • create	Any of these: • "properties" param value is sent but is not a plain object • "quantity" is sent for inventory of serialized type • "quantity" is not a positive integer number
TYPE_ACTION_PROPERTY		
CODE_ACTION_MANDATORY_PROPERTY_EMPTY	• install • deinstall • undo_install • undo_deinstall • create	[Reserved]
CODE_ACTION_PROPERTY_VALUE_INVALID	• install • deinstall • undo_install • undo_deinstall	Any of these: • Property type is 'file', its GUI type is 'signature' and its value is not a valid Data URI, or it has an invalid MIME-type • Property type is 'enumeration', and its value is not a valid enumeration item's index

Code	Caused by Action	Cause
	• create	
CODE_ACTION_PROPERTY_VALUE_TOO_LARGE	• install • deinstall • undo_install • undo_deinstall • create	Any of these: • Property type is 'field' and the length of its value exceeds 119 UTF-16 codepoints • Property type is 'file', its GUI type is 'signature' and the length of its value exceeds 102400 UTF-16 codepoints • Property is neither field nor signature and the length of its value exceeds 32767 UTF-16 codepoints See Property Value Length and Limits for details.
TYPE_ACTION_FAILED		
CODE_ACTION_INVENTORY_ACTIVITY_STATUS_INVALID	• install • deinstall • undo_install • undo_deinstall • create • delete	Any of these: • "inv_aid" param of "install", "deinstall", "undo_install" or "undo_deinstall" action is not equal to the id of a started activity • "inv_aid" param of "create" or "delete" action is not equal to the id of a started activity, and "invpool" is "install" or "deinstall" • "inv_aid" param of "create" or "delete" action is equal to the id of a completed, not done, or cancelled activity, and "invpool" is "customer" • "invid" param of "deisntall", "undo_install" or "undo_deinstall" action is equal to the id of an inventory, associated with a not started regular activity • "invid" param of "deisntall", "undo_install" or "undo_deinstall" action is equal to the id of an inventory, associated with a segmentable activity that is not a master activity of the currently started segment
CODE_ACTION_INVENTORY_ACTIVITY_TYPE_INVALID	• install • create	"inv_aid" param equal to the id of an activity, whose type doesn't support inventories
CODE_ACTION_INVENTORY_ACTIVITY_UNKNOWN	• install • create	"inv_aid" param isn't equal to: • Id of one of the activities in the queue of current provider / his teammates • Id of one of the activities in the unscheduled pool
CODE_ACTION_INVENTORY_POOL_TRANSITION_INVALID	• install • deinstall	Any of these: • "invid" param of "install" action is equal to the id of inventory, whose "invpool" isn't equal to "provider" • "invid" param of "deinstall" action is equal to the id of inventory, whose "invpool" isn't equal to "customer" • "invid" param of "undo_install" action is equal to the id of inventory, whose "invpool" isn't equal to "install" • "invid" param of "undo_deinstall" action is equal to the id of inventory, whose "invpool" isn't equal to "deinstall"
TYPE_INTERNAL		

Code	Caused by Action	Cause
CODE_UNKNOWN	• install • deinstall • undo_install • undo_deinstall • create	Oracle Fusion Field Service is unable to process the message due to an unexpected change of the system's state

Related Topics

- Which inventory actions does the plugin API support?

Supported Activity Actions

You can create scheduled and non-scheduled activities through custom plugins. For example, a custom plugin to check the expected delivery of a part can create a follow-up activity for a repair job using the part. You can achieve this using the activity action ‘create’, which is available for the ‘close’ method.

Note: The plugin can create activities only on the route of the currently selected resource.

create Method Parameters

This table provides the parameters of the *create* method.

Parameter Name	Mandatory	Type	Description
temporaryAid	No	String	Temporary ID of a created activity. If the param is not set, the temporary aid is generated by Oracle Fusion Field Service. The value must be unique, that is, must not duplicate a temporary aid of any existing activity. The value must be a string that matches the regular expression: /^\d+\-\d{4}p?$/ . The length of the string must be less than or equal to 25.
activityType	Yes	String	Label of one of the Activity Types, configured for Oracle Field Service (for example, "LU"). Activity types with enabled features "Allow mass activities", "Allow repeating activities" or "Enable segmenting and extended duration" are not allowed.
date	No	String	Date in YYYY-MM-DD format defined by ISO 8601 (for example, "2019-11-28") If the date is not set, or is empty (null or empty string), the activity is created for the date of the currently selected route. Is forbidden for scheduled equal to false.

Parameter Name	Mandatory	Type	Description
scheduled	No	Boolen	If scheduled is true, the activity is scheduled for date defined by value of date parameter. If scheduled is false, the activity is created as not scheduled. Non-scheduled activities can be created only for Activity types that have the "Allow non-scheduled" feature enabled. Default value: true
positionInRoute	No	Object	The value of this parameter determines the position of the activity in the route. It's the object with two fields: • position (string) - one of these values: "first", "last", "notOrdered", "afterActivity" • activityId (string) - the unique identifier of the activity in Oracle Fusion Field Service. If the value of the field 'position' is 'afterActivity', then created activity is scheduled right after activity with given activityId The "afterActivity" value of position field is not allowed, if the date parameter is not empty and is not equal to the date of the currently selected route. Not-ordered activities can be created only for Activity types that have the "Support of not-ordered activities" feature enabled. Default value: { position: "last" }
duration	No	Number	Duration of activity in minutes (integer number). Minimal value: 0, maximal value: 65535. Duration can be set only for Activity types that have the "Calculate activity duration using statistics" feature disabled.
timeSlot	No	String	The label of one of Time Slots configured in Oracle Fusion Field Service. The time slot defines the service window for the activity. The timeSlot parameter can be set only for Activity types that have the "Support of time slots" feature enabled. If set, the value of timeSlot must contain the label of one of the timeslots that are selected in the "Available time slots" section of Activity Type configuration.
serviceWindowStart	No	String	The time when the service window starts for the activity in hh:mm format defined by ISO 8601 (for example, "14:07") The serviceWindowStart parameter can be set only for Activity types that have the "Support of time slots" feature disabled.
serviceWindowEnd	No	String	The time when the service window ends for the activity in hh:mm format defined by ISO 8601 (e.g. "09:56") The serviceWindowStart parameter can be set only for Activity types that have the "Support of time slots" feature disabled.
slaWindowStart	No	String	The time when the service level agreement (SLA) window starts. The date and time must be in the format "YYYY-MM-DD hh:mm" or "YYYY-MM-DD hh:mm:ss" (for example, "2019-12-16 16:42").
slaWindowEnd	No	String	The time when the service level agreement (SLA) window ends. The date and time must be in the format "YYYY-MM-DD hh:mm" or "YYYY-MM-DD hh:mm:ss" (for example, "2019-12-20 16:42"). If slaWindowStart is specified then slaWindowEnd must be equal or greater than slaWindowStart (slaWindowEnd ≥ slaWindowStart).

Parameter Name	Mandatory	Type	Description
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Field Service activity properties to be written. Properties are validated and processed according to the way the plugin is configured.

Error Codes for Activity Actions

This table describes the error codes that are displayed for activity actions, and the reason for which they are displayed.

Code	Caused by Action	Cause
CODE_ACTION_PARAM_NOT_UNIQUE	create	Value of "temporary_aid" param of "create" activity action duplicates the value of "temporary_aid" field of another activity in the route.
TYPE_ACTION_ERROR		
CODE_ACTION_NUMBER_LIMIT_EXCEEDED	create	Number of items in the "actions" field of *close* or *update* message is greater than 10,000.
CODE_ACTION_ON_PAST_DATE_NOT_ALLOWED	create	Any of these: • "date" parameter of "create" action is set and the currently selected queue is in the past (and overnight has ended). • "date" parameter of the "create" action equals the date of the currently selected queue and the selected queue is in the past (and overnight has ended). • "date" parameter of the "create" action contains the date, which is earlier than the local date (that is, the date in the Resource's time zone) of currently the selected Resource.
CODE_ACTION_UNKNOWN	N/A	"action" parameter is not equal to one of the "create"
CODE_ACTION_ENTITY_UNKNOWN	N/A	"entity" parameter is not equal to "activity" or "inventory"
TYPE_ACTION_PARAM		
CODE_ACTION_ACTIVITY_TYPE_INVALID	create	Any of these: • "activityType" parameter value is not equal to the label of one of the Activity Types, configured for Oracle Fusion Field Service • "activityType" parameter contains the label of Activity type, for which any of these features is enabled: o Allow mass activities o Allow repeating activities o Enable segmenting and extended duration

Code	Caused by Action	Cause
CODE_ACTION_DATE_FORBIDDEN	create	The "scheduled" parameter is false and "date" parameter is set.
CODE_ACTION_NOT_SCHEDULED_FORBIDDEN	create	The "scheduled" parameter is false and the "activityType" parameter contains the label of the Activity type for which the feature "Allow non-scheduled" is disabled.
CODE_ACTION_TIME_SLOT_FORBIDDEN	create	The "timeSlot" parameter is set and the "activityType" parameter contains the label of Activity type for which the feature "Support of time slots" is disabled.
CODE_ACTION_TIME_SLOT_NOT_AVAILABLE	create	The "timeSlot" parameter contains the label of the time slot, which is not selected in the "Available time slots" section of Activity Type configuration.
CODE_ACTION_SW_FORBIDDEN	create	"serviceWindowStart" or "serviceWindowEnd" parameter is set and "activityType" parameter contains the label of the Activity type for which the feature "Support of time slots" is enabled.
CODE_ACTION_MANDATORY_PARAM_EMPTY	create	The "activityType" parameter is not set.
CODE_ACTION_PARAM_VALUE_INVALID	create	Any of these: • "date" parameter is set and its value is not a valid date string in YYYY-MM-DD format defined by ISO 8601 • "scheduled" parameter is set and its value is not a boolean • "duration" parameter is not an integer or out of range • "timeSlot" parameter is set and its value is not equal to the label of one of the Time Slots, configured for OFS • "timeSlot" parameter is set and its value is not a string • "serviceWindowStart" parameter is set and its value is not a valid time sting in hh:mm format defined by ISO 8601 • "serviceWindowEnd" parameter is set and its value is not a valid time string in hh:mm format defined by ISO 8601 • "positionInRoute" is set and is not an object • "position" field's value of "positionInRoute" parameter doesn't equal one of these: "first", "last", "notOrdered", "afterActivity" • "position" field of "positionInRoute" is "afterActivity" and "scheduled" parameter is false • "position" field of "positionInRoute" is "afterActivity" and "date" parameter is set and it is not equal to the date of the currently selected queue

Code	Caused by Action	Cause
		• The "position" field of "positionInRoute" is "afterActivity" and the "activityId" field of the "positionInRoute" parameter is not set, or it is not equal to the Activity ID of one of the activities in the currently-selected queue.
TYPE_ACTION_PROPERTY		
CODE_ACTION_PROPERTY_VALUE_INVALID	create	Any of these: • Property type is 'file', its GUI type is 'signature' and its value is not a valid Data URI or it has the invalid MIME-type. • Property type is 'enumeration', and its value is not a valid enumeration item's index.
CODE_ACTION_PROPERTY_VALUE_TOO_LARGE	create	Any of these: • Property type is 'field' and length of its value exceeds 119 UTF-16 code points. • Property type is 'file', its GUI type is 'signature' and length of its value exceeds 102400 UTF-16 code points. • Property is neither field nor signature and length of its value exceeds 32767 UTF-16 code points. See Property value's length limits for details.
TYPE_INTERNAL		
CODE_UNKNOWN	create	Oracle Fusion Field Service is unable to process the message due to an unexpected change in the system's state.

Example of *close* Method to Create an Activity

```
{
 "apiVersion": 1,
 "method": "close",
 "actions": [
 {
 "entity": "activity",
 "action": "create",
 "activityType": "SDI",
 "scheduled": true,
 "date": "2019-12-12",
 "timeSlot": "",
 "positionInRoute": {
 "position": "afterActivity",
 "activityId": "4225450"
 },
 "serviceWindowStart": "10:00",
 "serviceWindowEnd": "11:30",
 "properties": {
 "WO_COMMENTS": "Follow-up activity"
 }
 }
 ]
```

```
}
```

Temporary ID

Each time a new instance of an Oracle Fusion Field Service entity (Activity or Inventory) is created on a user's device, a temporary ID is generated. This happens regardless of whether the instance is created through the plugin API or through a standard page such as Add activity or one of the Inventory pages. The temporary ID used as a value of the Activity ID (aid) and Inventory ID (invid) fields until data is synchronized with the server, so the user and the plugins may operate the newly created entities even offline. After synchronization, the temporary ID is replaced with an actual aid or invid generated by the server.

To allow the plugin perform any operation on activities created on the client-side (using the "create" action of the plugin API or the standard Add activity page), both in offline in online, the plugin API provides the following:

- The value of the client-generated temporary id is stored in the "temporary_aid" field of the activity, which persists even after data is synced with server and a permanent aid (generated by a server) is obtained.
- Temporary aid may be used as a valid value of the "inv_aid" param for all inventory actions, "inv_aid" field for "inventoryList" and "inventory" collection updates, and as an activity key for the "activityList" collection updates even after synchronization with server.
- The plugin may generate a temporary id for activities on its own and send it in the "temporaryAid" param of the "create" activity action. The plugin may store this value and use it to reference a created activity in later acctions not even knowing the actual aid.
- The plugin can use the value of the generated temporaryAid as a value of the inv_aid param for inventory actions even in the same *close* or *update* message as the *create* activity action itself.

The temporary_aid field is not available for segmentable, mass, and repeating activities as the number and actual IDs of segments or instances may be changed at any time.

Example of actions that make use of temporaryAid:

```
{
 "apiVersion": 1,
 "method": "update",
 "activityList": {
 "16297679485790-0793": {
 "astatus": "cancelled"
 }
 }
 "actions": [
 {
 "entity": "activity",
 "action": "create",
 "temporaryAid": "16297673252100-2175p",
 "activityType": "4",
 "scheduled": true,
 "date": "2021-08-23"
 },
 {
 "entity": "inventory",
 "action": "create",
 "inv_aid": "16297673252100-2175p",
 "invtype": "NT",
 "invpool": "customer",
 "properties": {
 "invsn": "KCD1403WH"
 }
 }
 ]
```

```
}
```

Background Activity

See the wakeup method topic.

Example of *close* message with the *wakeupNeeded* param

```
{
 "apiVersion": 1,
 "method": "close",
 "activity":
 {
 cname: "John"
 },
 "wakeupNeeded": true
}
```

Related Topics

- wakeup Message

Redirection with the close Method

After the *close* method is applied, Oracle Fusion Field Service Application closes the plugin and redirects the user to another page. By default, the user is redirected to the same page from which the plugin was opened. However, you can specify which page the user must be redirected to, by setting the value of the optional field, "backScreen" in the *close* message.

This table provides the possible values for the backScreen field:

Page Name	Required Parameters	Optional Parameters	Description
activity_by_id	backActivityId	None	Details page for the activity with the given ID. If there's no activity with the given ID in a queue, the user is redirected to the previous page. ID of the activity must be sent in the field "backActivityId". If backActivityId contains ID of non-scheduled activity, user will be redirected to the details page of that activity only if it's visible on Activity list screen. If activity is not visible because of applied filter or because of "Mobile Activity Count" limitation, user will be redirected to the previous page.
next_activity	None	None	Details of the next pending activity by ETA, or the first pending activity, if there are no pending activities after the current activity. If there are no pending activities in a queue, user is redirected to the Activity List page.

Page Name	Required Parameters	Optional Parameters	Description
activity_list	None	None	Activity List page
start_activity	backActivityId	None	Start Activity page
end_activity	backActivityId	None	Complete Activity page
cancel_activity	backActivityId	None	Cancel Activity page
notdone_activity	backActivityId	None	Not done Activity page
suspend_activity	backActivityId	None	Suspend Activity page
enroute_activity	backActivityId	None	En route page for the activity with the given ID. If there's no activity with the given ID in a queue or the activity is not Pending, the user is redirected to the previous page. ID of the activity must be sent in the *backActivityId* field.
stop_travel	backActivityId	None	Stop travel page for the activity with the given ID. If there's no activity with the given ID in a queue or the activity is not in the Enroute status, the user is redirected to the previous page. ID of the activity must be sent in the *backActivityId* field.
delay_activity	backActivityId	None	Adjust Time page
inventory_list	None	backActivityId	List of inventories. If no additional params sent, Inventory List is shown as if it was opened from Activity List. If "backActivityId" contains a valid id of an activity, which is in the current queue, Inventory List is shown as if it was opened from the Activity Details page of the given activity.
inventory_by_id	backInventoryId	backActivityId	Inventory Details page for an inventory with id equal to the value of the backInventoryId field. If no additional params are sent, the page is shown as if it was opened from Activity List > Inventories. If "backActivityId" contains a valid id of an activity, which is in the current queue, Inventory List is shown as if it was opened from the Activity Details page of the given activity.
install_inventory	backInventoryId backActivityId	None	"Install" page for inventory with the id equal to the value of the backInventoryId field. After confirmation, inventory is installed to an Activity with an id which equals to the value of the backActivityId field.
deinstall_inventory	backInventoryId backActivityId	None	"Deinstall" page for inventory with the id equal to the value of the backInventoryId field. After confirmation, inventory is installed to an Activity with the id equal to the value of the backActivityId field.
plugin_by_label	backPluginLabel	backPluginOpenParams	The plugin with a label equal to the value of the "backPluginLabel" field. See "Navigation to another plugin" for details.

If the backScreen or other required parameters are inappropriate, or aren't set, the user is redirected to the previous page. Redirection to the 'reopen_activity' page is not available.

Examples of close Message with Redirection

```
// start_activity
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "start_activity",
 "backActivityId": "4225473"
}

// end_activity
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "end_activity",
 "backActivityId": "4225473"
}

// cancel_activity
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "cancel_activity",
 "backActivityId": "4225473"
}

// notdone_cancel
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "notdone_activity",
 "backActivityId": "4225473"
}

// suspend_activity
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "suspend_activity",
 "backActivityId": "4225473"
}

// delay_activity
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "delay_activity",
 "backActivityId": "4225473"
}


// inventory_list
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "inventory_list"
 "backActivityId": "4225473",
}


// inventory_by_id
```

```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "inventory_by_id",
 "backInventoryId": "3547689"
}

// install_inventory
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "install_inventory",
 "backActivityId": "4225473",
 "backInventoryId": "3547689"
}

// deinstall_inventory
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "deinstall_inventory",
 "backActivityId": "4225473",
 "backInventoryId": "3547689"
}

// plugin_by_label
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "plugin_by_label",
 "backPluginLabel": "plugin_b"
}
```

Navigation with the close Method

After a plugin page is closed, you can open the page of another or even of the same plugin. This allows you to implement complex business flows using several different plugins, or update Oracle Fusion Field Service entities without exiting the plugin's page. From the current plugin, you can redirect only to those plugins that use the Plugin API. The value of the **Disable Plugin in offline** option is not taken into account, so the plugin must handle the offline mode properly.

Additional Parameters for Plug-in on Redirection

List of all buttons that are configured for a plugin is sent to the plugin in the 'buttons' field of the 'init' message. This field is a list of objects that contains the 'buttonId' and 'params' fields. buttonId is the 'context layout item id' of the button. 'params' is an object that represents the parameters that are configured for the corresponding context layout item.

You can send additional parameters on redirection to a plugin using the *backScreen* param of the *close* message. With this, you can open the plugin in various states or show different pages of the plugin depending on the context. You can also use redirection to implement strict business flows using several plugins. Here are some examples:

- The plugin navigates to one of the many different plugins according to some business logic.
- The plugin shows some data only if it receives correct parameters from another plugin. It does not show the data if it's opened directly from a button, to force the user to follow business process.
- The plugin implements some business logic that is based on the data from another plugin, without the need to store this data in the entities' properties or the browser's local storage.

Navigation flow from plugin A to plugin B when sending parameters

The navigation flow is as follows:

- Plug-in A sends the close message with the backpluginLabel and backpluginOpenParams fields. backpluginLabel equals to the label of plugin B. backpluginOpenParams is an object, which contains the data needed by plugin B.
- After successful processing of the close message, plugin A is closed.
- Plug-in B is opened immediately and is shown on the page after the ready message is received.
- Plug-in B receives the open message, which contains the openParams field. The value of this field equals to the value of backpluginOpenParams sent by plugin A.

Requirements for the "backPluginOpenParams" Field

The requirements are as follows:

- backPluginOpenParams is a plain object.
- The maximum number of the object's fields is 20.
- Each field of the object has a scalar value (string, number, bool, null, undefined). Nested objects are forbidden.
- The size is limited to 5 KB and includes JSON structure overhead. That is, the limit is applied to the length of the serialized JSON string.

Example of the *close* message sent by plugin A

```
{
 "apiVersion": 1,
 "method": "close",
 "entity": "activity",
 "backScreen": "plugin_by_label",
 "backPluginLabel": "plugin_b",
 "backPluginOpenParams": {
 "foo": "bar"
 }
}
```

Example of the open message received by plugin B

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activity",
 "openParams": {
 "foo": "bar"
 }
 ...
}
```

error Method

The message for the *error* method always has the 'errors' field that contains the list of errors.

Example of *error* Method Message

```
{
```

```
"apiVersion": 1,
"method": "error",
"entity": "activityList",
"errors": [
{
"type": "TYPE_ENTITY_PROPERTY",
"code": "CODE_ACTIVITY_STATUS_INVALID",
"entity": "activity",
"entityId": "3956532",
"propertyLabel": "astatus"
},
{
"type": "TYPE_ENTITY_PROPERTY",
"code": "CODE_MANDATORY_PROPERTY_EMPTY",
"entity": "inventory",
"entityId": "20998086",
"propertyLabel": "inv_aid"
}
]
}
```

Each element of the errors list is an object and contains these fields:

- type: Describes the type of error that occurred while processing a message, for example, invalid property value, internal error. Type determines the additional fields that are available in the error object, for example, property label.
- code: Describes the error more specifically, for example, a validation rule that is violated by the data sent by a plugin.

The elements may contain additional fields, such as entity, entityId, propertyLabel depending on the type of the error.

Related Topics

- Error Types and Error Codes
- Error Types for Inventory Actions
- callProcedure Error Handling

update Method

You can use the *update* method to update Oracle Fusion Field Service entities through a plugin, without leaving the plugin's page.

Oracle Fusion Field Service validates the format of the *update* message and processes it in the same way as the *close* message. The differences between the *update* and *close* methods are:

- *update* messages may be sent by the plugin multiple times before closing. However, the plugin cannot send the next *update* message until the previous *update* is applied and the *updateResult* message is sent to the plugin.
- The plugin page is not closed after applying of *update* message and its iframe is not destroyed.
- The message fields wakeupNeeded, backScreen, backActivityId, backInventoryId, backPluginLabel, backPluginButtonId, backPluginOpenParams, and iconData are ignored.
- Upon successful processing of the *update* message, Oracle Fusion Field Service sends a message with the *updateResult* method.

If the validation or processing of the *update* message fails, the Plugin API sends the "error" message of same format (with the same "type" and "code" values) that is sent for the *close* message.

If a user has to stay on the plugin page after applying the updates, the best practice is to use the *update* method instead of *close*. This improves the user experience and reduces the consumption of the device's resources (RAM and CPU), as the plugin page won't be recreated and the plugin doesn't have to process the open data again.

Example of the *update* method:

```
{
 "apiVersion": 1,
 "method": "update",
 "activity": {
 "caddress": "Cleveland",
 "aid": "4224031"
 }
}
```

updateResult Method

Oracle Fusion Field Service sends the *updateResult* message in response to the *update* message. It contains the latest available entity data, including the changes applied by the last *update* message. The *updateResult* message has the same format and contents as an *open* message. This allows the plugin to work with actual data without having to close and reopen the plugin page.

If plugin has to get the latest entity data without changing anything, it may just send the *update* message with no entity collections. However, be aware that the changes applied on the server by other users or through REST API are not delivered instantly to the user's device, and the contents of the *updateResult* message show the current state of the entities on a device (in a particalar session of a browser or the installed app).

Example of *update* message:

```
{
 "apiVersion": 1,
 "method": "updateResult",
 "activityList": {
 "4224031": {
 "WO_COMMENTS": null,
 "cname": null,
 "caddress": "Cleveland",
 "ccity": null,
 "aworktype": "4",
 "astatus": "pending",
 "aid": "4224031",
 "atype": "regular"
 }
 },
 "inventoryList": {
 "21064417": {
 "invsn": "PTI1234789",
 "invpool": "provider",
 "invid": "21064417",
 "inv_aid": null,
 "inv_pid": 3000001
 }
 },
 "queue": {
```

```
"date": "2021-08-17",
"status": "notActivated",
"isActual": true
},
"resource": {
"pid": 3000001,
"external_id": "33001",
"currentTime": "2021-08-17 20:48:26",
"deviceUTCDiffSeconds": 0,
"timeZoneDiffSeconds": -14400
},
"team": {
"teamMembers": {},
"assistingTo": {},
"assistingMe": []
},
"user": {
"allow_desktop_notifications": 1,
"allow_vibration": 1,
"design_theme": 11,
"format": {
"date": "m/d/y",
"long_date": "l, F jS, Y",
"time": "h:i A",
"datetime": "m/d/y h:i A"
},
"providers": [
2
],
"sound_theme": 2,
"su_zid": 2,
"uid": 2315,
"ulanguage": 1,
"language": "en",
"ulogin": "admin",
"uname": "Admin",
"week_start": 0
},
"buttonId": "20348",
"openParams": {}
}
```


Example of an empty *update* message for data refresh:


```
{
"apiVersion": 1,
"method": "update"
}
```


callProcedure Method

The *callProcedure* method lets the plugin interact with Oracle Fusion Field Service Core Application without leaving the plugin's page. This method is implemented using the remote procedure call (RPC) approach.

The *callProcedureResult* method returns the result of running the *callProcedure*. As you can run multiple procedures simultaneously, it is possible that the plugin sends the next *callProcedure* messages to Oracle Fusion Field Service before it sends the *callProcedureResult* message back. These results can be associated with the procedure calls using

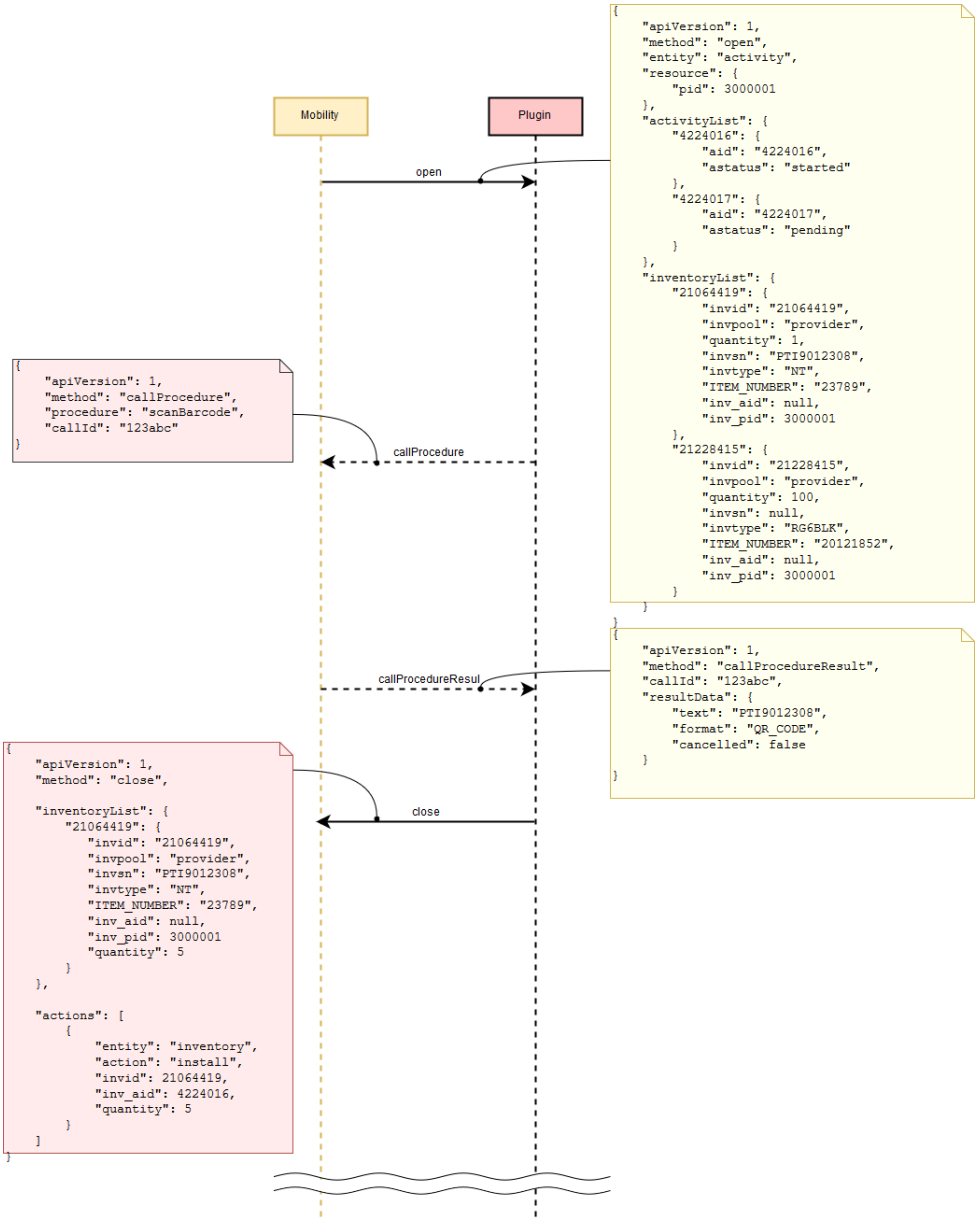
the *callId* parameter. For backward compatibility, simultaneous calls of multiple *openLink* procedures or multiple *scanBarcode* procedures cause an error with the error code CODE_METHOD_UNEXPECTED.

You can perform these actions on a plugin through the *callProcedure* method:

- Open a plugin in the background (*wake up*) even when the device is offline.
- Define a delay for opening a plugin in the background after closing its window. You can set it as short as 10 seconds.
- Extend the period of background operation for a plugin before being required to close (sleep). You can set it between 10 seconds and 1 hour.
- Update the appearance (icon, color, text) of a plugin's button (or multiple buttons) on **My Route** without closing a plugin that is working in the background.
- Navigate to En route and Stop travel pages when the plugin is closed.
- Call some procedures through the Plugin API without actually calling them.

callProcedure Sequence

This diagram shows the sequence in which *callProcedure* is run:



callProcedure method parameters

This table lists the parameters of the *callProcedure* method:

Parameter Name	Mandatory	Type	Description
apiVersion	Yes	Integer	Plugin API version.
method	Yes	String	Must equal *callProcedure*.
procedure	Yes	String	Procedure name.

Parameter Name	Mandatory	Type	Description
callId	Yes	String	Unique string identifier, which is used to apply the procedure response within the plugin.

Example of *callId* Generation

```
function generateCallId() {
 return btoa(String.fromCharCode.apply(null, window.crypto.getRandomValues(new Uint8Array(16))));
}
```

Calling of Procedures

You can send the *callProcedure* messages for plugins that are opened in background after they receive a *wakeup* message. You can use only the *updateIconData* and *updateButtonsIconData* procedures with the plugins that are opened in background. These procedures help the plugin update the appearance of its buttons in real-time to notify the user about the updates it has received.

Related Topics

- callProcedureResult Method

scanBarcode Procedure

Parts and equipment usually have barcodes printed on their package. The *scanBarcode* procedure provides the barcode and 2D (for example, QR, DATAMATRIX) code scanner functionality that helps searching for items in the inventory pools easy.

When the plugin calls this procedure, the scanner window opens and shows the live camera picture. When the barcode is recognized, the scanner window closes, and the result is sent to the plugin through the *callProcedureResult* method. If the barcode scanner is unavailable or Oracle Fusion Field Service Core Application isn't run inside the Oracle Fusion Field Service Mobile (for Android or iOS) application, an error code is returned to the plugin through an error message. For more information about the *callProcedureResult* method, see the callProcedureResult Method topic in the Mobile Plug-in Framework guide.

> **Note:** You must have the Oracle Fusion Field Service Mobile (for Android or iOS) application to use the Barcode Scanner through the plugin API and to search by barcode.

Supported Barcode and 2D Code Types

This table provides the barcode and 2D code types:

Barcode Type	Android	iOS
QR_CODE	Yes	Yes
DATA_MATRIX	Yes	Yes
UPC_A	Yes	Yes
UPC_E	Yes	Yes

Barcode Type	Android	iOS
EAN_8	Yes	Yes
EAN_13	Yes	Yes
CODE_39	Yes	Yes
CODE_93	Yes	No
CODE_128	Yes	Yes
CODABAR	Yes	No
ITF	Yes	Yes
RSS14	Yes	No
PDF417	Yes	No
RSS_EXPANDED	Yes	No

Example of the *callProcedure* Message

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "scanBarcode",
 "callId": "123abc"
}
```

Result of the *callProcedure* Procedure

For this procedure, the resultData param of the callProcedureResult message is an object, which contains these fields:

Parameter	Type	Description
apiVersion	String	Plugin API version.
format	String	Type of recognized barcode. See Supported barcode and 2D code types.
cancelled	String	Equals true if user closed the scanner window before the code's recognized.

Example of the *callProcedureResult* Message

When the barcode is scanned successfully:

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123abc",
 "resultData": {
 "text": "PT9012308",
 "format": "QR_CODE",
 "cancelled": false
 }
}
```

When user cancels scanning:

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123abc",
 "resultData": {
 "text": "",
 "format": "",
 "cancelled": true
 }
}
```

openLink Procedure

The openLink procedure provides a common way to open external URLs from Oracle Fusion Field Service Core Application run either in a web browser or in the Oracle Fusion Field Service Mobile for Android and iOS app. If Oracle Fusion Field Service Core Application is run in the Oracle Fusion Field Service Mobile for Android and iOS app, the URL is opened in a new web browser window. If not, it's opened as a new browser tab.

Example of the *callProcedure* Message

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "openLink",
 "callId": "123abc",
 "params": {
 "url": "https://play.google.com/store/apps/details?id=com.oracle.ofs"
 }
}
```

Result of the Procedure

The result is sent through the *callProcedureResult* message, just to indicate that the procedure is run successfully. The *resultData* param doesn't contain any data.

getPartsCatalogStructure Procedure

The *getPartsCatalogStructure* procedure returns the field type and schema for the available Parts Catalogs through the *callProcedureResult* method. This is the same data that you have configured using the *create* method of the Parts Catalog API.

Parameters

The *getPartsCatalogStructure* procedure doesn’t accept any parameters.

Response for the *getPartsCatalogStructure* Procedure

Here are the fields that are returned for every Parts Catalog:

Field Name	Type	Parts Catalog API Parameter	Description
catalogId	Number	None	None
label	String	label	None
language	String	language	None
name	String	name	Name of the catalog to be displayed in the user-interface.
fieldSchemas	Array	field_schemas	Array of fieldSchema items, each of which contains one of the fields to be set for the catalog.
typeSchemas	Array	type_schemas	Array of typeSchema items, each of which contains an item type and may also contain the inventory type corresponding to the item type.
cacheLoadingState	Object	None	Status of loading the Parts Catalog contents to the device storage to be available in offline. Can be used to visualize the progress of loading.

fieldSchema Item Structure

Here is the structure of the *fieldSchema* item:

Field Name	Type	Parts Catalog API 'typeSchema' Element Parameter	Description
label	String	label	Field identifier
name	String	name	Name of the field to be displayed in the user-interface.
propertyLabel	String	property_label	Label of the corresponding inventory property in Oracle Fusion Field Service.
searchable	Boolean	searchable	Whether the field is used for search.
preview	Boolean	preview	Whether the field is cached in offline mode and displayed in the offline search results.

typeSchema Item Structure

Here is the structure of the *typeSchema* item:

Field Name	Type	Parts Catalog API 'typeSchema' Element Parameter	Description
itemType	String	item_type	Item type according to the catalog.

Field Name	Type	Parts Catalog API 'typeSchema' Element Parameter	Description
inventoryType	String	inventory_type	Inventory type according to Oracle Fusion Field Service settings.

cacheLoadingState Item Structure

Here is the structure of the *cacheLoadingState* item:

Field Name	Type	Description
isLoaded	String	Whether the all needed data of Parts catalog is loaded and is available offline
loadedItemsNumber	Number	Number of catalog items that are loaded to the device
totalItemsNumber	Number	Total number of catalog items to be loaded to the device
loadedSize	Number	Amount of data that's loaded to the device (bytes)

Example of a Request

```
{
 "apiVersion": 1,
 "callId": "CMFYN5AKpc9Yg1POv6773g==",
 "method": "callProcedure",
 "procedure": "getPartsCatalogsStructure"
}
```

Example of a Response

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "CMFYN5AKpc9Yg1POv6773g==",
 "resultData": [
 {
 "catalogId": 2,
 "label": "network",
 "language": "en",
 "name": "Network devices",
 "fieldSchemas": [
 {
 "label": "model",
 "name": "Model",
 "propertyLabel": "switch_model",
 "searchable": true,
 "preview": true
 },
 {
 "label": "ports",
 "name": "Ports",
 "propertyLabel": false,
 "searchable": true,
 "preview": false
 },
 {
```

```
"label": "price",
"name": "Price",
"propertyLabel": "switch_price",
"searchable": false,
"preview": false
},
{
"label": "vendor",
"name": "Vendor",
"propertyLabel": false,
"searchable": false,
"preview": true
}
],
"typeSchemas": [
{
"itemType": "switch_type",
"inventoryType": "switch_general"
},
{
"itemType": "router_type",
"inventoryType": "router_general"
}
],
"cacheLoadingState": {
"isLoaded": true,
"loadedItemsNumber": 2,
"loadedSize": 1138,
"totalItemsNumber": 2
}
},
{
"catalogId": 4,
"label": "misc",
"language": "en",
"name": "Miscellaneous parts",
"fieldSchemas": [
{
"label": "item_type",
"name": "Item Type",
"propertyLabel": false,
"searchable": false,
"preview": false
},
{
"label": "description",
"name": "Item description",
"propertyLabel": false,
"searchable": true,
"preview": true
},
],
"typeSchemas": [
{
"itemType": "parts",
"inventoryType": "PART"
}
],
"cacheLoadingState": {
"isLoaded": true,
"loadedItemsNumber": 3558,
"loadedSize": 2793309,
"totalItemsNumber": 3558
}
}
]
```

```
}
```

getParts Procedure

The *getParts* procedure returns all the information for a Parts Catalog item, that was added using the *upload* method of the Parts Catalog API.

Parameters

Here are the parameters of the *getParts* procedure:

Parameter Name	Mandatory	Type	Description
items	Yes	Array	Array of itemKey objects that identify the catalog items to get info for.

itemKey Object Structure

Here is the structure of the *itemKey* object:

Parameter Name	Mandatory	Type	Parts Catalog API Parameter for 'item' Element of 'upload_catalog' Request	Description
catalogId	Yes	Integer	None	A unique identifier of a catalog, which contains the item. Id can be retrieved using the *getPartsCatalogsStructure* procedure.
label	Yes	String	label	A unique identifier of a part within a catalog. Can be changed after updating the catalog.

Response for the *getParts* Procedure

Here are the fields that are returned:

Field Name	Type	Description
items	Array	Array of FoundItem objects. Each object represents one item whose data is available.
notFoundItems	String	Array of itemKey objects. Each object contains the key fields of an item whose data is not available.

foundItem Object Structure

Here is the structure of the *foundItem* object:

Field Name	Type	Parts Catalog API Parameter for 'item' Element of 'upload_ catalog' Request	Description
catalogId	Integer	None	A unique identifier of a catalog.
itemId	Integer	None	A unique identifier of a part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a part within a catalog.
itemType	String	type	Item type
inventoryType	String	None	Label of a corresponding inventory type in Oracle Fusion Field Service. Can be empty if the mapping of item types to inventory types isn't configured in the catalog schema. Mapping between *itemType* and *inventoryType* can be found in the *typeSchemas* field of the *getPartsCatalogStructure* procedure result.
fields	Object	fields	An object (dictionary) that contains the item's fields by it's labels.
linkedItems	Array	linked_items	Array of LinkedItem objects.
images	Array	images	An array of strings, where each string is a URL of an image.

LinkedItem Object Structure

Here is the structure of the linkedItem object:

Field Name	Type	Parts Catalog API Parameter for 'LinkedItem' Element of 'upload_catalog' Request	Description
id	Integer	None	A unique identifier of a linked part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a linked part within a catalog.
displayData	String	display_data	Text comments to the linked item to be displayed in GUI.

Example of a Request

```
{
 "apiVersion": 1,
 "callId": "KnnXUxS7APzLBVIzY+8B0g==",
 "method": "callProcedure",
 "procedure": "getParts",
 "params": {
 "items": [
 {
```

```
"catalogId": "2",
"label": "Switch_model_001"
},
{
"catalogId": "2",
"label": "Switch_009"
},
{
"catalogId": "4",
"label": "FM3-2048-007"
}
]
}
}
```


Example of a Response


```
{
"apiVersion": 1,
"method": "callProcedureResult",
"callId": "KnnXUxS7APzLBVIzY+8B0g==",
"resultData": {
"items": [
{
"catalogId": 2,
"itemId": 2,
"label": "Switch_001",
"itemType": "switch_type",
"inventoryType": "switch_general_eta",
"linkedItems": [
{
"id": 3,
"label": "Switch_002",
"displayData": "better"
}
],
"fields": {
"descr": "Automatic direction",
"model": "Switch_model_001",
"ports": "8 x Fast Ethernet (10/100 Mbit/s)",
"price": "25$",
"size": "151x81x33 mm,200 g",
"vendor": "Oracle"
},
"images": [
"https://example.com/switch_1.jpg"
]
},
{
"catalogId": 4,
"itemId": 3463,
"label": "FM3-2048-007",
"itemType": "parts",
"inventoryType": "",
"linkedItems": [
{
"id": 2350,
"label": "Z4603011",
"displayData": "5"
},
{
"id": 3160,
"label": "Z0293015",
"displayData": "7"
},
],
```

```
"fields": {
"cost": "54.2346747003",
"description": "NETWORK I/F BOARD PCB ASSY\nFirmware update 2",
"item_disposition": "Repairable",
"item_type": "BOARD",
"price": "174.85",
"vendor": "ORCL"
},
"images": [
"https://example.com/pictures/12.jpg",
"https://example.com/pictures/8.jpg",
"https://example.com/pictures/23.jpg",
"https://example.com/pictures/14.jpg"
]
}
],
"notFoundItems": [
{
"catalogId": 2,
"label": "Switch_009"
}
]
}
}
```


getAuthorizationCode

OAuth Code Authorization Flow in Plugin API

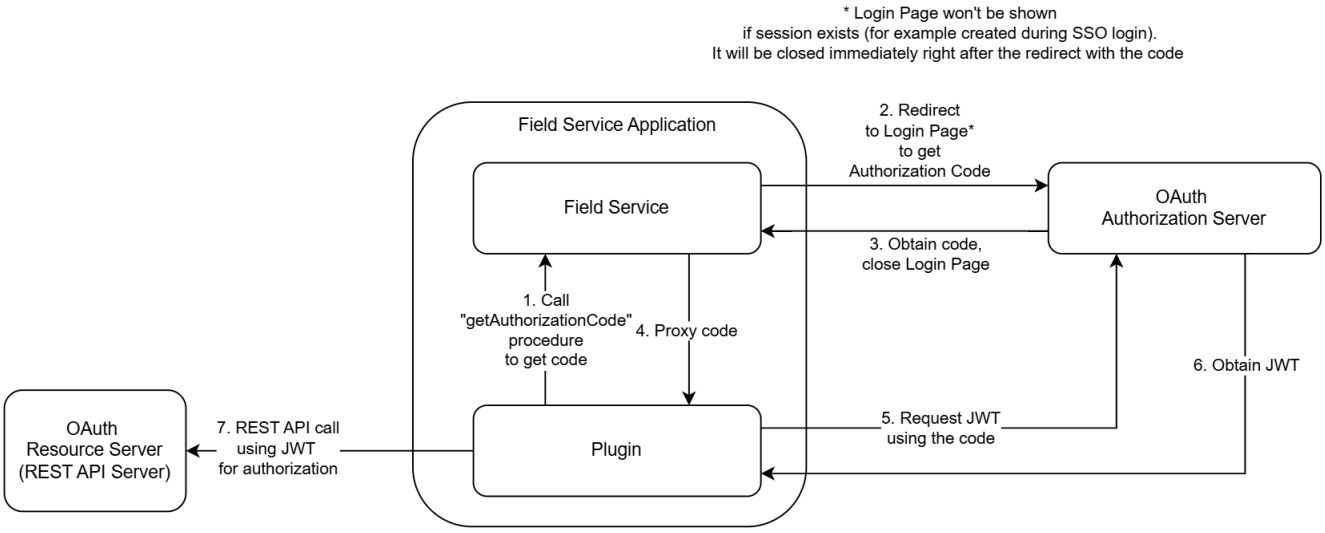
The OAuth Code Authorization flow in the Plugin API enables the acquisition of a JWT (access token) by using an already authenticated user session in Oracle Fusion Field Service through an Identity Provider. With the JWT access token, you must authenticate to the REST API of your Identity Provider to execute necessary actions, such as verifying user permissions, tracking data updates made by the user, and so forth.

The flow is detailed here:

- Configure an application on an Identity Provider that supports the OAuth Code Authorization flow.
- Use credentials such as Client ID, Scope, and Identity Provider endpoint to generate the URL to the Identity Provider Code Authorization endpoint.
- Call the "getAuthorizationCode" procedure from the plugin with this URL in the procedure parameters.Obtain an authorization code in the procedure response.
- Obtain a JWT access token by this authorization code from the plugin.
- Use the JWT access token in the REST API request authorization.

Note: The JWT access token is issued on behalf of a user rather than an application.

This flowchart shows the OAuth Code Authorization Flow in Plugin API:



getAuthorizationCode Procedure

The **getAuthorizationCode** procedure is used to obtain an Access token (JWT) using the OAuth 2.0 Authorization Code Grant flow. The code needs to obtain a JWT (access token) which is used to authorize REST API calls to the Resource Server.

Procedure Overview

The procedure accepts only one **mandatory** parameter - URL. This is the URL of the Identity Provider endpoint, which provides authorization by OAuth 2.0 Authorization Code Grant Flow. Usually the URL contains the "/authorize" suffix.

Parameters

The URL consists of several parameters as shown in the table:

Parameters	Required/Optional	Description
REST API Endpoint	Required	Provides OAuth 2.0 Authorization Code Flow.For example: *https://idcs-instance.example.com/oauth2/v1/authorize*
response_type=code	Required	The parameter that points to the Authorization Code Flow.
client_id	Required	The Client ID, provided by Identity Provider during the creation of OAuth 2.0 Authorization Code application. For example: client_id=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
redirect_uri	Required	The URL to which the user will be redirected. It is

Parameters	Required/Optional	Description
		URL encoded and consists of Oracle Fusion Field Service instance domain and "/plugin-auth-redirect/" suffix. For example: &redirect_uri=https%3A%2F%2Fofs-instance.example.com%2Fplugin-auth-redirect%2F
scope	Required	The scope requested for authorization. For example: '&scope=xxxx'. It should be one of scopes that is supported by OAuth 2.0 Authorization Code application (configured on Identity Provider).
Optional parameters, to support Proof Key for Code Exchange (PKCE) *https://tools.ietf.org/html/rfc7636*		
code_challenge_method=S256	Optional	The type of encryption method used. For example: code_challenge_method=S256. (The value could be "plain" but in that case Code Verifier and Code Challenge is equal)

code_challenge	Optional	The signature of the Code Verifier string that is generated by the following code example:
		
		async function generateCodeChallenge(codeVerifier) {
		
		const encoder = **new** TextEncoder();
		
		const data = encoder.encode(codeVerifier);
		
		const digest = await crypto.subtle.digest('SHA-256', data);
		
		return base64UrlEncode(digest);
		
		}
		
		The Code Verifier string should be used in the request to obtain JWT.

state	Optional	This can be used to keep some information when user will be redirected. Usually, the parameter is used to keep the information of plugin's current screen or some other

Parameters	Required/Optional	Description
		information of navigation inside the plugin before the procedure had been called.

Note: Oracle Fusion Field Service environment domain can be taken from the 'origin' item of the init message.

Example of a *Request*

The example of the procedure request to IDCS Identity Provider:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "getAuthorizationCode",
 "callId": "1111111111",
 "params": {
 "url": "https://idcs-instance.example.com/oauth2/v1/authorize?
response_type=code&client_id=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&redirect_uri=https%3A%2F%2Fofs-
instance.example.com%2Fplugin-auth-redirect%2F
 &scope=urn:opc:resource:faaas:fa:XXXXXXXXX-XXXXurn:opc:resource:consumer::all"
 }
 }
```

The documentation of configuration of Authorization Code Flow in IDCS - *https://docs.oracle.com/en-us/iaas/Content/Identity/api-getstarted/AuthCodeGT.htm*

The example of the procedure request to Microsoft Entra Identity Provider:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "getAuthorizationCode",
 "callId": "1111111111",
 "params": {
 "url": "https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/authorize?
response_type=code&client_id=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 &redirect_uri=https%3A%2F%2Fofs-instance.example.com%2Fplugin-auth-redirect
%2F&scope=User.Read&code_challenge_method=S256&code_challenge=VmLy6wpzYdHI99H0yeP64qEAyjJL_gu915gJUplobBA"
 }
}
```

The documentation of configuration of Authorization Code Flow in Microsoft Entra - https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow.

callProcedureResult method - success, completed

The response contains three fields:

- redirectUrl (mandatory) - the full URL which was used to redirect user to Oracle Fusion Field Service.
- code (mandatory) - the authorization code.
- state (optional) - if such parameter is present in redirectUrl it will be returned separately, just to make easier to get it, without parsing the redirectUrl.

```
{

 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "1111111111",
```

```
 "procedure": "getAuthorizationCode",
 "resultData": {
 "result": "completed",
 "code": "xxxxx",
 "redirectUri": "https://ofs-instance.example.com/plugin-auth-redirect/?
code=xxxxxx&some_other_return_param=xxxxx",
 "state": "some_state"
 }
}
```

callProcedureResult method - success, cancelled

This occurs when a new request is made. Once a new browser tab is opened, Oracle Field Service cannot track the user's actions and won't receive an event if the tab is closed. Therefore, the only way to obtain the code is by making an additional request. In this scenario, the previous procedure call will return a "callProcedureResult" with a "cancelled" result.

```
{

 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "1111111111",
 "procedure": "getAuthorizationCode",
 "resultData": {
 "result": "cancelled",
 "reason": "SAME_PROCEDURE_NEW_CALL_BEFORE_COMPLETION"
 }
}
```

callProcedureResult method - error

The example of the error messages that could be returned:

```
{

 "apiVersion": 1,

 "method": "error",

 "errors": [

 {

 "type": "TYPE_PROCEDURE_ERROR",

 "code": "CODE_UNKNOWN",

 "procedure": "getAuthorizationCode", "data": "Authorization Code obtaining is rejected. The mandatory
 parameter \"code\" is absent in redirect URI: https://ofs-instance.example.com/plugin-auth-redirect/

 ?error=invalid_request&error_description=Client+xxxx+provided+an+invalid+response+mode%3A+query."

 }

 ],

 "callId": "xxxx"

}

{

 "apiVersion": 1,
```

```
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_UNAVAILABLE",

 "procedure": "getAuthorizationCode"
 }
 ],

}
```

The example of fetch requests that could be used to get JWT and RESt API data from Microsoft:

```
fetch('https://login.microsoftonline.com/xxxx/oauth2/v2.0/token', {
 headers: {
 "content-type": "application/x-www-form-urlencoded; charset=UTF-8",
 },
 method: "POST",
 body: 'client_id=xxxx&grant_type=authorization_code&redirect_uri=https%3A%2F%2Fofs-instance.example.com
%2Fplugin-auth-redirect%2F&code=xxxx'
});

fetch("https://graph.microsoft.com/v1.0/me", {
 headers: {
 authorization: 'Bearer xxxx'
 }
});
```

searchParts Procedure

The *searchParts* procedure searches for parts in the Parts Catalog.

Parameters

Here are the parameters of the *searchParts* procedure:

Parameter Name	Mandatory	Type	Description
query	Yes	String	Search query. Minimum length - 3 symbols (spaces symbols at the beginning and at the end are trimmed), maximum length - 100 symbols.
limit	No	Integer	Maximum number of results returned as a result. Minimum value - 1, maximum - 1000. Default is 10.
cacheOnly	No	Boolean	Whether the search is to be performed only in cache, or it can make a network request to finish the search. If set to true, the search is performed in offline mode without any network requests. Default is false.

Response

Here is the response for of the *searchParts* procedure:

Field Name	Type	Description
items	Array	Array of *FoundItem* objects. Found parts. Limited by the *limit* parameter.
source	String	Source of the search. Possible values: • "cache": The search is performed only in the cache, no network request is sent. • "server": The search is performed with a network request.
searchId	Integer	A unique id of the search procedure within the plugin's open session. Used as a parameter for the *searchPartsContinue* procedure.
isContinueAvailable	Boolean	Indicates whether the total number of results overflows the *limit* parameter or not. If it's true - then the *searchPartsContinue* procedure can be used to return more results (limited by the limit parameter).

foundItem Object Structure

Here is the structure of the *foundItem* object:

Field Name	Type	Parts Catalog API Parameter for 'item' Element of 'upload_catalog' Request	Description
catalogId	Integer	None	A unique identifier of a catalog.
itemId	Integer	None	A unique identifier of a part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a part within a catalog.
itemType	String	type	Item type
inventoryType	String	None	Label of a corresponding inventory type in Oracle Fusion Field Service. Can be empty if the mapping of item types to inventory types isn't configured in the catalog schema. Mapping between *itemType* and *inventoryType* can be found in the *typeSchemas* field of the *getPartsCatalogStructure* procedure result.
fields	Object	fields	An object (dictionary) that contains the item's fields by it's labels.
linkedItems	Array	linked_items	Array of LinkedItem objects.
images	Array	images	An array of strings, where each string is a URL of an image.

LinkedItem Object Structure

Here is the structure of the linkedItem object:

Field Name	Type	Parts Catalog API Parameter for 'LinkedItem' Element of 'upload_catalog' Request	Description
id	Integer	None	A unique identifier of a linked part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a linked part within a catalog.
displayData	String	display_data	Text comments to the linked item to be displayed in GUI.

Loading More Search Results

If the *searchParts* procedure returns *isContinueAvailable* as true, then you can load more search results through the *searchPartsContinue* method using the returned *searchId*. The *searchPartsContinue* method is available for 10 most recent *searchId*. The *searchId* values returned by the *searchParts* procedure are stored in a queue for 10 most recent calls that have *isContinueAvailable* = true. If the queue overflows, the *searchId* for the first calls are removed and *searchPartsContinue* isn't available for them any more. Each *searchPartsContinue* call moves it's *searchId* to the top of the queue (makes it most recent) and when *searchPartsContinue* returns *isContinueAvailable* as false, the *searchId* is removed from the queue.

Example of a Request

```
{
 "apiVersion": 1,
 "callId": "3quvGlWIgNJlQhHrYRN4vg==",
 "method": "callProcedure",
 "procedure": "searchParts",
 "params": {
 "query": "055",
 "limit": 5
 }
}
```

Example of a Response

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "3quvGlWIgNJlQhHrYRN4vg==",
 "resultData": {
 "items": [
 {
 "catalogId": 3,
 "itemId": 4179,
 "label": "XG9-0552-000",
 "itemType": "parts",
 "inventoryType": "PT",
 "linkedItems": [
 {
 "id": 4298,
 "label": "D3625170",
 "displayData": "9"
 },
 {
 "id": 4547,
 "label": "D5853100",
 "displayData": "2"
 },
```

```
{
"id": 4824,
"label": "D8093048",
"displayData": "8"
},
{
"id": 6310,
"label": "AB014229",
"displayData": "7"
}
],
"fields": {
"description": "BEARING NP6560",
"vendor": "AGHA",
"cost": "5.704125031",
"price": "22.7",
"item_type": "NA",
"item_disposition": "NA"
},
"images": [
"https://example.com/picture/15.jpg",
"https://example.com/picture/10.jpg",
"https://example.com/picture/18.jpg",
]
},
{
"catalogId": 3,
"itemId": 5631,
"label": "KHB670550A00",
"itemType": "parts",
"inventoryType": "PT",
"linkedItems": [],
"fields": {
"description": "KID-MOD MF16, TENSION-DETACK",
"vendor": "KODAK",
"cost": "742.5143066464",
"price": "1270.86",
"item_type": "NA",
"item_disposition": "NA"
},
"images": [
"https://example.com/picture/26.jpg"
]
},
{
"catalogId": 3,
"itemId": 5029,
"label": "D0605507",
"itemType": "parts",
"inventoryType": "PT",
"linkedItems": [
{
"id": 3972,
"label": "FB6-2374-000",
"displayData": "0"
},
{
"id": 4975,
"label": "B1253830",
"displayData": "8"
}
],
"fields": {
"description": "PCB:B-C4B:SERVICE:ASS'Y",
"vendor": "HYTEC",
"cost": "1427.04",
```

```
"price": "2854.08",
"item_type": "BOARD",
"item_disposition": "Repairable"
},
"images": []
},
{
"catalogId": 3,
"itemId": 7551,
"label": "D3305502",
"itemType": "parts",
"inventoryType": "PT",
"linkedItems": [],
"fields": {
"description": "[XREF TO HD3305502] DF MAIN BOARD",
"vendor": "NWRS",
"cost": "191.5",
"price": "480.95",
"item_type": "BOARD",
"item_disposition": "Repairable"
},
"images": [
"https://example.com/picture/2.jpg"
]
},
{
"catalogId": 3,
"itemId": 4203,
"label": "D0746055",
"itemType": "other",
"inventoryType": "",
"linkedItems": [
{
"id": 4037,
"label": "AB012067",
"displayData": "1"
}
],
"fields": {
"description": "FLAT BELT-TRANSFER SERVICE PARTS",
"vendor": "SUNRISE",
"cost": "902.0665087976",
"price": "1848.46",
"item_type": "NA",
"item_disposition": "NA"
},
"images": [
"https://example.com/picture/26.jpg"
]
}
],
"isContinueAvailable": true,
"source": "cache",
"searchId": 1
}
}
```


searchPartsContinue Procedure

The *searchPartsContinue* procedure returns additional results for a search that is initiated by the *searchParts* procedure.

Parameters

Here are the parameters of the *searchPartsContinue* procedure:

Parameter Name	Mandatory	Type	Description
searchId	Yes	Integer	A unique id of the search procedure within a plugin's open session. Minimum value: 0, Maximum value: 2147483647 For more information, see Loading More Search Results.

Response

Here is the response for of the *searchPartsContinue* procedure:

Field Name	Type	Description
items	Array	Array of *FoundItem* objects. Found parts. Limited by the *limit* parameter.
source	String	Source of the search. Possible values: • "cache": The search is performed only in the cache, no network request is sent. • "server": The search is performed with a network request.
searchId	Integer	A unique id of the search procedure within the a plugin's open session. Used as a parameter for the *searchPartsContinue* procedure.
isContinueAvailable	Boolean	Indicates whether the total number of results overflows the *limit* parameter or not. If it's true - then the procedure can be called once more to return more results (limited by the *limit* parameter for the *searchParts* procedure).

foundItem Object Structure

Here is the structure of the *foundItem* object:

Field Name	Type	Parts Catalog API Parameter for 'item' Element of 'upload_catalog' Request	Description
catalogId	Integer	None	A unique identifier of a catalog.
itemId	Integer	None	A unique identifier of a part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a part within a catalog.
itemType	String	type	Item type
inventoryType	String	None	Label of a corresponding inventory type in Oracle Fusion Field Service. Can be empty if the mapping of item types

Field Name	Type	Parts Catalog API Parameter for 'item' Element of 'upload_ catalog' Request	Description
			to inventory types isn't configured in the catalog schema. Mapping between *itemType* and *inventoryType* can be found in the *typeSchemas* field of the *getPartsCatalogStructure* procedure result.
fields	Object	fields	An object (dictionary) that contains the item's fields by it's labels.
linkedItems	Array	linked_items	Array of LinkedItem objects.
images	Array	images	An array of strings, where each string is a URL of an image.

LinkedItem Object Structure

Here is the structure of the linkedItem object:

Field Name	Type	Parts Catalog API Parameter for 'LinkedItem' Element of 'upload_catalog' Request	Description
id	Integer	None	A unique identifier of a linked part within a catalog. Can be changed after catalog update.
label	String	label	A unique identifier of a linked part within a catalog.
displayData	String	display_data	Text comments to the linked item to be displayed in GUI.

Example of a Request

```
{
 "apiVersion": 1,
 "callId": "J3wa6jhKxwf6xfAZbsCdjQ==",
 "method": "callProcedure",
 "procedure": "searchPartsContinue",
 "params": {
 "searchId": 1
 }
}
```

Example of a Response

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "J3wa6jhKxwf6xfAZbsCdjQ==",
 "resultData": {
 "items": [
 {
 "catalogId": 3,
 "itemId": 7300,
 "label": "MU220055000",
```

```
"itemType": "parts",
"inventoryType": "PT",
"linkedItems": [
{
"id": 4481,
"label": "CF064-67901",
"displayData": "0"
},
{
"id": 4986,
"label": "B2469510",
"displayData": "5"
}
],
"fields": {
"description": "INTRACK JOGGER AY F4100-02",
"vendor": "KODAK",
"cost": "341.4693432572",
"price": "799.075",
"item_type": "NA",
"item_disposition": "NA"
},
"images": [
"https://example.com/picture/4.jpg",
"https://example.com/picture/26.jpg",
"https://example.com/picture/10.jpg",
"https://example.com/picture/8.jpg"
]
},
{
"catalogId": 3,
"itemId": 6337,
"label": "B8305562",
"itemType": "parts",
"inventoryType": "PT",
"linkedItems": [],
"fields": {
"description": "STEPPER MOTOR DC1 .56V 3.7W",
"vendor": "RICOH",
"cost": "36.97",
"price": "185.7",
"item_type": "NA",
"item_disposition": "NA"
},
"images": [
"https://example.com/picture/14.jpg",
"https://example.com/picture/15.jpg",
"https://example.com/picture/4.jpg"
]
}
],
"isContinueAvailable": false,
"source": "server",
"searchId": 1
}
}
```


Error handling

Examples of the error message


```
{
"apiVersion": 1,
"method": "error",
"callId": "123abc",
```

```
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_UNKNOWN"
 }
 ]
}

{
 "apiVersion": 1,
 "callId": "KnnXUxS7APzLBVIzY+8B0g==",
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_PARAM_ITEM",
 "code": "CODE_PROCEDURE_PARAM_ITEM_MANDATORY_FIELD_EMPTY",
 "procedure": "getParts",
 "paramName": "items",
 "itemId": 2,
 "itemField": "label"
 }
 ]
}
```

Get Access Token

Customers will now get an opportunity to access the Field Service REST API, Fusion REST API or External API from their custom plugin via JWT Access Token. The plugin configuration screen has been extended with the new ‘Applications’ section that allows to choose the available application. Once it’s configured, the Mobile Plugin Framework sends all configured applications to the plugin on ‘init’ stage. The new “getAccessToken” procedure returns the JWT access token that is used for authorization of direct API calls.

Companies have an opportunity to access the Field Service REST API, Fusion REST API or External API from their custom plugin via JWT Access Token. The “getAccessToken” procedure returns the JWT access token that is used for authorization of direct API calls. The procedure is listed in the "allowedProcedure" collection that is sent in the "open"/"wakeup” method and says that the procedure is available.

```
{
 "apiVersion": 1,
 "method": "open"/"wakeup",
 "entity": "...",
 ...
 "buttonId": "...",
 "openParams": {},
 "allowedProcedures": {
 ...
 "getAccessToken": true
 }
}
```

On init stage, OFS returns the data of the configured Application for the particular Plugin in the "init" method.

The "resourceUrl" field is populated with the URL of the resource server. In case of **Field Service API** application, it is populated with the URL of the Plugin in order to use it in direct REST API requests. In the case of **OAuth User Assertion** or **OAuth Client Credentials** application, it is retuned the same as it is configured on the Modify Application screen.

init method

```
{
 "apiVersion": 1,
```

```
 "method": "init",
 "attributeDescription": {},
 "buttons": [],
 "applications": {
 "ofs_rest_api": {
 "type": "ofs",
 "resourceUrl": "https:// plugins-0-ofsc-xxxx.test.fs.ocs.oc-test.example.com",
 },
 "fusion_rest_api": {
 "type": "oath_user_assertion",
 "resourceUrl": "https://fa-xxxx-pintlabfadev.fa.ocs.oc-test.example.com"
 },
 "external_rest_api": {
 "type": "oauth_client_credentials"
 "resourceUrl": "https://external-rest-api-url.example.com"
 }
 }
}
```

init method with application that is not configured on Plugin Edit screen

```
{
 "empty": {
 "type": "unknown",
 "resourceUrl": ""
 }
 }
}
```

callProcedure method

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "callId": "1111111111",
 "procedure": "getAccessToken",
 "params": {
 "applicationKey": "ofs_rest_api"
 }
}
```

callProcedureResult method - success case

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "1111111111",
 "resultData": {
 "token": "...",
 "status": "success",
 "detail": ""
 }
}
```

callProcedureResult method - fail case

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_PARAM",
 "code": "CODE_PROCEDURE_PARAM_VALUE_INVALID",
 "procedure": "getAccessToken",
 "paramName": "applicationKey"
 }
```

```
 ],
 "callId": "1111111111"
}

{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR",
 "code": "CODE_GET_ACCESS_TOKEN_APPLICATION_NOT_CONFIGURED",
 "procedure": "getAccessToken",
 "data": {
 "status": "unexpected_response",
 "detail": "",
 "token": ""
 }
 }
 ],
 "callId": "1111111111"
}
```

print Procedure

You can use the print procedure to implement scenarios when users can print text, text files, or images of pdf files from their devices using the installed Oracle Fusion Field Service applications. When you call this procedure, the Plugin API validates the parameters and calls the native (device or browser) print functionality with the provided parameters. The Plugin API doesn't return information about or respond to problems such as no printer or cancellation.

Example of the callProcedure message:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "print",
 "callId": "123abc",
 "params": {
 "documentType": "pdf",
 "fileObject": "fileObject",
 "text": "Some text string"
 }
}
```

You can see the resultData message in response to the print message, only if there are no validation errors.

Here is an example of the resultData message:

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123abc",
 "resultData": {
 "status": "ok"
 }
}
```

Print procedure params

Param	Value	is Required	Description
documentType	string	yes	Document type of the file to be printed (text, image, html, pdf)
fileObject	fileObject	required Not required only if documentType param is text	Value from the input file with maximum size **10MB** (10240kb) for image, HTML, and PDF.
text	string	required only if documentType param is text	Text string to be printed

If the documentType is text and the fileObject and text parameters are not empty, the text param is printed.

Supported Document Types

documentType	Dependent Field
text	text or fileObject
image	fileObject is required
html	fileObject is required
pdf	fileObject is required

If the HTML file contains JavaScript code, then the code is not processed based on the web security policy. Only the static content is printed.

Supported File Types

File Type	Example of the file name
image/jpeg	*.jpg
image/png	*.png
image/gif	*.gif
text/html	*.html
text/plain	*.txt
application/pdf	*.pdf

For image/gif file type that contains animation, only the first frame is printed. For application/pdf file type, the browser's built-in PDF Viewer is required.

Note: If you want to use the browser for printing, you must make sure that the version of the browser supports printing these types of documents.

Validation

If an error appears, it means that no actions are applied.

Example of an error message:

```
[
 {
 "type": "TYPE_PROCEDURE_PARAM",
 "code": "CODE_PROCEDURE_MANDATORY_PARAM_EMPTY",
 "procedure": "print",
 "paramName": "fileObject"
 }
]
```

updateIconData Procedure

You can implement a plugin to update the appearance of its button (or multiple buttons at once to make it look the same) on **My Route** without closing the plugin page or interrupting its background operation.

The procedure has only the parameter *iconData* that is mandatory and has the same format as the *iconData* field for the close, sleep, and initEnd messages.

Example of the *callProcedure* message with the *updateIconData* procedure:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "updateIconData",
 "callId": "123abc",
 "params": {
 "iconData": {
 "color": "highlight",
 "text": "117",
 "image": new Blob([
 '<?xml version="1.0"?>' +
 '<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny" viewBox="0 0 64 64">' +
 '<rect x="16" y="16" width="32" height="32" fill="#fff" />' +
 '</svg>'
 ], { type: 'image/svg+xml' });
 }
 }
}
```

updateButtonsIconData Procedure

You can implement a plugin to update the appearance of its buttons individually without closing the plugin page or interrupting its background operation.

The updateButtonsIconData procedure has only the *buttonsIconData* parameter that is mandatory and has the same format as the *buttonsIconData* field for the close, sleep, initEnd messages.

Example of *callProcedure* message with *updateIconData* procedure

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "updateButtonsIconData",
 "callId": "123abc",
 "params": {
 "buttonsIconData": {
 "17156": {
 "color": "highlight",
 "text": "123",
 "image": {}
 },
 "17155": {
 "color": "default",
 "text": null,
 "image": {}
 }
 }
 }
}
```

Errors of updateIconData and updateButtonsIconData

This topic contains information about errors, error codes and their descriptions when trying to work with updaeIconData and updateButtonsIconData.

Share Procedure

The Plugin API Framework has been extended with the new **share** procedure that allows you to save and send files that you uploaded from your device or files that were generated by plugins themselves.

Integrators can use the share procedure to address the following scenarios:

- Send text or any type of files through natively available options, such as Outlook, Gmail, WhatsApp, AirDrop, Google Drive, and so on (the application must be installed on your device) using the mobile device with installed iOS or Android Oracle Field Service applications or browser application.
- Save any types of files to the gallery using a mobile device through the installed iOS or Android Oracle Fusion Field Service application or the Oracle Fusion Field Service web application.

- Save any types of files to the local computer using a desktop browser (Sending is not supported on desktop devices).
- Open files with the applications available on the mobile device.

This share procedure is supported in both online and offline modes.

By calling the share procedure, the Plugin API validates parameters and calls the native (device or browser) share functionality with those provided parameters. The plugin API doesnt return info about or respond to problems such as cancellation, no printer, and so on.

Example of the "callProcedure" message:

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "procedure": "share",
 "callId": "123abc",
 "params": {
 "title": "Some text string",
 "fileObject": "fileObject",
 "text": "Some text string"
 }
 }
```

Oracle Fusion Field Service sends the "resultData" message in response to the "share" message only if there are no validation errors.

Example of the "callProcedureResult" message:

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123abc",
 "resultData": {
 "status": "ok"
 }
 }
```

Share procedure parameters:

Parameters of share procedure are as follows:

This table lists the details of each parameter such as the value, whether the parameter is required, and description:

Parameter	Value	is Required	Description
title	string	yes	text string to be shared
fileObject	fileObject	required only if the 'text' parameter is empty	value from input file with max size **50MB** (51200kb) for file
text	string	required only if the 'fileObject' parameter is empty	text string to be shared with max length equal to **50MB** (51200kb)

If the parameter fileObject is empty but the text parameter is not empty, then the title is used like a file name to send (on mobile devices) or save (on desktop).

Supported File Types

There are no restrictions of the file format for the share option. All file types supported by default web sharing API - https://www.w3.org/TR/web-share/

Validation

If an error appears then no actions have been applied.

allowedProcedures Field

The *open* and *wakeup* messages contain the *allowedProcedures* field, which contains a list of procedures that the plugin is allowed to send before it's closed. With this list, a plugin may check the procedures that are available for the current device without calling them. This helps the plugin disable its functions and/or user interface elements that depend on some procedures (for example, scanBarcode, which is available only in Oracle Fusion Field Service Mobile native app).

Only *updateIconData* and *updateButtonsIconData* are available when plugins work in the background. To consider a procedure as available, the plugin must assure that the key for a field in *allowedProcedures* corresponds to the name of the procedure and its value is true.

Example of *open* message with *allowedProcedures*

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "activityList": {
 "4224031": {
 "aworktype": "4",
 "astatus": "pending",
 "aid": "4224031"
 },
 },
 "buttonId": "20361",
 "openParams": {},
 "allowedProcedures": {
 "openLink": true,
 "searchParts": true,
 "searchPartsContinue": true,
 "getParts": true,
 "getPartsCatalogsStructure": true,
 "updateIconData": true,
 "updateButtonsIconData": true,
 "scanBarcode": true
 }
}
```

Example of *wakeup* message with *allowedProcedures*

```
{
 "apiVersion": 1,
 "method": "wakeup",
 "event": "timer",
 "allowedProcedures": {
 "updateIconData": true,
 "updateButtonsIconData": true
 }
}
```

callProcedure Error Handling

This topic describes the error messages and error codes returned by the *callProcedure* method.

Examples of Error Messages

```
{
 "apiVersion": 1,
 "method": "error",
 "callId": "123abc",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_UNKNOWN"
 }
 ]

}

{
 "apiVersion": 1,
 "callId": "KnnXUxS7APzLBVIzY+8B0g==",
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_PARAM_ITEM",
 "code": "CODE_PROCEDURE_PARAM_ITEM_MANDATORY_FIELD_EMPTY",
 "procedure": "getParts",
 "paramName": "items",
 "itemId": 2,
 "itemField": "label"
 }
 ]
}
```

Example Error Message for share Procedure

```
[
 {
 "type": "TYPE_PROCEDURE_PARAM",
 "code": "CODE_PROCEDURE_MANDATORY_PARAM_EMPTY",
 "procedure": "share",
 "paramName": "fileObject"
 }
]
```

Types of Error Messages

The error message contains only errors of these types:

Type	Occurs When	Available Message Fields
TYPE_PROCEDURE_ERROR	Procedure call is not valid due to missed parameters, and procedure is run with errors.	• procedure: Name of procedure on which the error has occurred. • callId (if available): Id of the procedure call that has caused the error. This is same as the callId param of callProcedure method received.

Type	Occurs When	Available Message Fields
TYPE_PROCEDURE_PARAM	Invalid or missed procedure parameters.	• procedure: Name of procedure on which the error has occurred. callId: Id of the procedure call that has caused the error. This is same as the callId param of callProcedure method received.
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR	Procedure is called with issue in Access Token that is absent in Plugin's configuration.	• procedure: Name of procedure on which the error has occurred. callId: Id of the procedure call that has caused the error. This is same as the callId param of callProcedure method received.

Error Codes

The error codes generated by *callProcedure* are as follows:

Code	Error Type	Cause
TYPE_PROCEDURE_ERROR		
CODE_CALL_ID_EMPTY	Validation error	Empty callId param.
CODE_CALL_ID_INVALID	Validation error	Invalid callId param.
CODE_CALL_ID_DUPLICATE	Validation error	Duplicate callId param.
CODE_PROCEDURE_FAILED	Run error	Running of procedure failed due to various reasons.
CODE_PROCEDURE_UNKNOWN	Run error	Procedure was called with unknown procedure name.
CODE_PROCEDURE_UNAVAILABLE	Internal error	Oracle Fusion Field Service Core Application service related to procedure is not available.
CODE_PROCEDURE_ACCEPTS_NO_PARAMS	Validation error	Procedure was called with params.
CODE_PROCEDURE_DEMAND_AT_LEAST_ONE_PARAM	Validation error	The params field of the callProcedure message is empty or is not an object.
CODE_PROCEDURE_MANDATORY_PARAM_EMPTY	Validation error	One of these: • iconData param of updateIconData procedure is not sent or is empty. • buttonsIconData param of updateButtonsIconData procedure is not set.
CODE_PROCEDURE_PARAM_VALUE_INVALID	Validation error	The buttonsIconData param of the updateButtonsIconData procedure is not an object or is empty.
TYPE_PROCEDURE_PARAM		
CODE_PROCEDURE_MANDATORY_PARAM_EMPTY	Validation error	Mandatory param is missed.

Code	Error Type	Cause
CODE_PROCEDURE_PARAM_VALUE_INVALID	Validation error	Param value is not valid.
CODE_PRINT_UNSUPPORTED_PRINT_FILE_TYPE	Validation error	Uploaded file type is not allowed.
CODE_PRINT_ATTACHED_FILE_IS_TOO_LARGE	Validation error	Uploaded file size is more then 50MB
CODE_PRINT_TYPE_AND_PRINT_FILE_FORMAT_NOT_MATCHED	Validation error	documentType and fileObject.type do not match.
CODE_PRINT_BROWSER_DOES_NOT_SUPPORT_PDF_VIEW	Validation error	The browser's built-in PDF Viewer is unavallible.
CODE_SHARE_ATTACHED_FILE_IS_TOO_LARGE	Validation error	Uploaded file size is more then 50 MB.
CODE_SHARE_TEXT_FIELD_IS_TOO_LARGE	Validation error	The 'text' field can not be larger than 52 428 800 symbols, which equals to 50 MB (51200 kb) file size when saved as text in UTF-8.
CODE_SHARE_INVALID_SHARE_FILE	Validation error	Error reading the file, if the file is not a file or is not a blob.
TYPE_PROCEDURE_PARAM_ITEM		
CODE_PROCEDURE_PARAM_ITEM_MANDATORY_FIELD_EMPTY	Validation error	Mandatory field of the item is missing.
CODE_PROCEDURE_PARAM_ITEM_MANDATORY_PARAM_EMPTY	Validation error	One of required fields is empty
CODE_PROCEDURE_PARAM_ITEM_FIELD_INVALID	Validation error	Value of item field is not valid.
TYPE_WAKEUP_PARAM		
CODE_WAKEUP_EVENTS_INVALID	Validation error	*wakeOnEvents* is not a plain object.
CODE_WAKEUP_EVENT_NOT_SUPPORTED	Validation error	*wakeOnEvents* contains a field, whose key is not *online* or *timer*.
CODE_WAKEUP_EVENT_PARAMS_INVALID	Validation error	*wakeOnEvents* contains a field which in not null and is not a plain object.
CODE_WAKEUP_EVENT_PARAM_VALUE_INVALID	Validation error	One of these: • Value of *wakeupDelay* is not an integer number. • Value of *wakeupDelay* is less than 10. • Value of *sleepTimeout* is not an integer number. • Value of *sleepTimeout* is less than 10 or greater than 3600.
TYPE_INTERNAL		

Code	Error Type	Cause
CODE_UNKNOWN	Validation error	Check JSON syntax and method that is sent
CODE_JSON_INVALID		
CODE_METHOD_NOT_SUPPORTED		
TYPE_PROCEDURE_GET_ACCESS_TOKEN_ERROR		
CODE_GET_ACCESS_TOKEN_WRONG_APPLICATION_KEY	Validation error	Do not call the procedure with an application differ from the applications that were received in init method
CODE_GET_ACCESS_TOKEN_APPLICATION_NOT_CONFIGURED	Validation error	Choose an Application on Plugin Configuration Screen
CODE_GET_ACCESS_TOKEN_OFFLINE_NOT_SUPPORTED	Validation error	Run procedure when the device is online
CODE_GET_ACCESS_TOKEN_PROCEDURE_TIMEOUT	Validation error	Usually such long timeout caused problems with internet connection or problems of Authorization server. Normally it should respond in <4 sec.
CODE_PROCEDURE_FAILED	App misconfigured	Choose an Application on Plugin Configuration Screen Activate an OFS Application
	Unexpected response	Check that Auth Server URL is correct. Example: Fusion Token URL is set without suffix "/oauth2/v1/token"
	Cannot infer used id	The field of user should be filled.
	Token service error - invalid request	Check the scope field is not empty, in case of Fusion it is mandatory.
	Token service error - invalid client	Activate an IDCS Application Check that Client ID / Client Secret is correct
	Token service error - invalid grant	Check that certificate is valid
	Token service error - unauthorized client	Check that client has authorization to use the requested grant.
	Token service error - unsupported grant type	Check is Authorization server supports grant_type: client_credentials or urn:ietf:params:oauth:grant-type:jwt-bearer
	Token service error - invalid scope	Check the scope value

Code	Error Type	Cause
	Connection error	Check that Auth Server URL is correct.
	Internal Error	Exceptional case. Contact Administrator.
	Access denied	Exceptional case. Contact Administrator.

Provide Key and Comments of required API to plugin Configurator

Developer can prepare an XML file of a plugin where they can specify the Key and Comment of the application which is required for Plugin to request API. So, once a configurator imports the plugin XML, they can see the 'Applications' section on the Plugin Edit screen with applications that should be assigned to a plugin to provide access to the required API.

- **Key** could be hardcoded in the Plugin's code. Configurator will choose some application from the list that will be used for it.
- **Comment** is used to give to Configurator more details which application should be chosen.

Example of Plugin Applications in import.xml file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<root>
<format version="1"/>
<product version="24.4.0"/>
<plugins>
<plugin label="hosted" action_label="" action_entity="" action_type="addon_action" type="addon">
 <translations>
 <translation lang="en" val="hosted"/>
 </translations>
 <fields>
 <field label="atype" entity="activity"/>
 <field label="aworktype" entity="activity"/>
 <field label="astatus" entity="activity"/>
 </fields>
 <plugin_applications>
 <plugin_application name="EXT" type="oauth_client_credentials" key="external_api" comment="API to update
 work orders in CRM"/>
 <plugin_application name="FUSION" type="oauth_user_assertion" key="employees_api" comment="API to get
 statuses of ordered parts"/>
 <plugin_application name="OFS" type="ofs" key="activity_api" comment="API to get a list of user's
 activities"/>
 </plugin_applications>
 <plugin_data>
 <plugin_data_item path="" post_data="" width="" height="" options="32" user_agent_mask="" sort_order="0"
 native_app_label="" auth_type="" auth_login="">
 <hosted_plugin_data name="hosted" content_hash="...">
 <content><![CDATA[...]]></content>
 </hosted_plugin_data>
 </plugin_data_item>
 </plugin_data>
</plugin>
</plugins>
</root>
```

callProcedureResult Method

A message with the *callProcedureResult* method is sent by Oracle Fusion Field Service to a plugin when Oracle Fusion Field Service calls a procedure using the *callProcedure* method successfully. The message data contains the *callId* field, which is same as the *callId* parameter of the *callProcedure* message, so that the request and response can be unambiguously associated with each other.

callProcedureResult Method Parameters

Here are the parameters of the *callProcedureResult* method:

Parameter Name	Mandatory	Type	Description
apiVersion	Yes	Integer	Plugin API version.
method	Yes	String	*callProcedureResult*.
procedure	Yes	String	Procedure name.
callId	Yes	String	Id of the procedure call, for which the result is returned. This is same as the received *callId* param of the *callProcedure* method.
resultData	No	String	Result of running the procedure.

Example of the *callProcedureResult* Message

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "1111111111",
 "resultData": {
 "token": "...",
 "status": "success",
 "detail": ""
 }
}
```

For more information about the possible responses, see the description of procedures in the callProcedure section.

sleep Message

For the description of the *sleep* method, see the *wakeup* method. The appearance of a plugin button, that is, icon image, status text, and color can be changed using the optional "iconData" parameter. See the *Change the Plug-In Tile Appearance* section for details.

Example of "sleep" message

```
{
 "apiVersion": 1,
```

```
 "method": "sleep",
 "wakeupNeeded": true,
 "iconData": {
 "color": "highlight"
 }
}
```

Related Topics

- Change the Plugin Tile Appearance

wakeup Message

Oracle Fusion Field Service destroys a plugin's iframe window after the *close* message is successfully run, regardless of whether the device is online or offline. So, no JavaScript code runs after the plugin is closed. However, the plugin may have data, which must be synchronized with its server side, Oracle Fusion Field Service REST API, or third-party services. Hence, the *wakeup* parameter is available with the *close* message.

wakeupNeeded with *close*

The optional parameter *wakeupNeeded* is added to the *close* message to let the plugin synchronize its data as mentioned earlier. If *wakeupNeeded* is set to true, the Oracle Fusion Field Service Core Application opens the plugin's hidden iframe in the background, as soon as Oracle Fusion Field Service Core Application is online, but no earlier than 10 seconds after the plugin is closed. After the plugin iframe is opened, Oracle Fusion Field Service Core Application sends the *wakeup* message to the plugin, in response to the *ready* message. The plugin then sends the *sleep* message back to Oracle Fusion Field Service Core Application, when it finishes synchronization. This lets Oracle Fusion Field Service Core Application destroy the iframe.

If the plugin tries to synchronize, but still has data to be sent, it sends the *sleep* message with the *wakeupNeeded* param set to true. In this case, Oracle Fusion Field Service Core Application opens the plugin's iframe in the background again, as soon as Oracle Fusion Field Service Core Application is online, but no earlier than five minutes after the plugin is closed. If the plugin doesn't send the *sleep* message in two minutes (120 s) after the *wakeup* message is sent, Oracle Fusion Field Service Core Application destroys its iframe and reopens it again, as if the plugin sent the *sleep* message with the *wakeupNeeded* param set to true.

Example of *close* with *wakeupNeeded*

```
{
 "apiVersion": 1,
 "method": "close",
 "activity": {
 cname: "John"
 },
 "wakeupNeeded": true,
 "wakeOnEvents": {
 "online": { wakeupDelay: 120 },
 "timer": { wakeupDelay: 10, sleepTimeout: 1800 }
 }
}
```

wakeupNeeded with *initEnd*

The optional parameter *wakeupNeeded* is added to the *initEnd* message, to let the plugin synchronize even after refreshing Oracle Fusion Field Service Core Application or closing the browser. If the plugin doesn't synchronize within two minutes that is allowed for initialization, it sends the *initEnd* message with the *wakeupNeeded* parameter set to true.

In this case, Oracle Fusion Field Service Core Application opens the plugin's iframe in the background, as soon as Oracle Fusion Field Service Core Application is online, but no earlier than five minutes after it receives the *initEnd* message.

The plugin opens in five minutes after it's closed, if the *wakeupNeeded* parameter of *close*, *sleep*, or *initEnd* messages is set to true. This happens even if Oracle Fusion Field Service Core Application doesn't detect the offline mode, when the plugin is opened or closed. If the user opens the plugin by clicking its button, the background iframe is destroyed without sending any messages to the plugin. If the plugin still has data to be synchronized, it sends the *close* message with the *wakeupNeeded* parameter set to true.

Example of *wakeup* Message for the *online* event

```
{
 "apiVersion": 1,
 "method": "wakeup",
 "event": online
}
```

Example of *wakeup* Message for the *timer* event

```
{
 "apiVersion": 1,
 "method": "wakeup",
 "event": timer
}
```

Configure the Frequency and Duration of a Background Operation

Apart from the *wakeupNeeded* field, you can also use the optional field *wakeOnEvents* to control the frequency and duration of a plugin's background operation. This field is applicable only for *close*, *initEnd*, and *sleep* methods. If the *wakeupNeeded* field is absent, empty, or is set to false, then the *wakeOnEvents* field is ignored.

The value of *wakeOnEvents* is an object with two possible keys, which define the event for which the plugin must be opened for background operation:

- online: If this field is set and is not null, the plugin is opened in the background only when Oracle Fusion Field Service is online, as if *wakeOnEvents* field was not sent.
- timer: If this field is set and is not null, the plugin is opened in the background regardless of the connection status.

The value of these fields has the same format - it's an object with two optional fields:

Field	Type	Min Value	Max Value	Default Value	Description
wakeupDelay	Number (integer)	10	-	300	Delay (in seconds), after which Oracle Fusion Field Service opens the plugin in the background and sends a *wakeup* message. Oracle Fusion Field Service *wakes* a plugin as close as possible to the requested time, but not earlier than that. The actual time may be longer because of the browser's limitations.

Field	Type	Min Value	Max Value	Default Value	Description
sleepTimeout	Number (integer)	10	3600	120	Duration in seconds, after which Oracle Fusion Field Service forcibly closes a background frame of the plugin if it hasn't sent a *sleep* message explicitly. This period starts when Oracle Fusion Field Service sends the *wakeup* message to the plugin.

If both *online* and *timer* are set, the plugin is opened on the first event for which all conditions are met (*wakeupDelay* period has passed, Oracle Fusion Field Service is online (for the *online* event)).

If both fields have the same value for *wakeupDelay* and Oracle Fusion Field Service is online, then there's no guaranteed order of *wakeup* events.

The default value for *wakeOnEvents* is { online: {} }. That is, the plugin is woken only on an online event with a default delay to maintain backward compatibility.

If *wakeOnEvents* is set and is empty, or all its fields equal to null, it's equivalent to *wakeupNeeded*: false.

Background synchronization schedule is discarded as soon as the plugin sends the *close*, *initEnd*, or *sleep* message. So if a plugin has to be opened in the background again after that, it must send the new (or the same) value of *wakeupNeeded* and *wakeOnEvents* in the *close*, *initEnd*, or *sleep* message.

> **Note:** Constant working of plugins in the background is not advised, as it may negatively affect a device's performance (hence the user experience) and its battery life. However, if it is necessary for your business, then the best practice is to set a higher value for *sleepTimeout* (up to 3600 s (1 hour)). This helps to avoid repeated reopening and closing of the plugin's frame in high-frequency series (which will be the case if the values of *wakeupDelay* and *sleepTimeout* both set to low values and waking up is requested by each *sleep* message).

Available Message Fields for *wakeup*:

- eventName: Name of the *wakeup* event which caused the error (if applicable).
- paramName: Name of the event field which caused error (if applicable).

error_codes_wakeup

You can find the errors codes and their relevent descriptions of the errors that are created by the wakeup method.

Following table describes the various error codes and their descriptions that are created by the wakeup method.

Code	Cause
TYPE_WAKEUP_PARAM	
CODE_WAKEUP_EVENTS_INVALID	"wakeOnEvents" is not a plain object
CODE_WAKEUP_EVENT_NOT_SUPPORTED	"wakeOnEvents" contains a field which key is not one of: "online", "timer"

Code	Cause
CODE_WAKEUP_EVENT_PARAMS_INVALID	"wakeOnEvents" contains a field which in not *null* and is not a plain object
CODE_WAKEUP_EVENT_PARAM_VALUE_INVALID	One of: • "wakeupDelay" field value is not an integer number • "wakeupDelay" field value is less than 10 • "sleepTimeout" field value is not an integer number • "sleepTimeout" field value is less than 10 or greater than 3600

Supported Functions

The in-app camera module provides several useful benefits to the user such as control of flashlight, zoom, autofocus adjustment, retake photo and front/back camera switch. The resulting photo can be controlled by additional parameters: quality, targetWidth and targetHeight. Masking secure plugin parameters helps you prevent unauthorized access to sensitive data, enhance clarity and consistency and ensure compliance with regulations.

The Mobile Plugin Framework now supports a new 'takePhoto' procedure that provides the ability for the user to use their device camera to get a picture into a plugin. This procedure utilizes the in-app-camera that works for 'image' properties within OFS. Masking secure plugin parameters helps you prevent unauthorized access to sensitive data, enhance clarity and consistency and ensure compliance with regulations.

For more details on Camera support and Masking securing plugin parameters, see:

- Mask Secure Plugin Parameters
- Camera support in Mobile Plugin

Camera support in Mobile Plugin Framework

The in-app camera module provides several useful benefits to the user such as control of flashlight, zoom, autofocus adjustment, retake photo and front/back camera switch. The resulting photo can be controlled by additional parameters: quality, targetWidth and targetHeight. These provide the opportunity to control the size of the photo and it's resolution. The resulting photo that is returned by the "takePhoto" procedure is in JPEG format.

The Mobile Plugin Framework supports a new 'takePhoto' procedure that provides the ability for the user to use their device camera to get a picture into a plugin. This procedure utilizes the in-app-camera that works for 'image' properties within Oracle Field Service. This feature is designed for use within the installed iOS and Android applications. Browser support is not available.

- The resulting photo can be controlled by additional parameters: quality, targetWidth and targetHeigh. These provide the opportunity to control the size of the photo and it's resolution.
- The resulting photo that is returned by the "takePhoto" procedure is in JPEG format.
- The resolution of the photo is based on a device's camera module and it is cropped according to the screen aspect ratio. So the final image aspect ratio could differ between portrait and landscape modes. Also, the final image can be cropped via functionality of the in-app camera; in this case the width or height of the photo will be cut.

How To Use

1. Add **"sendMessageAsJsObject": true** item to "ready method"
2. Open the plugin on iOS/Android device (not in browser)
3. Check that the "takePhoto" procedure is available in the list of "allowedProcedures" in "open" method
4. Run the "takePhoto" procedure by using the following code:

```
let data = {
 "apiVersion": 1,
 "method": "callProcedure",
 "callId": "123",
 "procedure": "takePhoto"
}
parent.postMessage(data, document.referrer);
```

Request

The request could be called with or without parameters. These parameters are all optional:

- quality - the percent of JPEG compression
  - min: 1
  - max: 100
  - default: 50
- targetWidth - maximal width of a picture
  - min: 10
  - max: 7000
  - default: no fixed value, depends on resolution of device camera
- targetHeight - maximal height of a picture
  - min: 10
  - max: 7000
  - default: no fixed value, depends on resolution of device camera
- The quality of the picture set to '50' by default. The maximum supported size of the photo is 8Mb. If the photo exceeds this value, the error 'CODE_TAKE_PHOTO_FILE_IS_TOO_LARGE' is returned.
- Please take into account that the size of the picture can influence the device's performance.
- The targetWidth and targetHeight parameters can be used to resize the final picture. The aspect ratio won't be changed, and the picture can be reduced or increased in size in order to fit the target width and target height.

Example of ready method with the "sendMessageAsJsObject" flag (Plugin -> OFS):

```
{
 "apiVersion": 1,
 "method": "ready",
 ...
 "sendMessageAsJsObject": true
}
```

Example of open method with the availability of the "takePhoto" procedure that reflects that the "takePhoto" procedure is available: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "open",
```

```
 ...
 "allowedProcedures": {
 ...
 "takePhoto": true
 }
}
```

Example of the calling of the "takePhoto" procedure: (Plugin -> OFS)

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "callId": "123",
 "procedure": "takePhoto"
}
```

Example of the calling of the "takePhoto" procedure with params: (Plugin -> OFS)

```
{
 "apiVersion": 1,
 "method": "callProcedure",
 "callId": "123",
 "procedure": "takePhoto",
 "params": {
 "quality": 50,
 "targetWidth": 1000,
 "targetHeight": 1000
 }
}
```

Success response

Any success response will contain a "cancelled" field with the boolean value. In the case of where the photo has been taken and was sent to a plugin, the result will contain a "photo" field with a Blob file of the photo.

Example of the "takePhoto" procedure calling result: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123",
 "resultData": {
 "cancelled": false,
 "photo": [Blob]
 }
}
```

Example of the "takePhoto" procedure calling the result in case the camera was closed without a photo: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "callProcedureResult",
 "callId": "123",
 "resultData": {
 "cancelled": true
 }
}
```

Error response

Error responses will include an error type and an error code. The following errors can be returned:

TYPE_INTERNAL	CODE_UNKNOWN	-	Common errors
TYPE_MESSAGE_FORMAT	CODE_METHOD_UNEXPECTED	-	Is returned if the plugin tries to run the "takePhoto" procedure while this procedure is already running. Plugin should wait for a result of this procedure before calling it again.
TYPE_PROCEDURE_ERROR	CODE_PROCEDURE_UNAVAILABLE	procedure	Is returned if the procedure is not available. (i.e. OFS is not open in iOS/Android Mobile Application)
TYPE_PROCEDURE_ERROR	CODE_PROCEDURE_JS_OBJECT_FLAG_REQUIRED	procedure, data	Is returned if the plugin tries to run the "takePhoto" procedure without plugin request to transfer data with the JS object (i.e. didn't send "sendMessageAsJsObject": true on "ready" method). "procedure" = procedure name, "data" = error message.
TYPE_PROCEDURE_ERROR	CODE_PROCEDURE_PARAMS_IS_NOT_OBJECT	procedure	Is returned if the "params" item was sent and it is not an object.
TYPE_PROCEDURE_TAKE_PHOTO_ERROR	CODE_TAKE_PHOTO_FILE_IS_TOO_LARGE	procedure, data	Is returned if the photo file that is returned by the camera is larger than 8Mb (for example if quality parameter is too high). "procedure" = procedure name, "data" = error message.
TYPE_PROCEDURE_TAKE_PHOTO_ERROR	CODE_PROCEDURE_FAILED	procedure, data	Any other error that is not listed here. "procedure" = procedure name, "data" = error message.
TYPE_PROCEDURE_PARAM	CODE_PROCEDURE_PARAM_VALUE_INVALID	procedure, paramName	Is returned if: • quality is out of range (1..100) • targetWidth is out of range (10..7000) • targetHeight is out of range (10..7000)

Example of an error if the procedure "takePhoto" is called from plugin that was opened not from the native Android/iOS App, or the procedure "takePhoto" is called on the OFS platform: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_UNAVAILABLE",
 "procedure": "takePhoto"
 }
 ],
 "callId": "123"
}
```

Example of an error where the procedure "takePhoto" is called with "params" that is not an object: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_PARAMS_IS_NOT_OBJECT",
 "procedure": "takePhoto"
 }
 ],
 "callId": "123"
}
```

Example of an error where the procedure "takePhoto" is called with a "quality", "targetWidth" or "targetHeight" parameter that is not an integer, or if the value is larger or smaller than the minimum or maximum values: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_PARAM",
 "code": "CODE_PROCEDURE_PARAM_VALUE_INVALID",
 "procedure": "takePhoto",
 "paramName": "quality"
 }
 ],
 "callId": "123"
}
```

Example of an error where the procedure "takePhoto" is called and the resulting file is too large: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_TAKE_PHOTO_ERROR",
 "code": "CODE_TAKE_PHOTO_FILE_IS_TOO_LARGE",
 "procedure": "takePhoto",
 "data": "..."
 }
 ],
 "callId": "123"
}
```

Example of an error where the plugin is not requesting transfer with the JS object in "ready" method: (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_ERROR",
 "code": "CODE_PROCEDURE_JS_OBJECT_FLAG_REQUIRED",
 "procedure": "takePhoto",
 "data": "..."
 }
 ],
 "callId": "123"
}
```

Example of an error for any other reason of procedure processing (OFS -> Plugin)

```
{
 "apiVersion": 1,
 "method": "error",
 "errors": [
 {
 "type": "TYPE_PROCEDURE_TAKE_PHOTO_ERROR",
 "code": "CODE_PROCEDURE_FAILED",
 "procedure": "takePhoto",
 "data": "..."
 }
 ],
 "callId": "123"
}
```

Data transfer format (JSON / JS object)

The data between a Plugin and OFS can be transferred in two formats: as a JSON string (initially) and as a JS object (since 23C).

Transferring data as a JS object has a couple benefits that makes it more useful:

- it allows the sending of files from OFS to a Plugin (in the case of the "takePhoto" procedure)
- the data transferring is faster because there is no parsing and validation of the JSON format

It is highly recommended to send a "sendMessageAsJsObject": true item in "ready" method within all modern plugins.

OFS keeps information about the chosen variant of transfer in the browser's memory. So, if a "sendMessageAsJsObject":true flag was sent, OFS will transfer data as a JS object until either the "sendMessageAsJsObject": false is sent or while the browser tab is open.

Example of the "ready" message:

```
{
 "apiVersion": 1,
 "method": "ready",
 "sendMessageAsJsObject" :true
}
```

Mask Secure Plugin Parameters

Masking secure plugin parameters helps you prevent unauthorized access to sensitive data, enhance clarity and consistency and ensure compliance with regulations.

Oracle considers security one of its main priorities, and therefore continuously evolves applications to comply with higher security standards. To this point, this new feature introduces improvements in the configuration of any secure parameters of plugins. With the 23B update, it will be possible to define parameters containing secure data and mask their values, therefore preventing unauthorized access. The feature solves two challenges:

the situation known as "man behind"inadvertent access to values of secure parameters by other people who have privileges to configure the application

How it works

Let's take a typical example of a plugin interacting with some other system via Oracle Integration Cloud. To make it happen, you have to configure the Client ID of the OIC application as one of the plugin parameters.

When adding this new parameter to a plugin, you should check the "Secure parameter" checkbox identifying that the parameter contains sensitive data. When enabling the checkbox, the application will mask a value of the parameter within the UI, replacing it with a series of "dots".

You'll be able to uncheck the checkbox and verify that the value is correct until you save the configuration of the plugin. Once the configuration is saved, the application will mask the value of the parameter on the following screens:

- "Edit plugin"
- "View parameter"

However, you'll still be able to edit the parameter and change its name and value.

> **Note:** When opening previously saved secure parameters for editing, you'll have to replace the values and specify them over again as the system will delete the value from the field. This is needed in order to not confuse users, as they won't be able to edit masked data since the real values cannot be accessed under any conditions.

Avoid Cross-Domain Communication Blocking

To avoid cross-domain communication blocking when an Oracle Fusion Field Service API is called from a plugin:

- For Oracle Fusion Field Service hosted plugins: When calling an Oracle Fusion Field Service API, the plugin must use a plugin hosting domain (available in Java script as a value of window.location.hostname property) instead of <instance_name>.fs.ocs.oraclecloud.com.
- For externally hosted plugins: The plugin must not call an Oracle Fusion Field Service API directly from the browser. Instead, the plugin must call its server side. All the API calls must be performed by the server side and the call results transmitted to the plugin.

3 Use a Plugin

Add a Plugin

A plugin is available out-of-the-box in Oracle Fusion Field Service and is supported by Oracle. It contains the logic that covers specific business scenarios and can support integrations with other Oracle products such as Service Logistics. You cannot change a plugin.

1. Click **Configuration** > **Forms & Plugins**.
   The **Forms & Plugins** page appears and displays the existing forms and plugins.
2. Click **Add Plugin**.
3. Select any of these plugin type and then click **Next**.

Plugin Type	Description
Plugin Archive	An archive plugin is hosted in Oracle Fusion Field Service and uses the Plugin API to interact with Oracle Fusion Field Service. This means, if your plugin consists only of HTML, CSS, and JavaScript files and doesn't contain server-side files, then you can host it in Oracle Fusion Field Service. No other hosting is required. The plugin framework handles the communication between the hosted plugin and Oracle Fusion Field Service.
External Plugin	An external plugin is hosted elsewhere and communicates with Oracle Fusion Field Service through the Plugin API. You add only a link to the plugin here.
External Application	An external application can be added as a plugin and it will be opened as a web page in a new window, or the same window within Oracle Fusion Field Service.
Sample Plugin	There are some sample plugins such as Meter Reading plugin that you can use. You can download the sample plugins from the "www-sites.oracle.com/downloads/samplecode/ofsc-samplecode-downloads.html" page. You may need to use the secure protocol https and a user account to access this site. Unless explicitly identified as such, the sample code available on this page isn't certified or supported by Oracle; it's intended for educational or testing purposes only.

4. For example, click Sample Plugin.
   These sections are displayed:
   - **Properties will be installed.** These are the properties that are automatically installed with the plugin. These properties will be available on the **Configuration** > **Properties** page. If you de-install this plugin in the future, these properties will still remain on the Properties page.
   - **Existing properties will be used.** These are the properties that are required for the plugin and are currently present in Oracle Fusion Field Service.

   > **Note:** If a property has an incorrect configuration (for example, for property type or entity), then you will see a corresponding message. Open the plugin's documentation, find the property requirements, and change the property settings accordingly.

5. Click **Activate** and confirm the installation.

   A message similar to, 'Sample Plugin Successfully activated.' is displayed on the **Forms and Plugins** page after the installation You can install a plugin only once. The **Activate** button is disabled when you try to activate the plugin again. Be aware that plugins are supported only in the English language.
6. To delete the plugin that is not in use:
   a. Click **Configuration** > **Forms & Plugins**.
   b. Locate the plugin that you want to delete.
   c. Click the actions icon and then click **Delete**.

      The plugin is deleted and is no longer displayed on the **Forms and Plugins** page.

Modify the Settings of a Plugin

You can modify the settings of a plugin with some exceptions. You cannot change the required fields and the plugin label. If a plugin is designed to work with some specific properties and parameters, you cannot change them either.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **View** to search for the plugin for which you want to modify the settings.
3. Click the actions icon and then click **Edit**.
4. On the **Edit Plugin** page, change the details as required and click **Update**.

Change the Code of a Plugin

You can change the code for a plugin to suit your business requirements and upload it back as a Hosted plugin.

> **Note:** If you change the code for a plugin, the plugin becomes your custom plugin and it will no longer be supported by Oracle.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **View** to search for the Standard plugin for which you want to change the code.
3. Click the actions icon and then click **Edit**.
4. On the **Edit Plugin** page, click **Download Source** and download the source files to the required folder.
5. Unzip the files and change the code as required.
6. Follow instructions in the README.md and create an archive to upload the plugin back as a Hosted plugin.
7. Follow the instructions in the Add a Hosted Plugin topic and add the modified plugin as a new hosted plugin.
8. Add a button for the plugin on the required page.
9. Open the plugin and test your scenarios.

Install the Sample Plugin

The Sample plugin is available and accessible out-of-the-box within Oracle Fusion Field Service. You can install and use it as a sandbox for testing, checking requests, and checking features such as the Barcode scanner, Print file, and Service Worker.

1. Use the procedure described in the Add a Plugin topic and install the plugin.
2. Verify the available features of the plugin.
3. Add the required settings to the plugin (properties, secure parameters, and so on).
4. Add a button for the plugin on the required page.
5. Open the plugin and test your business scenarios.
6. Learn the source code:
    a. Go to the Plugins page and click Sample plugin.
    b. Click **Download Source**.
    c. Verify the implementation (for example, how to implement the Service Worker to support in offline mode).
    d. Update the code as required.
    e. Follow the instructions in the README.md file to create an archive to upload it back as a Hosted plugin for testing.
    f. Upload the plugin either using the **Plugin archive** section on the **Edit Plugin** page, or through a REST API.
    g. Add a button for the plugin on the required page.
    h. Open the plugin and test your business scenarios.
7. Deinstall the plugin if you don't need it for testing purposes.

Metadata API for Plugin Installation

The metadata API for plugin installation helps in faster, seamless implementation of the plugins. Automate the testing and development process by minimizing manual configurations of plugins.

You can use the Metadata API method to automate the installation of plugins such as "Debrief" and Asset View" as part of your business process. Also, Fusion Service configuration via Configuration | Applications page now supports the automatic installation of the Debrief plugin.

A new Metadata API request is available to install standard plugins in OFS.

Request

POST rest/ofscMetadata/v1/plugins/{pluginLabel}/custom-actions/install

URL data params:

- **pluginLabel** [Required] - String containing the plugin which is to be installed

This operation installs the plugins as defined in the path parameter. If a plugin with the given label is already in place, then it will return "Plugin already existing" error message; otherwise the plugin will get installed successfully.

Response

204 Response - This response code 204 indicates that the operation completed successfully. This operation does not return any elements in the response body.

Use Case	Status	Response
Standard plugin installation is successful	204	
No permission	403	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.4", "title": "Forbidden", "status": "403", "detail": "Authentication was provided, but the authenticated user is not permitted to perform the requested operation." }
No standard plugin exists for the provided pluginLabel	404	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.5", "title": "Not Found", "status": 404, "description": "Plugin {LABEL} not found",}

Validation Errors

Error Codes
LABEL_NOT_UNIQUE
OLDER_PRODUCT_VERSION
EXISTING_PROPERTY_TYPE_MISMATCH
INVALID_PROPERTY

Use Case	Status	Response
Plugin is already installed	409	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Plugin's label is already in use", "status": 409, "description": "A plugin with label "{PLUGIN_LABEL}" already exists. Remove it to proceed", "o:errorCode": "LABEL_NOT_UNIQUE"}
Platform version is less than the required version	409	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Plugin cannot be installed", "status": 409, "description": "Field Service version must be 22.08.0 or higher in order to install the plugin", "o:errorCode": "OLDER_PRODUCT_VERSION"}
Existing property type mismatch	409	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property type mismatch", "status": 409, "description": "A property {PROPERTY_LABEL} has type {CURRENT_TYPE} but the plugin requires it to be with type

Use Case	Status	Response
		{EXPECTED_TYPE}", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"} { "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property gui mismatch", "status": "409", "detail": "A property {PROPERTY_LABEL} has GUI {CURRENT_GUI} but the plugin requires it to be with type {EXPECTED_GUI}", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"} { "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property entity mismatch", "status": "409", "detail": "A property {PROPERTY_LABEL} is created for entity {CURRENT_ENTITY} but the plugin requires it to be created for entity {EXPECTED_ENTITY}", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"} { "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property mime_types mismatch", "status": 409, "description": "A property {PROPERTY_LABEL} has MIME types {CURRENT_CONFIGURED_MIME_TYPE} but the plugin requires it to be with types {EXPECTED_MIME_TYPE}", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"} { "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property mime_types mismatch", "status": 409, "description": "MIME types value of property {PROPERTY_LABEL} must be in array format", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"} { "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.10", "title": "Existing property mime_types mismatch", "status": 409, "description": "The MIME types is required for property {PROPERTY_LABEL}. OFS is configured with the following: {EXPECTED_MIME_TYPES}", "o:errorCode": "EXISTING_PROPERTY_TYPE_MISMATCH"}

Modify Property

For resolving the "EXISTING_PROPERTY_TYPE_MISMATCH" validation error, in the Configuration | Properties screen, the problematic property's type can be changed to the required property type shown in the error response. Or, the property could also be deleted and the plugin installation would automatically install the property with the needed values.

Other unexpected errors

Use Case	Status	Response
Unable to get plugin data from the Plugin Repository service (This occurs when the plugin repository service is unavailable)	500	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1", "title": "Internal Server Error", "status": "500", "detail": "Unable to install plugin {pluginLabel}"}
Unable to create property {propertyLabel} of plugin {pluginName} from Plugin Repository service (This occurs when the property creating throws an exception. This should not occur in normal cases)	500	{ "type": "http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.5.1", "title": "Internal Server Error", "status": "500", "detail": "Unable to create property {propertyLabel} of plugin {pluginName}"}

Debrief Plugin

Debriefing involves documenting the time and materials used during an activity. Mobile workers can click the Debrief button on an ongoing activity to record time, expenses, or materials information for an invoice report. They can save this information and obtain the customer's signature.

A mobile worker uses the debriefing process to report this information:

- Labor: Includes travel time and working time (measured in hours)
- Parts: Parts and materials used while performing the activity
- Charges: Any extra charges such as tolls or parking (measured in money spent)

All parts, labor, and expense items are stored in the installed inventory pool of the corresponding customer activity. The invoice is saved as a PDF file to the file property of the activity.

Configure the Debrief Plugin

Here are the high-level steps to configure the Debrief plugin. You must follow the order given here strictly to configure the plugin.

1. Activate the plugin.
2. Create or configure the required inventory types and user types.
3. Add the URL of the company logo that must displayed on the invoice.
4. Upload the Parts Catalog, if you don't have one.

Activate the Debrief plugin

Debrief is available as a part of the Screens-> Activity in Oracle Fusion Field Service. Debriefing require additional set up before mobile workers can use it.

You can add additional secure parameters to the plugin, but you can't change the label, or available properties.

1. On the User Types page, navigate to **Screens-> Activity->Debriefing**.
   The **Debriefing Activate Screen** page appears and these sections are displayed:
   - **Properties to be added.** These are the properties that are automatically installed with the plugin. These properties will be available on the **Configuration** > **Properties** page. If you de-install this plugin in the future, these properties will still remain on the Properties page.
   - **Existing properties to be used.** These are the properties that are required for the plugin and are currently present in Oracle Fusion Field Service.

   > **Note:** If a property has an incorrect configuration (for example, for property type or entity), then you'll see a corresponding message. Open the plugin's documentation, find the property requirements, and change the property settings accordingly.

2. Click **Activate** and confirm the activation.
   A message similar to, 'Debriefing Successfully activated.' is displayed after the activation. You can activate the Debrief plugin only once. The **Activate** button is not available after the activation. Be aware that plugins are supported only in the English language. Further, the Debrief plugin doesn't work in offline mode.

> **Note:** You can download the plugin source by clicking **Download as Plugin Source**.

Oracle Fusion Field Service creates the required properties automatically or notifies you that some existing properties will be used by the plugin, if they're already configured. If you've created the properties in the application with the corresponding names and labels, but with the improper configuration, you must change the property settings and activate the plugin again. Here are the properties for resource, activity, and inventory entities that the plugin uses:

Resource entity properties

Name	Label	Type	GUI	Description
ID	pid			Internal ID of the resource.
Name	pname			Name of the resource.

Activity entity properties

Name	Label	Type	GUI	Description
Invoice	invoice	File	File	PDF file of the generated invoice. For

Name	Label	Type	GUI	Description
				example, mime_types ="application/pdf"
Company name	ccompany	String	Text	Customer's company name, displayed as the title of the invoice.
Activity ID	aid			Internal ID of the activity.
Name	cname			Name of activity used in the PDF invoice.
Address	caddress			Activity address used in the invoice.
City	ccity			Activity city used in the invoice.
State	cstate			Activity state used in the invoice.
ZIP/Postal Code	czip			Postal code used in the invoice.
Work Order	appt_number			Work order used in the invoice.
Signature	csign			Customer signature, required prior to saving the invoice as PDF.

Inventory entity properties

Name	Label	Type	GUI	Description
Expense	expense_amount	String	Text	Amount of expense.
Expense Currency	expense_currency_code	Enumeration	Combobox	Value of each enumeration item is separated with the "|" character.

Name	Label	Type	GUI	Description	
				Index	**Value**
				USD	$|US Dollars
				EUR	€|Euro
Part Disposition	part_disposition_code	Enumeration	Combobox	The value that identifies whether the inventory is consumable by the customer and there is no need to track it anymore, or whether the inventory is returnable. If the inventory is returnable, the Inventory Management system of Oracle SCM Cloud must track the part until it is returned by the customer.	
Part Unit of Measure	part_uom_code	Enumeration	Combobox	The unit of measurement (UOM) of parts (inventories).	
Part Item Description	part_item_desc	String	Text	The description of the part. For example, 'Magnetic hard drive'. It is used to search for inventory in the catalog.	
Part Item Number	part_item_number	String	Text	The number of the part that has been installed or taken from the customer. It is specified as a code. For example, FS908765.	
Part Item Revision	part_item_revision	String	Text	A single-letter code, for example, "A" or "B". Also, it is possible to have a single digit like "1" or "2". Usually, the inventory is identified by Part Item + Part Item Revision, but Item Revision is optional.	
Part Item Number + Revision	part_item_number_rev	String	Text	The Part Item number concatenated with the Part Item Revision. For example, FS908765A, where "FS908765" is a Part Item Number and "A" is a Part Item Revision. It is used to search for inventory in the catalog.	
Expense Activity	expense_service_activity	Enumeration	Combobox	Type of expense.	
Expense Item	expense_item_number	Enumeration	Combobox	The subtype of expense.	

Name	Label	Type	GUI	Description
Expense Item Description	expense_item_desc	Enumeration	Combobox	The description of expense subtypes. The indices must be the same as in the expense_item_number property. The values must describe the corresponding expense_item_number element.
Labor End Time	labor_end_time	String	Text	The time when a mobile worker stops working on particular service activity. It must be not later than the end time of the work order (Oracle Fusion Field Service activity). Also, there must be no overlap between the items in the labor list. The format is T24:59:59.
Labor Start Time	labor_start_time	String	Text	The time when a mobile worker starts working on a particular service activity. It must be not be earlier than the start time of the work order (Oracle Fusion Field Service activity). Also, there must be no overlap between the items in the labor list. The format is T24:59:59.
Labor Activity	labor_service_activity	Enumeration	Combobox	The type of labor.
Labor Item	labor_item_number	Enumeration	Combobox	The subtype of labor.
Labor Item Description	labor_item_desc	Enumeration	Combobox	The description of labor subtype. The indices must be the same as in labor_item_number property. The values must describe the corresponding labor_item_number element.
Labour Hours	laborItemNumberForR	Enumeration	Combobox	This is updated with the billing item under which the Labor hours can be tracked.
FS Travel Time	laborItemNumberForTr	Enumeration	Combobox	This is updated with the billing item under which the Travel Time can be tracked
Inventory ID	invid			Internal ID of the inventory.
Activity ID	inv_aid			Internal ID of the activity to which the inventory is assigned.
Resource ID	inv_pid			Internal ID of the resource to which the inventory is assigned.

Name	Label	Type	GUI	Description
Inventory Pool	invpool			The inventory pool (Resource, Customer, Installed, De- installed).
Inventory Type	invtype			Type of inventory. See Add Inventory Types for the Plug-In.
Quantity	quantity			The installed parts or the parts taken from the customer. It can be either counted or specified in inches, feet, and so on. The quantity is defined as an integer number.
Serial Number	invsn	Field	Text	The serial number of the inventory.

3. To add your company logo in the Time & Labor Report, add a new secure parameter with the name "logoUrl" with the value "url of the company logo".

   Logo only supports .jpeg images and the recommended size of the image is less than "150X60 ".

Add the Inventory Types

You must add inventory types (expense, labor, part, part_sn) to capture the information about the time and materials used for the activity. The plugin stores the reported information about time and expense in the installed pool of the activity. However the parts used and parts returned are stored in the resource and customer pools respectively.

1. Log in to Oracle Fusion Field Service as an administrator.
2. Click **Configuration** > **Inventory Types** and click **Add New**.
3. To add the 'Expense' inventory type:
   a. Type 'expense' in the Label field.
   b. Type 'Expense' in the Name field.
   c. Select 'Expense_item' from the Model Property drop-down list.
   d. Click **Add**.
4. To add the 'Labor' inventory type:
   a. Type 'labor' in the Label field.
   b. Type 'Labor' in the Name field.
   c. Select 'Labor_Item' from the Model Property drop-down list.
   d. Click **Add**.
5. To add the 'Part' inventory type:
   a. Type 'part' in the Label field.
   b. Type 'Part' in the Name field.
   c. Select 'Part_item+Revision' from the Model Property drop-down list.
   d. Click **Add**.

6. To add the 'Part SN' inventory type:
    a. Type 'part_sn' in the Label field.
    b. Type 'Serialized Part' in the Name field.
    c. Select 'Part_item+Revision' from the Model Property drop-down list.
    d. Click **Add**.

Add the Parts Catalog

You can search for the parts used or returned from the catalog and then add these parts to an invoice. To view the parts from the catalog, you must create the catalog using the create_catalog method of the SOAP API or the REST API.

1. Use this code sample to understand how to create a catalog for Debrief:

```
{
 "name": "my_catalog",
 "fieldSchemas": [
 {
 "label": "part_disposition_code",
 "name": "Part Disposition Code",
 "searchable": true,
 "preview": false
 },
 {
 "label": "part_item_number",
 "name": "Item Number",
 "searchable": true,
 "preview": true
 },
 {
 "label": "part_item_revision",
 "name": "Item Revision",
 "searchable": true
 },
 {
 "label": "part_item_desc",
 "name": "Item Description",
 "preview": true
 },
 {
 "label": "part_uom_code",
 "name": "UOM",
 "preview": true,
 "searchable": true
 }
 ],
 "typeSchemas": [
 {
 "itemType": "part",
 "inventoryType": "part_general"
 },
 {
 "itemType": "cartridge",
 "inventoryType": "part_cartridge"
 }
 ]
}
```

2. Use this code sample to understand how to create or update parts catalog items for Debrief scenarios:

```
"type": "part",
 "fields": [
 {
 "label": "part_disposition_code",
 "value": "ECM100001A"
 },
 {
 "label": "part_item_number",
 "value": "ECM100000"
 },
 {
 "label": "part_item_revision",
 "value": "ECM100000A"
 },
 {
 "label": "part_item_desc",
 "value": "2" x 5" Robotically Welded Steel Frame"
 },
 {
 "label": "part_uom_code",
 "value": "ea"
 }

 ],
 "tags": [
 "Printer",
 "Cartridge"
 ],
 "linkedItems": [
 {
 "itemLabel": "RG5-7691-250CN",
 "data": "1"
 },
 {
 "itemLabel": "RG5-7691-250CF",
 "data": "2"
 },
 {
 "itemLabel": "RG5-7691-250CZ",
 "data": "3"
 }
 ],
 "images": [
 {
 "imageURL": "https://www.storage-service.com/rg5_7691_250cz.png"
 },
 {
 "imageURL": "https://www.storage-service.com/rg5_7691_250cf.png"
 }
 ]
}
```

3. Double check the value for the *type* field as follows:
    - The 'type' field within the 'create' or 'update' parts catalog calls must be set as "part" for serialized inventory.
    - The 'type' field within the 'create' or 'update' parts catalog calls must be set as "part_sn" for non-serialized inventory.
    - Each item's 'Fields' schemas must contain these elements:

Label	Property Label	Searchable
part_uom_code	part_uom_code	0
part_item_revision	part_item_revision	0
part_item_number	part_item_number	0
part_item_desc	part_item_desc	1
part_disposition_code	part_disposition_code	0

Add the Debrief Plugin to a Page

To make the plugin available to multiple user types, you must associate it with the corresponding pages for the required user types. As debriefing is done only for Started activities, you must add the Debrief button to the **Activity Details** page so that it is visible only when the activities are in that status.

1. To add the Debrief button to a page, follow the instructions in the *Add a plugin to a page* topic.
2. Be sure to add the visibility condition as `Activity status in (equal) Started`.

Asset Details Plugin

The "Asset Details" standard plugin provides comprehensive asset information for mobile resources and dispatchers, to carry out their work more effectively. It includes asset pictures, maintenance orders, warranty details, asset notes, and history. Seamlessly integrated with Oracle Supply Chain, it requires no additional development effort. Customers can tailor it using the Plugin Framework and integrate it with third-party supply chain solutions if required.

1. Open the Asset Details page to view the basic information of the asset.

Field	Description
Asset ID	Value that uniquely identifies the asset
Asset Description	Description of the asset features, characteristics, and other details.
Item ID	Value that uniquely identifies the item referenced by the asset.

Field	Description
Serial Number	Numeric value that uniquely identifies the serial number referenced by the asset.
Primary Contact	Name of the contact person associated with the asset.
Address	Name of the location where the asset currently resides.
Image	The item image. (You can view the full image by clicking the image; if there are multiple images, you can click Next to view the other images.)

2. To view the future work orders associated with the asset, click **View Details** and then **Future Work Orders**.

Field	Description
Work Order Number	The alternate unique identifier of the work order.
Scheduled On	The date when the service is scheduled to be performed.
Mobile Worker	The scheduled resource assigned to complete the work order.
Problem Description	The information captured to share with the field resource assigned to the work order.

The page displays up to 15 work orders. Mobile workers can use OFS capabilities such as self-assignment to prepare for upcoming tasks while on site.

3. To view the warranty, active subscriptions, and service level coverage details associated with the asset, click **View Details** and then **Warranty and Subscriptions**.

Field	Description
Coverage Name	The name of the coverage.
Coverage Description	The description of the coverage.
Subscription Number	The alternate unique identifier of the subscription.
Coverage Product Name	The name of the product associated with the coverage.
Service Activity Name	The name of the service activity.
Adjustment Type	The code associated with the adjustment type. The type can be either Markup or Discount.
Adjustment Amount	The value of the adjustment. Adjustment amount can be percentage or a Fixed value.

4. You can view the latest 15 notes associated with an asset in the **Notes** section.

Field	Description
Created Date	Date when the note was created.
Created Time	Time when the note was created.
Author	Name of the user who created the note for the asset.
Note text	Decoded note text.

5. You can view past records of up to 15 work orders in the **History** section.

Field	Description
Work Order	The alternate unique identifier of the work order.
Completed On	The date when the service was performed.
Mobile Worker	The scheduled resource assigned to complete the work order.
Problem Description	The additional information captured to share with the field service resource assigned to the work order.
Resolution	The information added to the work order from Oracle Field Service Cloud activity integration.

Activate the Asset Details Plugin

Asset Details is available as a Standard plugin in Oracle Fusion Field Service. You can install it from the **User Types & Screens** page.

1. On the User Types page, navigate to **Screens-> Inventory->Asset Details**.
   The **Asset Details Activate Screen** page appears and these sections are displayed:
   - **Properties will be added**. These are the properties that are automatically installed with the plugin. These properties will be available on the Configuration > Properties page. If you de-install this plugin in the future, these properties will still remain on the Properties page. Make sure you configure Note Type as a plugin parameter. Note type is the comma separated note type codes of asset notes. You can access Note type codes from the Fusion Manage Contract Note Types.
   - **Existing properties to be used**. These are the properties that are required for the plugin and are currently present in Oracle Fusion Field Service.
   - **Applications**.

     The plugin requires connection to REST API. Select the following applications:
     - Fusion application configured in the OFS Applications screen to authenticate the plugin.
     - Oracle Fusion Field Service application configured in the OFS Applications screen to authenticate the plugin.
2. Click **Activate** and confirm the activation.

   A message similar to "Asset Details Successfully activated" is displayed after the activation.

   The following **Activity entity** properties are installed and used as part of Asset Details plugin.

Name	Label	Type	GUI	Description
Asset Id	wo_asset_id	string	text	The asset id of install base asset associated with the activity. This custom property will be populated when the work order sync happens. See, Work Order Created Field Mappings

Name	Label	Type	GUI	Description
				R13 (1.0) in the Integrating Fusion Service with Field Service guide.
Work Order Number	wo_number	string	text	The Fusion work order number associated with the activity. This custom property will be populated when the work order sync happens. See, Work Order Created Field Mappings R13 (1.0) in the Integrating Fusion Service with Field Service guide.
Date	date	field	text	Date when the activity was assigned to route. Based on the activity date, you can view the history, warranty information, future work orders, and past work orders.

Modify the Parameters of Asset Details Plugin

You can modify the following parameters of Asset Details plugin. You cannot change the mandatory fields and the plugin label.

1. Click **Configuration** > **User Types > Screens**.
2. Select the **Asset Details** plugin.
3. On the **Asset Details Edit Screen** page, modify the following parameters as required and click **Update**.

Parameter	Description	Secure Parameter
fusionRestEndpoint	The Fusion end point URL.	N
fusionLogin	Fusion Login with roles mentioned as in the "APIs and Roles" section.	N
fusionPassword	Fusion Password.	Y
noteTypeCodes	Comma separated note type codes of asset notes, that can be shown to the mobile worker. If not configured, the Notes section will show the message "Please configure noteTypeCodes in Plugin Parameters."	N

Parameter	Description	Secure Parameter
plannedActivityMaxDays	If checked, parameter's value will be masked on the screen after saving. It will be possible to replace the value or delete the parameter.	N
nearbyRadius	If checked, parameter's value will be masked on the screen after saving. It will be possible to replace the value or delete the parameter.	N
nearbyRadiusUnit	If checked, parameter's value will be masked on the screen after saving. It will be possible to replace the value or delete the parameter.	N

Extend the Asset Details Plugin

You can extend Asset Details plugin for your organization specific requirements and upload it back as a hosted plugin. If you change the code for Asset Details plugin, it becomes your custom plugin and will no longer be supported by Oracle. See, *Change the Code of a Plugin* .

Asset Details Plugin APIs and Roles

The Fusion user configured in the fusionLogin secure parameter should have the below privileges to get the data in Asset Details plugin.

API Reference	Role
Installed Base Assets REST Endpoints	*Maintenance Technician (Job Role)*
Work Orders REST Endpoints	*Customer Service Representative (Job Role)*
Entitlements REST Endpoints	*Customer Service Representative (Job Role)*
Product Image Attachments REST Endpoints	*Employee (Abstract Role)*

Order and Receive Parts using Parts Ordering Plugin

Parts Ordering is a standard plugin that comes with Oracle Fusion Field Service; it automates the process of ordering and receiving parts. You can use the plugin to order parts when the parts you need aren't available with nearby mobile workers, warehouses, or in your van.

You can see the **Parts Ordered** tile on the **My Route** page only if your administrator has added it. For more information, refer to **Add the Parts Cart and Ordered Parts Buttons**.

You can order for a part to perform a follow up activity for a customer, or to replenish the stock in your van. When you order a part, the plugin creates a not-ordered activity in Oracle Fusion Field Service and an order in Oracle Supply Chain & Manufacturing. Currently you can order parts only for yourself.

Here's a general outline of the process:

- **Install and Activate the Plug-In**: Ensure the Parts Ordering Plug-In is installed and activated in your system. For more information, see Activate the Parts Ordering Plugin.
- **Navigate to the Activity**: Log in as a technician or user with the appropriate permissions and locate the activity where you want to add parts. For more information, see Add the Activity Type Required for the Parts Ordered Plugin.
- **Order Parts**: Use the plugin to order the required parts. For more information, see Order a Part topic in the Using Core Application Guide.
- **Add Parts to the Activity**: Once the parts are ordered and received, you can associate them with the specific activity. For more information, see Receive an Ordered Part in the Using Core Application Guide.

Activate the Parts Ordering Plugin

Parts Ordering is a standard plugin that comes with Oracle Fusion Field Service; it automates the process of ordering and receiving parts. Mobile workers can use the plugin to order parts when the parts they need aren't available with nearby mobile workers, warehouses, or in their van.

High-level steps to activate the Parts Ordering plugin:

1. Create an application for Oracle Fusion Field Service REST API and add the required entities.
2. Install the plugin.
3. Add the activity and inventory types.
4. Add the buttons for the plugin on the pages that mobile workers need.
5. Register a parts catalog. See the **Register an Item Catalog** topic in the Administering Oracle Fusion Field Service guide and register the parts catalog that you want to use.
6. Optionally, integrate with Workflow Manager.

To activate the Parts Ordering plugin:

1. Create an application to connect to Oracle Fusion Field Service REST API from the **Configuration** > **Applications** page. This application is used to connect Oracle Fusion Field Service with Oracle Fusion.
    a. Select OFSC in the **Token Service** field and Authenticate using Client ID/Client Secret in the **Authentication settings** field.
    b. Provide ReadWrite access for Activity and Inventory entities in the **Core API** section.

2. On the User Types page, navigate to **Screens-> Inventory->Parts Ordering**.
   The **Parts Ordering Activate Screen** page appears and these sections are displayed:
   - Properties to be installed: These are the properties that are automatically installed with the plugin. These properties are available on the **Configuration** > **Properties** page. If you de-install this plugin in the future, these properties will remain on the **Properties** page.
   - Existing properties to be used: These are the properties that are required for the plugin and are currently present in Oracle Fusion Field Service.

   > **Note:** If a property has an incorrect configuration (for example, for property type or entity), then you'll see a corresponding message. Open the plugin's documentation, find the property requirements, and change the property settings accordingly.

3. Select the application that you created in Step 1.
4. Click **Activate** and confirm the activation.
   A message similar to, 'Parts Ordering Successfully activated.' is displayed after the activation. You can activate Parts Ordering plugin only once and the **Activate** button is activate button is not available. The plugins are supported only in the English language and the Parts Ordering plugin doesn't work in offline mode.
   Oracle Fusion Field Service creates the required properties automatically or notifies you that some existing properties will be used by the plugin, if they're already configured. If you've created the properties in the application with the corresponding names and labels, but with an improper configuration, you must change the property settings and install the plugin again. Here are the properties for activity and inventory entities that the plugin uses:

Name	Label	Type	GUI	Description
Resource entity				
External ID	external_id	field	text	External id of resource
Order Warehouse List	order_warehouse_list	string	text	Order Warehouse List
Activity entity				
Address	caddress	field	text	Address field
City	ccity	field	text	City field
ZIP/Postal Code	czip	field	text	ZIP/Postal Code field
State	cstate	field	text	State filed
Activity status	astatus	field	text	Activity status field
Activity type	aworktype	field	combobox	Type of activity
Work Order	appt_number	field	text	Activity Work order filed
SLA End	sla_window_end	field	text	End of SLA window
Order Date	order_date	string	text	Date on which the requester at the destination wants the Order to be delivered at

Name	Label	Type	GUI	Description
				the destination. That is this is the Need By Date. This should include both date and timestamp.
Order Number	order_number	string	text	Activity ID of Receive Order activity
Order Items	order_items	string	text	Is filled by JSON array in Receive Order activity. Plugin uses this property for internal purposes.
Order Destination Type	order_destination_type	string	text	This specifies the destination type to which the Order is to be received. Current values are 'activity' and 'van'
Order Received Items	order_received_items	string	text	
Order Status	order_status	string	text	Duplicates Order Item Status inventory property in Receive Order activity
Order Arrival Date	order_arrival_date	string	text	Can be updated by external integration script. Represents approximate available date to receive ordered parts.
Order Follow-up Activity ID	order_followup_aid	string	text	Is filled by follow-up activity Id in Receive Order activity. If Order is created with follow-up activity.
Order Follow-up Activity Number	order_followup_ apptnumber	string	text	Is filled by follow-up activity Work order in Receive Order activity. If Order is created with follow-up activity.
Order Initial Activity ID	order_initial_aid	string	text	Is filled by id of activity which order was created for in Receive Order activity. If option 'Activity' is set in 'Order is for' plugin field.
Order Initial Activity Number	order_initial_apptnumber	string	text	Is filled by appt_number of activity which order was created for in Receive

Name	Label	Type	GUI	Description
				Order activity. If option 'Activity' is set in 'Order is for' plugin field.
Order Initial Activity Type	order_initial_type	string	text	Is filled by aworktype of activity which order was created for in Receive Order activity. If option 'Activity' is set in 'Order is for' plugin field.
Order Initial Activity Address	order_initial_address	string	text	Is filled by concatenation of Address, City, State, Zip/Postal code fields of activity which order was created for in Receive Order activity. If option 'Activity' is set in 'Order is for' plugin field.
Transfer Order Header	transfer_order_header_id	string	string	Can be updated by external integration script. Stores the value of Transfer Order Header ID filed
Inventory entity				
Serial Number	invsn	field	text	The serial number of inventory
Inventory pool	invpool	field	text	Inventory pool (Resource, Customer, Installed, Deinstalled)
Inventory Type	invtype	field	combobox	Type of inventory
Inventory Id	invid	field	text	Internal id of inventory
Activity Id	inv_aid	field	text	Internal id of activity the inventory belongs to
Resource Id	inv_pid	field	text	Internal id of resource the inventory belongs to
Quantity	quantity	field	text	Describes how many parts have been installed or taken from the customer. It could be either counted or specified in "inches", "feets" etc. Quantity is defined as an integer

Name	Label	Type	GUI	Description
				number, fractions are unavailable.
Order Item Initial Activity ID	order_item_initial_aid	string	text	Contains Initial activity ID value when Ordered part is created
Order Item Status	order_item_status	string	text	Represents current status of ordered parts in SCM. Is updated by external integration script. Can be found in Transfer order line payload as 'FulfillmentStatusLookup'. Default value is 'WAIT_FULFILL' for parts ordered by the plugin. Possible values are 'WAIT_FULFILL', 'SHIPPED', 'SHIPPED_RECEIVED', 'RECEIVED', 'P_SHIPPED_RECEIVED', 'P_SHIPPED'. Technician can receive parts if value is 'SHIPPED_RECEIVED'.
Order Item Shipment Number	order_item_shipment_number	string	text	Is updated by external integration script by value of shipment number when it submitted in SCM.
Order Item Received Quantity	order_item_received_quantity	string	text	Describes how many parts have been Received. It could be either counted or specified in "inches", "feets" etc. Quantity is defined as an integer number, fractions are unavailable.
Part Item + Revision	part_item_number_rev	string	text	*Part Item Number* concatenated with *Part Item Revision.*For example "FS908765A", where "FS908765" is a *Part Item Number* and "A" is a *Part Item Revision.* It's used to search for inventory.

Name	Label	Type	GUI	Description
Part Item Number	part_item_number	string	text	Defines the part number of the part that has been installed or taken from the customer. It's usually specified in the form of some code. For example: "FS908765".
Part Item Revision	part_item_revision	string	text	Usually a single letter code, e.g. "A" or "B". Also it is possible to have a single digit like "1" or "2". Usually, the inventory is fully identified by Part Item + Part Item Revision, but Item Revision is optional.
Order Item Details	order_item_details	string	text	Details of ordered item
Order Item Description	order_item_description	string	text	Description of ordered item

If mobile workers see an error and are unable to order for a part, the reason could be:

- A configuration item required for the plugin is missing in the environment.
- The parts catalog structure doesn't contain a required field.
- A required permission in the configuration of the activity or inventory types is missing.

Verify these thoroughly before adding the buttons for the plugin to any page.

Add the Activity Type Required for the Parts Ordered Plugin

You must add a new activity type ORD to capture the details of the orders placed using the Parts Ordered plugin.

1. Sign in to Oracle Fusion Field Service as an administrator.
2. Click **Configuration** > **Activity Types** > **Add New**.
3. Add the label as ORD.
4. Select the **Active** check box.
5. Add a name for the activity type in the English language and other required languages.
6. Select Customer in the **Group** field.
7. Enter 60 in the **Default duration, minutes** field.
8. Select these features: **Support of not-ordered activities**, **Allow non-scheduled**, **Support of inventory**, **Support of required inventory**, and **Allow to create from Incoming interface**.
9. Click **Add**.

Add the Inventory Types Required for the Parts Ordered Plugin

You must add three inventory types part, ordered_part, and received_part to capture the details of the orders placed using the Parts Ordered plugin.

1. Sign in to Oracle Fusion Field Service as an administrator.
2. Click **Configuration** > **Inventory Types** > **Add New**.
3. Complete the fields for each inventory type as given in this table:

Label	Active	Model Property	Name	Inventory is Non-serialized	Unit of Measurement
part	Selected	Part Item Number	Part	Selected	Each
ordered_part	Selected	Part Item Number	Ordered part	Selected	Each
received_part	Selected	Part Item Number	Received part	Selected	Each

4. Click **Add**.

Add the Parts Cart and Ordered Parts Buttons

To make the plugin available to mobile workers, you must associate it with the required pages. As only mobile workers can order for parts, you must add the **Parts Cart** and **Ordered Parts** buttons to the plugin on the **Activity List/My Route**, **Parts details**, and **Activity Details** pages.

To add the buttons, follow the instructions in the *Add a plugin to a page* topic and the details given in this table:

Page	Button Name	Screen Type	Plug-In	Parameter
Activity List/My Route	Part Cart	Plugins	Parts Ordering	Parameter: defaultScreen Value: new-order-screen
	Ordered Parts	Plugins	Parts Ordering	Parameter: defaultScreen Value: order-list-screen
Parts details	Order Part	Plugins	Parts Ordering	Parameter: defaultScreen Value: new-order-screen
Activity Details	Ordered Parts	Plugins	Parts Ordering	Parameter: defaultScreen Value: order-list-screen

Integrations for Parts Ordering Plugin

You can integrate the Parts Ordering plugin with Oracle Fusion Field Service Workflow Management and Oracle Supply Chain Management.

Workflow Management: You might add the plugin to the Workflow Manager. Remember to configure the Workflow steps using the same rules for name naming conventions and parameters as that of User Types.

Oracle Supply Chain Management: You can install the Service Logistics accelerator in Oracle Integration and configure it to get the 'activityCreated' and 'activityUpdated' events. Use 'filterExpressions' in the event subscription to get events for only the 'Order' type of activity.

4 Use a Custom Plugin

Types of Plugins

You can add a hosted plugin, an external plugin, or an external application as a plugin. You can also download and use sample plugins that are shipped with Oracle Fusion Field Service.

Oracle Fusion Field Service supports the following type of plugins:

- **PlugIn Archive:** An archive plugin is hosted in Oracle Fusion Field Service and uses the Plugin API to interact with Oracle Fusion Field Service. This means, if your plugin consists only of HTML, CSS, and JavaScript files and doesn't contain server-side files, then you can host it in Oracle Fusion Field Service. No other hosting is required. The plugin framework handles the communication between the hosted plugin and Oracle Fusion Field Service.
- **External PlugIn:** An external plugin is hosted elsewhere and communicates with Oracle Fusion Field Service through the Plugin API. You add only a link to the plugin here.
- **External Application:** An external application can be added as a plugin and it will be opened as a web page in a new window, or the same window within Oracle Fusion Field Service.
- **Sample Plugin**: There are some sample plugins such as Meter Reading plugin that you can use. You can download the sample plugins from the *https://www.oracle.com/downloads/samplecode/ofsc-samplecode-downloads.html* page. You may need to use the secure protocol https and a user account to access this site. Unless explicitly identified as such, the sample code available on this page isn't certified or supported by Oracle; it's intended for educational or testing purposes only.

Migration of Standard Plugins

Standard Plugins are integrated into the main structure of the application. They can now be found under **User types → Screen** in the following sections:

- **Asset Details, Debriefing** - located in the Activity section
- **Parts Ordering** - located in the Inventory section

This change means that the concept of Standard Plugins has been discontinued and is replaced by a set of default screens, but you can add more by creating forms or plugins.

> **Note:** Asset Details, Debriefing, and Parts Ordering require additional setup before mobile workers can use them. For more information, refer to the activation steps mentioned in their sections.

Add a Plugin Archive

A plugin archive is hosted in Oracle Fusion Field Service and uses the Plugin API to interact with it. The hosted plugin can contain multiple pages and have its own navigation flow. This topic describes the high-level steps to configure and host a plugin.

1. Determine whether you want to host the plugin in Oracle Fusion Field Service. If yes, prepare your plugin for upload.
2. Configure the plugin.
    a. Upload the hosted plugin.
    b. Add the available properties.
3. Add the plugin to the required page.

How Plug-Ins are Hosted

If your plugin consists only of HTML, CSS, and JavaScript files and doesn't contain server-side files, then you can upload it in Oracle Fusion Field Service. No additional hosting is required. The plugin framework handles the communication between the hosted plugin and Oracle Fusion Field Service.

You can host a maximum of 35 plugins per environment. This limitation doesn't include the Standard plugins.

The steps to host a plugin are:

- Complete the prerequisites to upload the plugin.
- Upload the plugin.

After hosting a plugin, you can:

- Use it on a page
- Move between environments
- Modify
- Rollback to a previous version
- Delete

Note: A hosted plugin works only with Oracle Fusion Field Service Mobility Cloud Service.

Prerequisites to Upload a Plug-In

The plugin must be in a specific format to be uploaded. If not, you cannot upload it, you must host it elsewhere.

The plugin files must meet these requirements:

- You must upload a ZIP archive of the plugin files.
- You can upload only the files of following types:
    - .html
    - .css
    - .js
    - .jpg
    - .jpeg
    - .png
    - .gif
    - .svg

    - appcache
- You can organize files in sub-directories, but you must have the "index.html" file in the root folder.
- Each file can be a maximum of 1 MB and the total size of the compressed archive must be less than 500 KB.
- You can have a maximum of 10 files or directories in the archive.

Note: The plugin files uploaded in Oracle Fusion Field Service are available by unique URLs on the Internet. The URLs are generated automatically and contain a long string. There is no authentication to access these files, so anyone who has the direct link to the file can download the file. Therefore, don't store any sensitive information such as passwords or login names in the plugin archive. If you don't want your code to be available without authentication, we recommend that you don't use the hosted plugin functionality. Be aware that the communication between the plugin and Oracle Fusion Field Service starts only when a user successfully logs in to Oracle Fusion Field Service.

Add a Hosted Plug-In

A Hosted plugin is hosted in Oracle Fusion Field Service and uses the Plugin API to interact with Oracle Fusion Field Service.

Note: You can host a maximum of 35 internal plugins excluding Standard plugins. However, there is no restriction on hosting external plugins.

1. Click **Configuration** > **Forms & Plugins**.
   The Forms & Plugins page appears and displays the existing forms and plugins.
2. Click **Add Plugin**.
3. Click **Plugin Archive** and then click **Next**.
4. Complete these fields:

Field Name	Description
General Information section	
Label	A required field defining a unique action or a label for the plugin.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plugin is to be related. For example, if you select Inventory, the action will appear only in the contexts related to inventory. Leave the field blank for the action to be available in all contexts of all the entities.
Visibility rules similar to	The base action from which the plugin is to be derived, if needed. When a base action is selected, the resulting plugin functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plugin. It appears only in the contexts in which the base action appears and is shown or hidden according to the same visibility conditions. For example, if start_activity is selected as the base action for a plugin, the plugin is only be shown in the context of a pending activity when there's no started activity in the same route, similar to the Start action. The list of available base actions is filtered according to the Entity that's selected.
Name (English)	A required field defining the plugin name in the English language. The action or plugin appears under this name in the actual context.

Field Name	Description
Name (other languages)	Plug-in name translations to other languages, if used.
Plugin settings section	
Plugin archive	The zip file for the Hosted plugin, which contains HTML, CSS, and JavaScript files. Click the field to browse and select file, or drag a file.
Disable plugin in offline	Determines whether you want to disable the pluginug-in when Oracle Fusion Field Service is offline. Clear this check box forpluginug-in to work in offline mode.
Plugin parameters	The section where sensitive information such as a user name and password that is used to access external sites is entered. Click plus to add the parameters. The **Add new parameter** dialog box appears with these fields: o Name: Enter a name for the parameter that is used to access an external application. For example, Client ID. o Value: Enter a value for the parameter. o Secure parameter: Select the check box to indicate that the parameter contains sensitive data. When you enable the check box, the application masks the value of the parameter and replaces it with a series of dots. You can deselect the check box and verify that the value is correct until you save the configuration of the pluginug-in. Once you save the configuration, the application masks the value of the parameter on the Edit Plugin and Modify parameter pages. However, when you open an existing secure parameter, you can only replace the value, as the application deletes the value from the field. o Click **Add**. The parameter is added topluginug-in. You can add a maximum of 20 key-value textbox pairs, after which the icon is hidden. The maximum size of the parameters allowed is 5 KB. This size includes the data structure overhead and doesn't correspond to the length of keys and values of strings. Changes to the secure data are sent to Oracle Fusion Field Service during the next synchronization. The data is sent topluginug-in when the next message is sent. If you open the values saved earlier, the application deletes them. You must add them again.
Available properties	The properties that you want to be passed topluginug-in or updated bypluginug-in. These properties are added as read-only and are available through the Plugin API. Click the field to select the properties. You need not define the visibility for the properties explicitly. These properties can't be updated through the Plug-in API: o activity_capacity_categories o auto_routed_to_date o auto_routed_to_provider_id o aworkzone o date o time_delivered You can't add these properties to the list of Available properties: o activity_alerts o access_hours o activity_compliance o atravelarea

Field Name	Description
	○ travel_estimation_method ○ service_window_end ○ service_window_start ○ eta_end_time ○ pid (it's still available for the Resource entity) ○ ctime_delivered_start ○ ctime_delivered_end

5. Click **Add**.

   The archive is uploaded only if these conditions are met:
   - The archive is a ZIP archive and has the extension .zip.
   - The size of the archive is less than 500 KB.
   - The archive includes only directories and files of these types:
     - .html files
     - .css files
     - .js files
     - .appcache files
     - .jpg, .jpeg, .png, .gif, .svg files
     - Directories
   - Files are less than 1 MB.
   - The "index.html" file is found in the root of the archive.
   - The archive includes a maximum of 10 entries, including empty directories.

   If any of these conditions isn't met, an error message is displayed and the archive isn't uploaded.

   To usepluginug-in, you must add it to a button or a link. See the Add the Plug-In to a Page topic. The URL for the hosted pluginug-in is typically https://plugins-0-instance_name.fs.ocs.oraclecloud.com.

Working Offline

You can create the plugin to work offline using two possible approaches, or a combination of them.

The approaches are:

- Using Service Worker API (See https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API): This is the preferred way to implement the offline functionality for the plugin. It is supported by most browsers, except Internet Explorer 11.
- Using Application Cache API (deprecated) (See https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache): This is deprecated and will be removed in the future versions of modern browsers, but it's supported by Internet Explorer 11. The WHATWG (Web Hypertext Application Technology Working Group) notifies that Application Cache API feature is being removed from the web platform. Using any of the offline web application features at this time is highly discouraged. Use service workers instead.
- Combination approach: Use the Service Worker API for modern browsers and Application Cache API as a fallback mechanism for Internet Explorer 11. The basic principles of this solution are:

a. Create the mainfest.appcache file and link it with the index.html by adding the "manifest" attribute to main html tag:

```
<html manifest="manifest.appcache">
```

The manifest.appcache must contain the list of cached files in the "CACHE" section.

> **Tip:** Do not add "index.html" to the CACHE section to enable possible future updates of the plugin's resources.

b. Create the Service Worker javascript file (for example, service-worker.js). This file must implement the network behavior of your plugin using ServiceWorker API. It may load "manifest.appcache" file on the "install" event, parse the Application Cache file and add all the files from the "CACHE" section to browser's cache using the CacheStorage interface. After that you can implement any network behavior strategies to handle the "fetch" event: "network first then cache", "cache first then network", or "network only".

c. Register your Service Worker file at the JS part of your plugin, before sending the "ready" post message, for example:

```
if (navigator.serviceWorker) {
 navigator.serviceWorker.register('service-worker.js').then(function (registration) {
 this.startApplication();
 }.bind(this), function (error) {
 console.error('Service Worker registration failed: ', error);
 startApplication();
 }.bind(this));
} else {
 startApplication();
}
```

In this code example, the `startApplication()`function sends the "ready" post message. It's important to postpone sending the "ready" message until the "install" event is handled properly and all files from the CACHE section of manifest.appcache are loaded to the browser's cache.

Modify, Download, or Delete an Archive

After uploading a plugin archive, you might want to modify it, download it, or delete it.

1. To modify a hosted plugin, you upload a newer version. To upload a newer version of the archive, click **Browse** on the **Modify plugin** page and upload it again.

   You can have only two versions of the plugin at any time. Whenever you upload a newer version of a plugin:
   - The current version becomes a historical one.
   - The newly uploaded version becomes the current one.
   - The newly uploaded version is displayed in the first row of the **Version history** table.
   - The previous version is moved to the second row of the **Version history** table.
2. To download a plugin, click **Download** in the **Version history** section. Save it to the desired location.
3. To rollback to a previous version, download the version that you want to rollback to. Click **Browse** and upload it again.
4. To delete a plugin, first unassign it from all the buttons it's added to. Then, click **Delete** on the **Forms & Plugins** page.

   The plugin is deleted with all its historical versions.

5. To move all the uploaded plugins between environments, export from the required environment using the **Export** function on the **Forms & Plugins** page. Import the exported files using the **Import** function in the target environment.
6. To move a single plugin between environments, download it from the required environment. Upload it in the target environment. You can use the **Export** option here as well.

Add an External Plugin

An external plugin is hosted elsewhere and communicates with Oracle Fusion Field Service through the Plugin API. You add only a link to the plugin here.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **Add Plugin**.
3. Click **External Plugin** and then click **Next**.
4. Complete these fields:

Field Name	Description
General Information section	
Label	A mandatory field defining a unique action or a label for the plugin.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plugin is to be related. For example, if you select Inventory, the action will appear only in the contexts related to inventory. Leave the field blank for the action to be available in all contexts of all the entities.
Visibility rules similar to	The base action from which the plugin is to be derived, if needed. When a base action is selected, the resulting plugin functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plugin. It appears only in the contexts in which the base action appears and is shown or hidden according to the same visibility conditions. For example, if start_activity is selected as the base action for a plugin, the plugin is only be shown in the context of a pending activity when there is no started activity in the same route, similar to the Start action. The list of available base actions is filtered according to the Entity that is selected.
Name (English)	A mandatory field defining the plugin name in the English language. The action or plugin appears under this name in the actual context.
Name (other languages)	Plug-in name translations to other languages, if used.
Plugin settings section	
URL	The path to the URL of the external plugin. This URL processes the HTML5 application and it runs the plugin in the entire browser window. The URL must start with the protocol (https) and must point to the main file of the plugin. Oracle Fusion Field Service adds the backUrl parameter to the URL automatically. This parameter contains the address of the current page of Oracle Fusion Field Service.
Disable button in Offline	Determines whether you want to disable the plugin when Oracle Fusion Field Service is offline. Clear this check box for the plugin to work in offline mode.
Authentication	The type of authentication used by the external server hosting the plugin source to verify access to the plugin. Select one of these options:

Field Name	Description
	○ Basic HTTP: The Basic Access Authentication method working over HTTP or HTTPS. The Basic HTTP authentication method requires a valid login and password. When the entered login and password are verified by the server, the server returns the plugin content. ○ HMAC: Hash-based message authentication code verifying that the data is received from an authorized source. HMAC authentication method requires a secret key configured for each plugin. This field is hidden, if **Hosted plugin** is selected. **Note:** The best practice is to use HMAC authentication instead of basic HTTP authentication. This is because, Google Chrome doesn't support the use of Basic HTTP authentication in sub-resources starting from release 59.
Login/Password	The user name and password to log in to the plugin. These fields are displayed only when Basic HTTP is selected for Authentication.
Plugin parameters	The section where sensitive information such as a user name and password that is used to access external sites is entered. Click plus to add the parameters. The **Add new parameter** dialog box appears with these fields: ○ Name: Enter a name for the parameter that is used to access an external application. For example, Client ID. ○ Value: Enter a value for the parameter. ○ Secure parameter: Select this check box to mask this value. When you save this value, the Edit Plugin and View Parameters pages show 'dots' in this field. ○ Click **Add**. The parameter is added to the plugin. You can add a maximum of 20 key-value textbox pairs, after which the icon is hidden. The maximum size of the parameters allowed is 5 KB. This size includes the data structure overhead and doesn't correspond to the length of keys and values of strings. Changes to the secure data are sent to Oracle Fusion Field Service during the next synchronization. The data is sent to the plugin when the next message is sent. If you open the values saved earlier, the application deletes them. You must add them again.
Available properties	The properties that you want to be passed to the plugin or updated by the plugin. These properties are added as read-only and are available through the Plugin API. Click the field to select the properties. You need not define the visibility for the properties explicitly. These properties can't be updated through the Plug-in API: ○ activity_capacity_categories ○ auto_routed_to_date ○ auto_routed_to_provider_id ○ aworkzone ○ date ○ time_delivered You cannot add these properties to the list of Available properties: ○ activity_alerts ○ access_hours ○ activity_compliance ○ atravelarea ○ travel_estimation_method ○ service_window_end

Field Name	Description
	○ service_window_start ○ eta_end_time ○ pid (it's still available for the Resource entity)

5. Click **Add**.

Add an External Application

You can add an external application as a plugin and it will be opened as a web page in a new window, or the same window within Oracle Fusion Field Service.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **Add Plugin**.
3. Click **External Application** and then click **Next**.
4. Complete these fields:

Field Name	Description
General Information section	
Label	A mandatory field defining a unique action or plugin label.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plugin is to be related. For example, if you select Inventory, the action will appear only in the contexts related to inventory. Leave the field blank for the action to be available in all contexts of all the entities.
Visibility rules similar to	The base action from which the plugin is to be derived, if needed. When a base action is selected, the resulting plugin functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plugin. It appears only in the contexts in which the base action appears and is shown or hidden according to the same visibility conditions. For example, if start_activity is selected as the base action for a plugin, the plugin is only be shown in the context of a pending activity when there is no started activity in the same route, similar to the Start action. The list of available base actions is filtered according to the Entity that is selected.
Name (English)	A mandatory field defining the plugin name in the English language. The action or plugin appears under this name in the actual context.
Name (other languages)	Plug-in name translations to other languages, if used.
Plugin settings section	
Fields for the Open as Web Link Option	
Open as Web Link	Shows the fields to open an external web page from the plugin.
URL	The path to a URL for the external plugin. This URL processes the HTML5 application and it runs the plugin in the entire browser window. The URL must start with the protocol (https) and must point to the main file of the plugin.

Field Name	Description
	Oracle Fusion Field Service adds the backUrl parameter to the URL automatically. This parameter contains the address of the current page of Oracle Fusion Field Service.
POST Data	The data that you want to be sent to the external plugin.
Disable button in Offline	Determines whether you want to disable the plugin when Oracle Fusion Field Service is offline. Clear this check box for the plugin to work in offline mode.
Open inside Field Service	Determines whether the plugin uses the iframe layout. If the field is cleared, the plugin's URL is opened in a new browser tab or window.
Show scrollbars on Dialog	Determines whether the window in which the plugin runs has scroll bars. This setting is applicable to the Legacy Manage application.
Dialog Width in Pixels/ Dialog Height in Pixels	The width and height of the plugin window in pixels. This setting is applicable to the Legacy Manage application.
Fields for the Open as another Application on the same device Option	
Open as another Application on the same device	Shows the fields to open a native app from the plugin.
Native application name	The name of the application to be launched by the plugin.
User-Agent string mask	The browser in which the application is to be launched. The Native application link will be available in GUI, if the browser user agent matches the specified mask. For example, Safari, Android, iPad, iPhone.
Launch application URL	The template for building the external application URL from properties. The URL template contains parameters key and placeholders for parameters value. Properties are interpolated with placeholders, surrounded with braces "{" and "}". For example: http://www.example.com/android?type=LOCATION'&'action=View'&'alt={acoord_y}'&'long=| {acoord_x}'&'address={caddress|url}'&'city={ccity|url}'&'state={cstate}'&'zip={czip}
Add	The button to add another User-Agent string mask.

5. Click **Add**.

Configure a Plugin to Add to the Main Menu

You can add plugins that are created as HTML5 applications to the Main menu. You cannot add native application plugins.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **Add Plugin**.
3. Complete these fields on the **Add Plugin** page:

Field Name	Description
General Information section	

Field Name	Description
Name (English)	A mandatory field defining the plugin name in the English language. The action or plugin appears under this name in the actual context.
Name (other languages)	Plugin name translations to other languages, if used.
Label	A mandatory field defining a unique action or plugin label.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plugin is to be related. For example, if you select Inventory, the action will appear only in the contexts related to inventory. Leave the field blank for the action to be available in all contexts of all the entities.
Visibility rules similar to	The base action from which the plugin is to be derived, if needed. When a base action is selected, the resulting plugin functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plugin. It appears only in the contexts in which the base action appears and is shown or hidden according to the same visibility conditions. For example, if start_activity is selected as the base action for a plugin, the plugin is only be shown in the context of a pending activity when there is no started activity in the same route, similar to the Start action. The list of available base actions is filtered according to the Entity that is selected.
Type	The type of plugin you want to use. Select HTML5 application. This means, the plugin uses an external application to extend the functionality. The HTML5 plugin is a URL of an external resource that is opened in a new window or in an iframe.
Fields for the **HTML 5 Application** Option	
Use Plugin API	Determines whether you want the plugin to communicate with Oracle Fusion Field Service using the Plugin API. Clear this check box. When you do not use the Plugin API, the URL is opened in a new tab, window, or iframe. To interact with Oracle Fusion Field Service you must pass some data (such as activity id, resource name) to the plugin using the placeholders in the "POST Data" and "URL" fields.
URL	The path to a URL (for external plugins). This URL processes the HTML5 application and it runs the plugin in the entire browser window. The URL must start with the protocol (https). The URL must point to an external resource, which is opened either in a new window or inside Oracle Fusion Field Service in an iframe (if the “Tab or iframe layout” option is selected).
POST Data	The data that you want to be sent to the external plugin. You can use only User entity fields as placeholders.
Disable plugin in offline	Determines whether you want to disable the plugin when Oracle Fusion Field Service is offline. Clear this check box for the plugin to work in offline mode.
Open in iframe	Determines whether the plugin uses the iframe layout. If you clear the field, the plugin’s URL is opened in a new browser tab or window.

4. Click **Save.**

   If you set the external plugin as the first item on the Main menu context layout, the menu item is displayed in the correct order. However, the plugin does not open when a user logs in to the application. Instead, a standard page or a plugin that you have created using the Plugin API Framework that is next in the order is opened.

Available Placeholders for POST Data for Main Menu Plug-Ins

You can use only User entity fields as placeholders for the POST data and URL.

Here are the fields that you can use as placeholders:

Placeholder	User Property
allow_desktop_notifications	Popup Notification
allow_vibration	Vibration
design_theme	Design Theme
main_resource_id	Main Resource
mobile_activity_count	Mobile Activity Count
mobile_inventory_count	Mobile Inventory Count
mobile_provider_count	Mobile Resource Count
sudate_fid	Date Format
sulong_date_fid	Long Date Format
sustatus	Status
sutime_fid	Time Format
su_zid	Time zone
uid	User ID
ulanguage	Language
ulogin	Login
uname	User
uname	User Name
user_type_id	User Type

Some of these properties are available only if you add them to an application page, either directly or through a button for a Form for which you have configured these fields.

Plugin Configuration Screen

The new "Applications" section has been added to the screen for hosted and external plugins.

Each Application in this section has the following attributes:

- **Key** - identifier of an application that's used in plugin code that should be associated with the existing application in Field Service.

- **Comment** - description of the required application that can provide more details about this API (e.g. "for obtaining inventory list from Fusion SCM").

Once the plugin is imported with XML file and the required applications were specified there, then configurator will see keys and descriptions in the list of 'Application' section, then they can select an appropriate application from the drop down list. In case the application is not associated, then the API access is not provided to the plugin. In case the selected application is not of a supported type, then the mobile plugin framework will return an Error.

If the plugin was added from the Forms & Plugins screen, the Application section is empty by default and the Configurator can add it from scratch, by clicking the 'Add' button, specifying the application identifier in the Key field and adding necessary Comment.

Control Field Service REST API calls according to current user visibility

In case you want to restrict access to the REST API calls by JWT access token to the visibility of the current user, the checkbox "Application details" should be enabled. If a REST API call tries to obtain data from a resource outside of the Visible Resources list of the current user, it will be denied.

Change the Plugin Tile Appearance

Use the *iconData* parameter to change the appearance of the plugin button on the Landing page. You can change the status text, icon image, and color of the plugin button. The status you can show includes number of processed activities, status of the order, number of pending actions, and so on. When the data is synchronized successfully, you can even change the icon to indicate it.

1. Determine the data that you want to display or update and the message through which you want to update.
2. Send the information that you want to display or update, in the *iconData* parameter.
   iconData is available for *close*, *initEnd*, and *sleep* messages. The data is applied real time. That is, if the plugin is woken up when the user is on the Landing page, the icon is updated immediately after the plugin sends the *iconData* parameter in the *sleep* message. The *iconData* parameter includes these fields:

Field	Type	Limits	Description
color	String	One of: o "default" o "highlight"	If the value is set to "highlight", the plugin's button on the Landing page changes its color to get the user's attention.
image	Blob	o Max size: 64 KiB o Allowed types: - image/svg+xml - image/png	Icon picture. It's recommended to use a scalable monochrome white icon in SVG format. If the bitmap picture is in PNG format, it must not be smaller than 64x64 px.
text	String	Max length: 3 chars	The short text that is shown as large title on the plugin's button. May contain both letters and digits.

Example of the *close* Message:

```
{
var closeData = {
 "apiVersion": 1,
```

```
 "method": "close",
 "iconData": {
 "color": "highlight",
 "text": "117",
 "image": new Blob([
 '<?xml version="1.0"?>' +
 '<svg xmlns="http://www.w3.org/2000/svg" version="1.2" baseProfile="tiny" viewBox="0 0 64 64">' +
 '<rect x="16" y="16" width="32" height="32" fill="#fff" />' +
 '</svg>'
 ], { type: 'image/svg+xml' });
 }
}


window.parent.postMessage(closeData, origin);
}
```

Placeholders in the URL

You can use several placeholders in the plugin's URL. The placeholders are replaced with the values of the corresponding properties and are processed by the server that hosts the plugin

This table describes the placeholders.

Placeholder	Description
{user_id}, {uid}	ID of current user
{date}	current date
{uname}	User name
{ulanguage}	ID of user language
{ulogin}	User login
{su_zid}	User timezone
{allow_desktop_notifications}	Parameter defining whether the user allows HTML5 notifications
{allow_vibration}	Parameter defining whether the user allows vibration alerts

Authentication

You can use the HTTP Basic or HMAC method of authentication to load the plugin's URL securely in the *init* stage.

HTTP Basic

The HTTP Basic method uses the standard method, which is a part of the HTTP 1.0 standard (RFC 1945) called Basic Access Authentication. It works over HTTPS as well. Check whether your browser supports this, before you decide to use this method. This table describes the three conditions you must fulfill to implement the HTTP Basic method.

Condition	Description
Oracle Fusion Field Service Configuration	On the **Forms & Plugins** page, select "HTTP Basic" authentication type. Fill up the **Login** and **Password** fields. These credentials are encrypted and saved to the Oracle Fusion Field Service database.
Server Side	Configure the web server on which the plugin sources are hosted to return the HTTP 401 Unauthorized status, if you are requesting the configured plugin URL without the credentials. See the NGINX and Apache documents for details. The server must return the plugin content if its URL is requested with the HTTP header. **`Authorization: Basic bXlsb2dpbjpteXBhc3M=`** Where bXlsb2dpbjpteXBhc3M= is a valid Base64 - encoded pair of login:password. The credentials configured for the plugin in the **Add plugin** and **Modify plugin** pages must be accepted as valid.

Client Side	When the user logs in, Oracle Fusion Field Service reads the credentials from the database and loads the plugin URL into the hidden iframe as follows: **`<iframe src="https://mylogin:mypass@example.com/myPlugin.php"/>`** This way, the browser loads the plugin sources over HTTPS using HTTP Basic Authentication:
	`GET /myPlugin.php HTTP/1.1`
	`Host: example.com`
	`Authorization: Basic bXlsb2dpbjpteXBhc3M=`

HMAC Authentication

HMAC (Hash based message authentication code) lets you sign HTTP requests and their GET parameters. The HMAC signature ensures that the URL is generated by an authorized source. The MAC signature (digest) is added as an additional GET parameter at the end of a query string: **`<!CDATA[[http://www.example.com/path?user=test&section=D%26G&activity=33&hmac=D2BJn9P1EcLhaFrNhbAzCQTVQXCCwCBQsrg8V6h4YoU%3D]]>`**

HMAC Function Algorithm

The algorithm is defined in RFC 2104 , and can be very roughly described as: **`hmac = BASE64(HMAC-SHA-256(data, SHA256(SecretKey)))`**. SHA - 256 accepts SecretKey as a string and returns the hash string. The secret key is configured per plugin in the **Add plugin** and **Modify plugin** pages in Oracle Fusion Field Service Core Application, hashed by SHA256, encrypted and stored in the database. HMAC-SHA-256 accepts data and key as strings and returns a binary array of HMAC signature. BASE64 accepts the binary array and returns BASE64 encoded string. Data required for generating HMAC is query resource location with query parameters sorted lexicographically:

- Remove the protocol identifier from the URL together with colon and slashes (http:// or https://).
- Remove the resource name and port from the URL.
- Append query location to the output string.
- If there are query parameters append the character ? to the output string.
- Decode every name and value for URL parameters.
- Sort the list of parameters alphabetically by name.
- For each name/value pair:
  - Append the encoded name to the output string.
  - Append the '=' character to the output string.
  - Append the encoded value to the output string.

- If there are more key/value pairs remaining, append an & character to the output string.

Example: Request URL: http://www.example.com/path?user=test&section=D%26G&activity=33

SecretKey : 'mysecret'

1. http://www.example.com/path?user=test&section=D%26G&activity=33 => www.example.com/path?user=test&section=D%26G&activity=33
2. www.example.com/path?user=test&section=D%26G&activity=33 => /path?user=test&section=D%26G&activity=33
3. data = '/path'
4. data = '/path?'
5. ['user'='test','section'='D&G','activity'=33]
6. ['activity'=33,'section'='D&G','user'='test']
7. ['activity'=33,'section'='D&G', 'user'='test'] => data
8. data = '/path? activity'
9. data = '/path? activity='
10. data = '/path? activity=33'
11. data = '/path? activity=33&'
12. data = '/path? activity=33&section=D %26G&user=test'

hmac = BASE64(HMAC-SHA-256('/path?activity=33&section=D%26G&user=test',SHA256('mysecret'))) = BASE64(HMAC - SHA - 256('/path? activity=33&section=D %26G&user = test' ,'652c7dc687d98c9889304ed2e408c74b611e86a40caa 51c4b43f1dd5913c5cd0')) = BASE64([0f,60,49,9f,d3,f5,11,c2,e1,68,5a,cd,85,b0,33,09,04,d5,41,70,82,c0,20,5 0,b2,b8,3c,57,a8,78,62, 85]) = 'D2BJn9P1EcLhaFrNhbAzCQTVQXCCwCBQsrg8V6h4YoU='

The full signed URL is 'http://www.example.com/path?user=test&section=D %26G&activity=33&hmac=D2BJn9P1EcLhaFrNhbAzC QTVQXCCwCBQsrg8V6h4YoU%3D'

Sensitive Data

You can set key-value pairs of sensitive data that is securely stored by Oracle Fusion Field Service on both, client and server sides. Examples of sensitive data include passwords and endpoints for Oracle Fusion Field Service Core API or external APIs.

Sensitive data is passed to the plugin through the plugin API in decrypted form, so the plugin can access APIs or third-party services without having to store and secure the credentials. The plugin stores the sensitive data in a JavaScript variable every time it receives a message from Oracle Fusion Field Service. Changes to this data are sent to Oracle Fusion Field Service during the next synchronization. This data is sent to the plugin when the next message is sent. The plugin also receives the up-to-date data with every message.

Configure Sensitive Data

You add sensitive information in the **Secure parameters** section on the **Configuration** > **Forms & Plugins** > **Add/Modify plugin** page. You can add up to 20 key-value pairs. When the plugin is modified, JSON is read from the database, decrypted, parsed, and displayed as key-value text box pairs, maintaining original order. Key and values are validated against length limitations in this way:

- Key-value pairs are translated to JSON.
- Length of the resulting JSON string (in bytes) is displayed on the page.
- If the length exceeds 5 KB, a warning message is shown.

Additionally, to prevent request forging, the resulting string is validated on the server to be:

- Valid JSON string
- Has correct format
- Doesn't exceed length limit

If these requirements aren't met, a warning message is shown. Each message sent by Oracle Fusion Field Service Core Application to a plugin, where the method is one of the supported methods, contains the field *securedData*. Format of the messages for other methods, for example, 'error' does not change. The message contains *securedData*, if at least one key-value pair is configured on the **Configuration** > **Forms & Plugins** > **Add/Modify plugin** page.

Supported Methods

The *securedData* parameter is available for the messages of init, open, and wakeup methods.

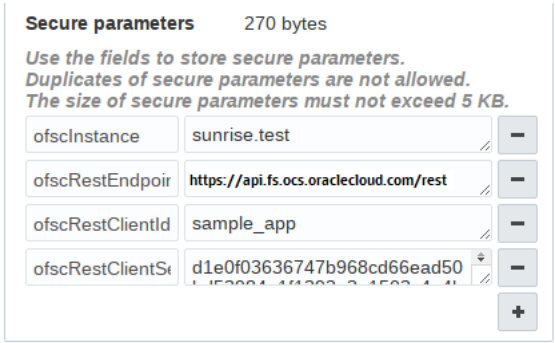
Format of *securedData*

securedData is an object, where:

- Each key is a String, which equals to the contents of "key" text input on the Add/Modify plugin page.
- Each value is a String, which equals to the contents of "value" text input for the corresponding key on the Add/Modify plugin page.
- Order of entries is not guaranteed to be identical to the order of key-value pairs on the Add/Modify plugin page.

Example of *open* method data for Supervisor Plug-in

If your configuration is as given in the screenshot:



Here's the message the plugin receives when opened:

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "resource": {
 "external_id": "33001",
 "manager": "admin"
 },
 "activityList": {
 "4224031": {
 "aid": "4224031"
 }
 },
 "inventoryList": {
 "21064417": {
```

```
"invid": "21064417"
}
},
"securedData": {
"ofsInstance": "sunrise.test",
"ofsRestEndpoint": "https://<instance_name>.fs.ocs.oraclecloud.com/rest/",
"ofsRestClientId": "sample_app",
"ofsRestClientSecret": "d1e0f03636747b968cd66ead50bd53984e1f1393a3e1503c4e4be9421be00aa5"
}
}
```

Add a plugin to a page

You add a plugin to a context layout page, so that Mobile Workers can open it. You can configure the parameters for a button to send the parameters to the plugin, or to open a specific page, or another plugin.

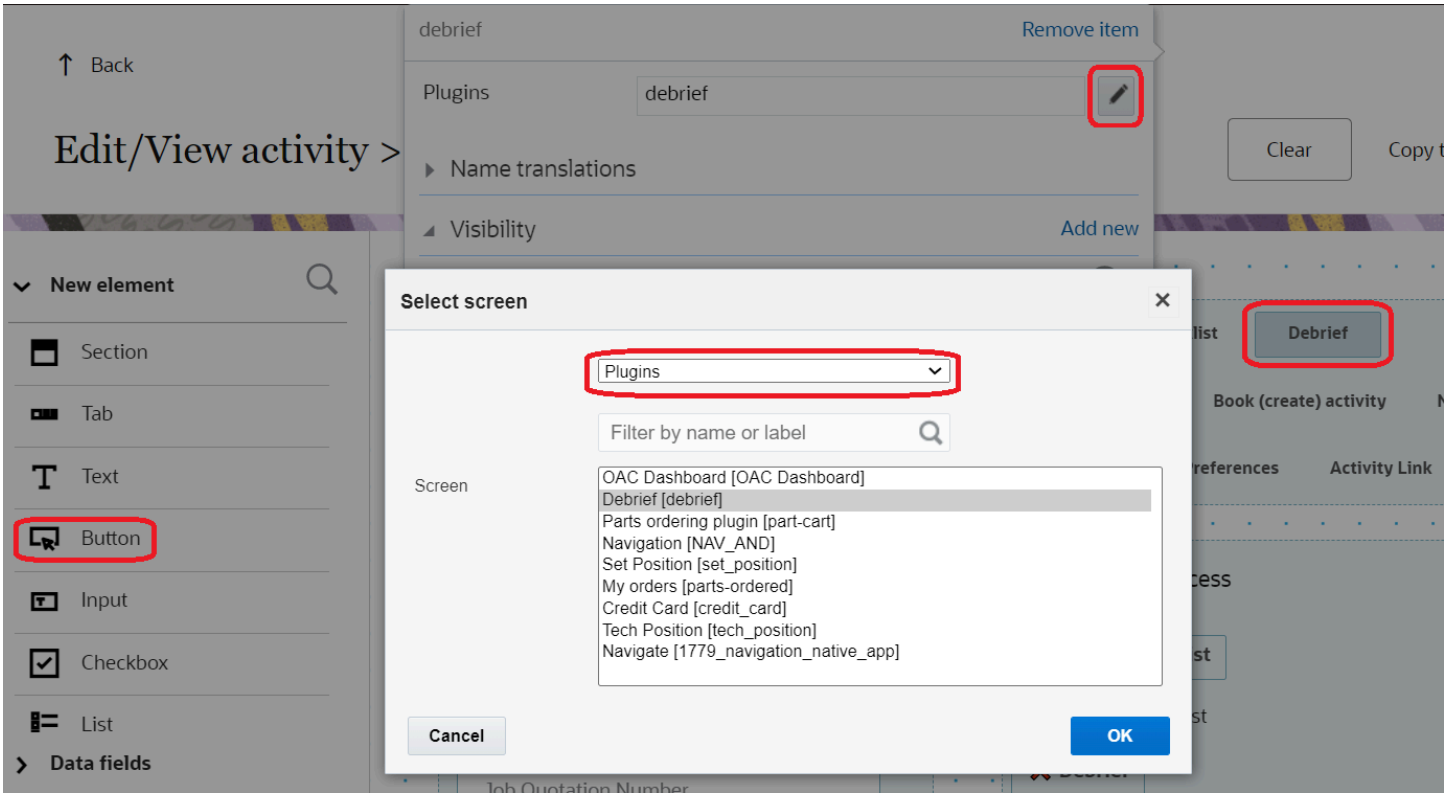
1. 1. Click **Configuration** > **User Types**.
2. Select the type of user for which you want to add the plugin.
3. Click **Screens**.
4. Find and click the page to which you want to add the plugin.

   The **Visual Form Editor** page appears. Plug-ins are available not only on the Visual Form Editor, but on old context layout structures such as Parts Details as well. On such pages, add an action and select a plugin from the list.
5. Drag-and-drop the **Button** element to the section from where you want to invoke the plugin.

   > **Note:** You cannot add buttons to context layout structures that are responsible for changing the state of an activity, simultaneously with submitting data. Some of the context layout structures where you cannot add buttons are Add activity, Not done activity, Install inventory, End activity. Further, you cannot remove or change the visibility of the two predefined buttons on these pages: Dismiss and Submit. This is to preserve the data integrity within transitions between states.
6. Click the button.
7. In the **Standard action screen** field, click the pencil icon.
8. Select **Plug-ins**.
9. In the **Screens list**, select the name of the plugin that you want to open and click **OK**.

   The label of the plugin is displayed in the **Plug-in** field. By default all plugins have a visibility of Read-only.
10. In the **Visibility** section, add the conditions based on which the plugin is visible.

11. In the **Translations** section, add a name for the plugin. [Optional]
    This name is displayed on the page from which the plugin will be invoked. This screenshot shows the Visual Form Editor page where a plugin is added to a Button element:



    If you retain the default name and if you happen to change the name of the plugin later, the new default name is populated automatically.

12. To configure the parameters:
    a. Click **Add new** in the **Parameters** section.
    b. Enter a name for the parameter in the **Name** field.
       For example, enter defaultScreen to define a page as the default page in the plugin. The maximum length of the name that you can enter is 248 characters.
    c. Enter a value for the parameter.
       For example, enter part_order to display the Part order page as the default page in the plugin. The maximum length of the value that you can enter is 4000 characters.
    d. Click **Save**.
    e. Repeat the procedure for all the parameters that you want to configure.
       The total combined length of all parameter names and values must not exceed 5000 characters. These parameters are not encrypted when sent to the plugin.
13. Click **Save** on the **Visual Form Editor** page.
    The plugin is added to the selected page.

Plugin Buttons and Icons

Let's say you use a plugin that contains several pages and you want to configure links that open different pages of the plugin. Or, you want to configure different icons for different tiles of the plugin on the Landing page and change the icon of the tile according to the data in the plugin.

You can configure the plugin for all these scenarios, using the Parameters option:

- Configure parameters for a button associated with a plugin: You can specify custom parameters for each button of a plugin. For example, when you click the button, you can open a specific page in the plugin. You can implement this on multiple pages, by uploading a single plugin archive.
- Update icons for a button on the Landing page: You can update the appearance of each button on the Landing Page separately. You can change the icon when you open Oracle Fusion Field Service Core Application ('initEnd'), after closing ('close') a plugin, or when a plugin receives new data in the background mode ('sleep'). You can also associate each plugin's page or function with its own icon and text. You can update each icon as needed, no matter with which button you open the plugin.

How it Works

The data sent to the plugin during initialization includes the list of all buttons configured for each plugin with the key-value pairs of configured parameters. When a user opens the plugin, the ID of the button is passed to the plugin with the parameters configured for this button. When a user closes the plugin, you can redirect the user to another plugin, as if the plugin has been opened through a button, and parameters that are configured for this button are passed to the plugin.

Modify the Icons and Text of a Plug-In Tile

You can implement a flexible flow for the buttons of a plugin on the Landing Page. You can change the icons on the buttons individually, or all at once. The icon can be changed when the Oracle Fusion Field Service Core Application is opened ('initEnd'), after plugin is closed ('close'), or when a plugin gets new data in the background mode (‘sleep’). See the Example of Changing Buttons in a Plug-in topic to understand how it works.

List of all buttons that are configured for a plugin is sent to the plugin in the 'buttons' field of the 'init' message. This field is a list of objects that contain the 'buttonId' and 'params' fields. buttonId is the 'context layout item id' of the button. 'params' is an object that represents the parameters that are configured for the corresponding context layout item.

The 'open' message contains the buttonId and openParams fields. buttonId is the 'context layout item id' of the button that the user clicks to open the plugin. openParams contains the parameters that are configured for this button.

If a plugin is opened by sending the backScreen: "plugin_by_label" from another plugin, the buttonId and openParams fields are sent in accordance with the backPluginButtonId param of the 'close' message. If the backPluginOpenParams field of the 'close' message contains the key, which is already configured for the button, the openParams field contains the value that's sent in backPluginOpenParams.

If backPluginButtonId was not set, the 'open' message doesn't contain the buttonId and openParams fields. This table shows the data in the 'init', ‘open’ and ‘close' messages that is used for changing appearances:

Message Type	Field	Description
init	buttons	List of objects that contain the 'buttonId' and 'params' fields. • buttonId: Context layout item id of the button. • params: Object that represents the parameters, configured for the corresponding context layout item.
open	buttonId	Context layout item id of the button that the user clicks to open the plugin.
	openParams	The parameters that are configured for this button.
close	buttonId	Context layout item id of the button that the user clicks to open the plugin.
	openParams	The parameters that are configured for this button.
	backPluginButtonId	

init Message

```
{
 "apiVersion": 1,
 "method": "init",
 "attributeDescription": {
 "aid": {
 "fieldType": "field",
 "entity": "ENTITY_ACTIVITY",
 "gui": "text",
 "label": "aid",
 "title": "Activity ID",
 "type": "string",
 "access": "READ_WRITE"
 }
 },
 "buttons": [
 {
 "buttonId": "17155",
 "params": {
 "defaultScreen": "order-part",
 "someOptions": "{showCart: true}"
 }
 },
 {
 "buttonId": "17156",
 "params": {
 "defaultScreen": "search-parts"
 }
 }
 ]
}
```

open Message

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "resource": {
 "pid": 8100059
 },
 "activityList": {
```

```
 "4225376": {
 "aid": "4225376"
 }
 },
 "inventoryList": {},
 "buttonId": "17155",
 "openParams": {
 "defaultScreen": "order-part",
 "someOptions": "{showCart: true}"
 }
}
```

close Message: Navigate to Another Plug-in

```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "plugin_by_label",
 "wakeupNeeded": false,
 "backPluginLabel": "sample_plugin",
 "backPluginButtonId": "17155",
 "backPluginOpenParams": {
 "someOptions": "{ anotherOption: 123 }",
 "thirdParam": null
 }
}
```

open Message: Navigated from Another Plug-In

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "activityList",
 "resource": {
 "pid": 3000001
 },
 "activityList": {},
 "inventoryList": {},
 "buttonId": "17155",
 "openParams": {
 "defaultScreen": "order-part",
 "someOptions": "{ anotherOption: 123 }",
 "thirdParam": null
 }
}
```

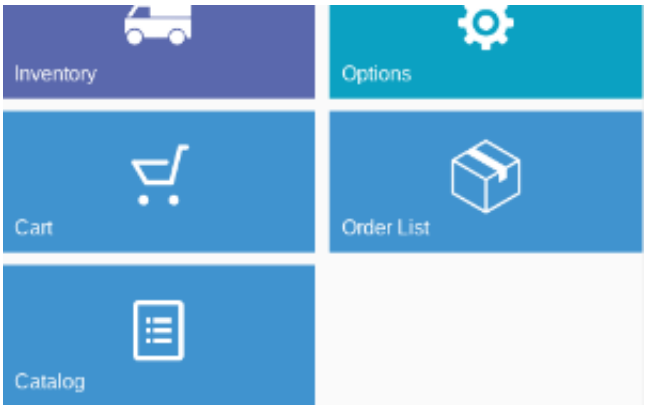
close Message: Update Icons

```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "default",
 "wakeupNeeded": false,
 "buttonsIconData": {
 "17156": {
 "color": "highlight",
 "text": "123",
 "image": {}
 },
 "17155": {
 "color": "default",
 "text": null,
 "image": {}
 }
 }
}
```

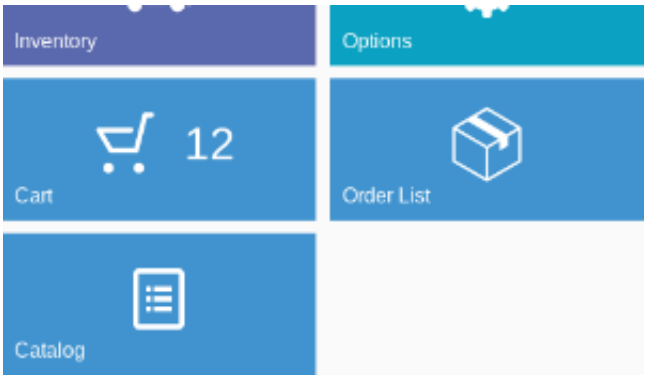
Example of Changing Buttons in a Plugin

Consider the example of a catalog and a cart. The Order List Button changes its color when the supervisor approves the order.

1. There's a plugin, which implements three pages: "Catalog", "Cart" and "Order List". Each button has a corresponding icon:

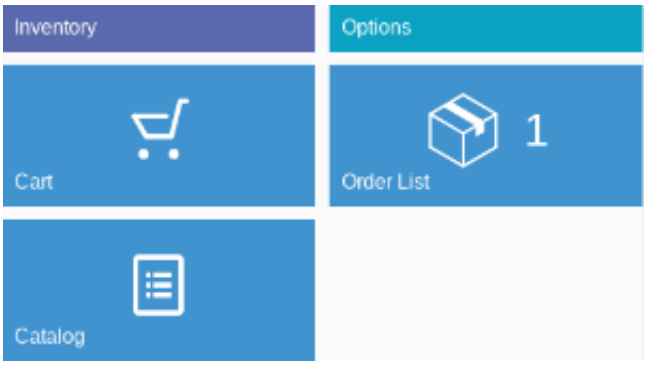


2. User clicks "Catalog" and the plugin shows the list of items available for order.
3. User selects the items to order and closes the plugin page.
4. The plugin updates the "Cart", so that it shows the count of the items in the cart:

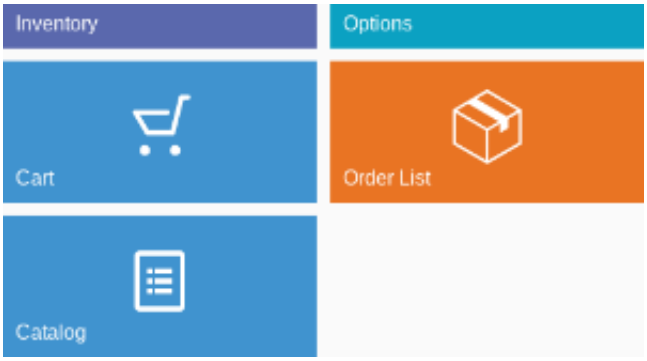


5. User clicks "Cart" and the plugin shows the list of ordered items.
6. User confirms the order and closes the plugin page.

7. The plugin clears the counter on the "Cart" button and updates the "Order List" button so it shows the number of orders for approval:



8. The plugin prompts Oracle Fusion Field Service to run it in the background every five minutes to get the updated information from the server.
9. The supervisor of the user approves the order.
10. The plugin runs in the background and receives the updated status of the order from the server. It clears the counter on the "Order List" button and highlights it:



User sees that the order is processed and approved without the need to reopen the plugin.

Export and Import Plugins

When you've plugins that work in a similar way, you can export the properties and configuration from one plugin and import them into another. You can export or import multiple plugins simultaneously.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **Export** in the header.
3. On the **Export Forms or Plugins** dialog box, select **Export plugins**.
4. Select the checkbox against the plugins that you want to export.
   A counter shows the total number of plugins available in the environment and the number you've selected.
5. Click **Export**.

The selected plugins are exported with their configuration as a single .xml file. Plug-ins with secure parameters are exported with the keys, but without their values.

6. To import a plugin configuration, click **Import** > **Plugins**.
7. Drag the XML file that you've exported.
8. Click **Continue**.

   The plugins are validated and errors, if any, are displayed.
9. Fix the errors and import the file again.
10. Click **Apply**.

    The plugins are imported.

Revision History

This document will continue to evolve as existing sections change and new information is added.

Date	What's Changed	Notes
February 2023	These topics are updated: • Add a Hosted Plug-In • open Method These topics are added: • Add a Standard Plug-In • Modify the Settings of a Standard Plug-In • Change the Code of a Standard Plug-In • Install the Sample Plug-In • Overview of Debrief Plug-In • Configure the Debrief Plug-In • Install the Debrief Plug-In • Add the Inventory Types • Add the Parts Catalog • Add the Debrief Plug-In to a Page • Add an External Plug-In • Add an External Application This topic is deleted: • A Sample Plug-In	
August 2022	These topics are updated: • callProcedure Error Handling • Sample Plug-In These topics are added: • share Procedure • Configure a Custom Domain for the Where is My Technician URL	
February 2022	These topics are added: • print Procedure • updateIconData Procedure • updateButtonsIconData Procedure • allowedProcedures Field These topics are updated: • Export and Import Plug-Ins	

Date	What's Changed	Notes
	• callProcedure Error Handling • callProcedure Method • wakeup Message • Plug-In Lifecycle • Redirection with the close Method • Sample Plug-In	
November 2021	These topics are added: • update Method • updateResult Method These topics are updated: • Supported Activity Actions • init Method • Error Codes for Inventory Actions • A Sample Plug-in	
August 2021	These topics are updated: • The Plug-in Framework • open Method	
May 2021	These topics are added: • Available Fields for the 'queue' Entity Collection • Supported Queue Actions • Error Codes for Queue Actions These topics are updated: • open Method • close Method • Sample Plug-In	
February 2021	These topics are updated: • init Method • open Method • Non-Serialized Inventory Update • Example of the close Message	