Field Service

Mobile Plug-in Framework

Field Service
Mobile Plug-in Framework

F75115-08

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 Plug-In API

About the Plug-In API

Customers or third-party integrators can extend the functionality of Oracle Field Service using the plug-in API. Plug-ins that use this API are opened inside the application like any other application page and support complex business flows, even when the user is offline. Data that is received from the plug-in or is sent to it, is synchronized with the application server-side, so there's no need to pass the data from the plug-in to the server.

The Plug-In Framework

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

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

The plug-in framework offers these features:

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

The plug-ins can manipulate these entities:

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

Consequently, the plug-ins can be added to these contexts:

- Activity list
- Edit/View activity

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

Plug-In Rules and Guidelines

You create a plug-in on the **Forms & Plugins** page and add it as a button on the required page. You can also add buttons on the activity hint.

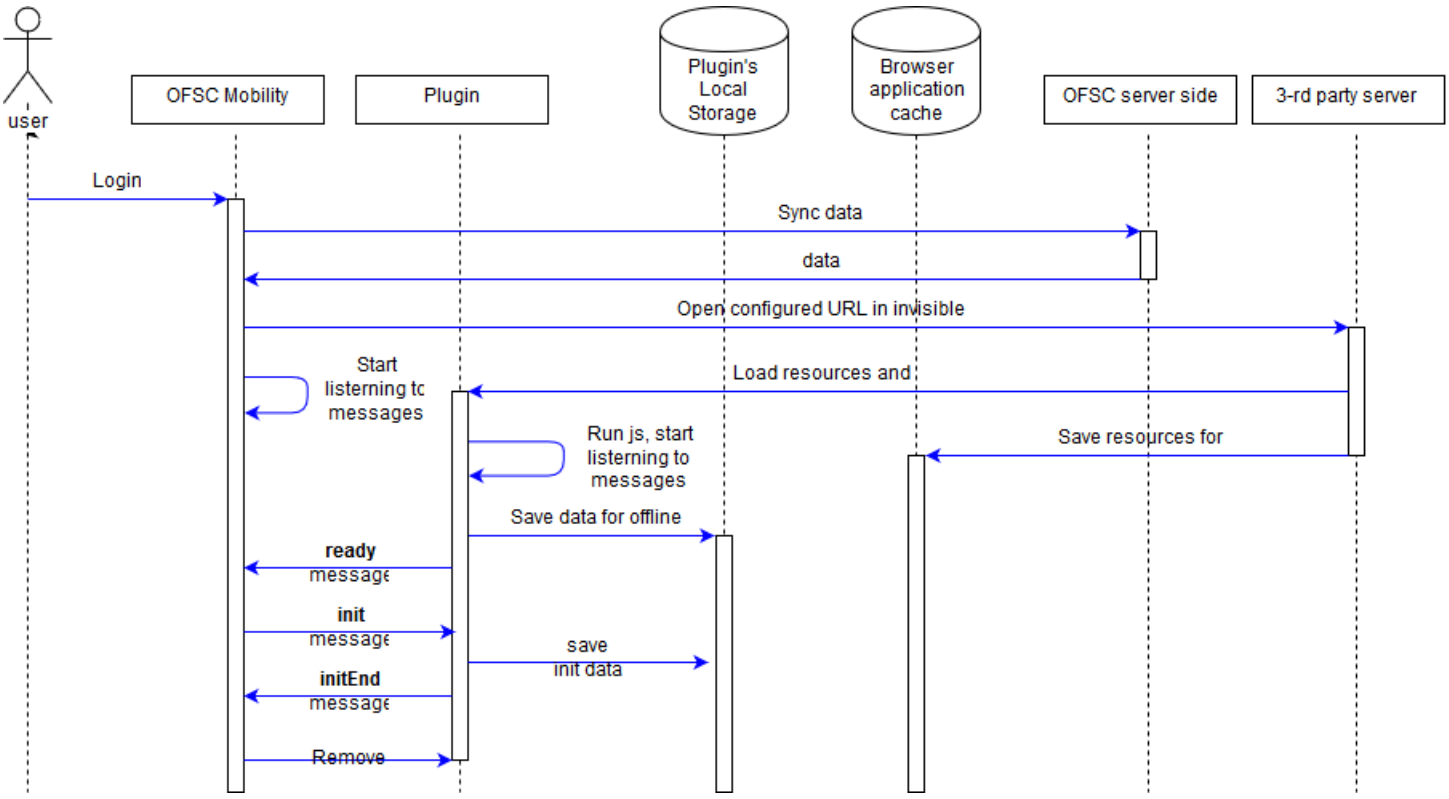
Here are some rules and guidelines that you must follow, while creating a plug-in:

- The plug-in URL must point to the main page of the plug-in's source files. The main page must include valid HTML/XHTML content that loads or contains JavaScript code sources and static resources such as images and .css style sheets.
- The plug-in needs the HTTPS protocol.
- For a plug-in to be treated as a valid plug-in, the main page must run the JavaScript code that interacts with Oracle Field Service through the plug-in API.
- To make a plug-in 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 plug-in can save data for offline locally on the user's device using cookies, the localStorage property, or the IndexedDB object.
- The plug-in's URL is loaded into iframe, and the URL points outside the Oracle Field Service domain. Therefore the plug-in's application cache, cookies, the localStorage property, and the IndexedDB object are separated from that of Oracle 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 plug-in's JavaScript. So, the only way to interact with Oracle Field Service is through the plug-in API.
- The plug-in API is based on messages. Oracle Field Service sends the messages and the plug-in receives them. Similarly, the plug-in sends the messages and Oracle 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.

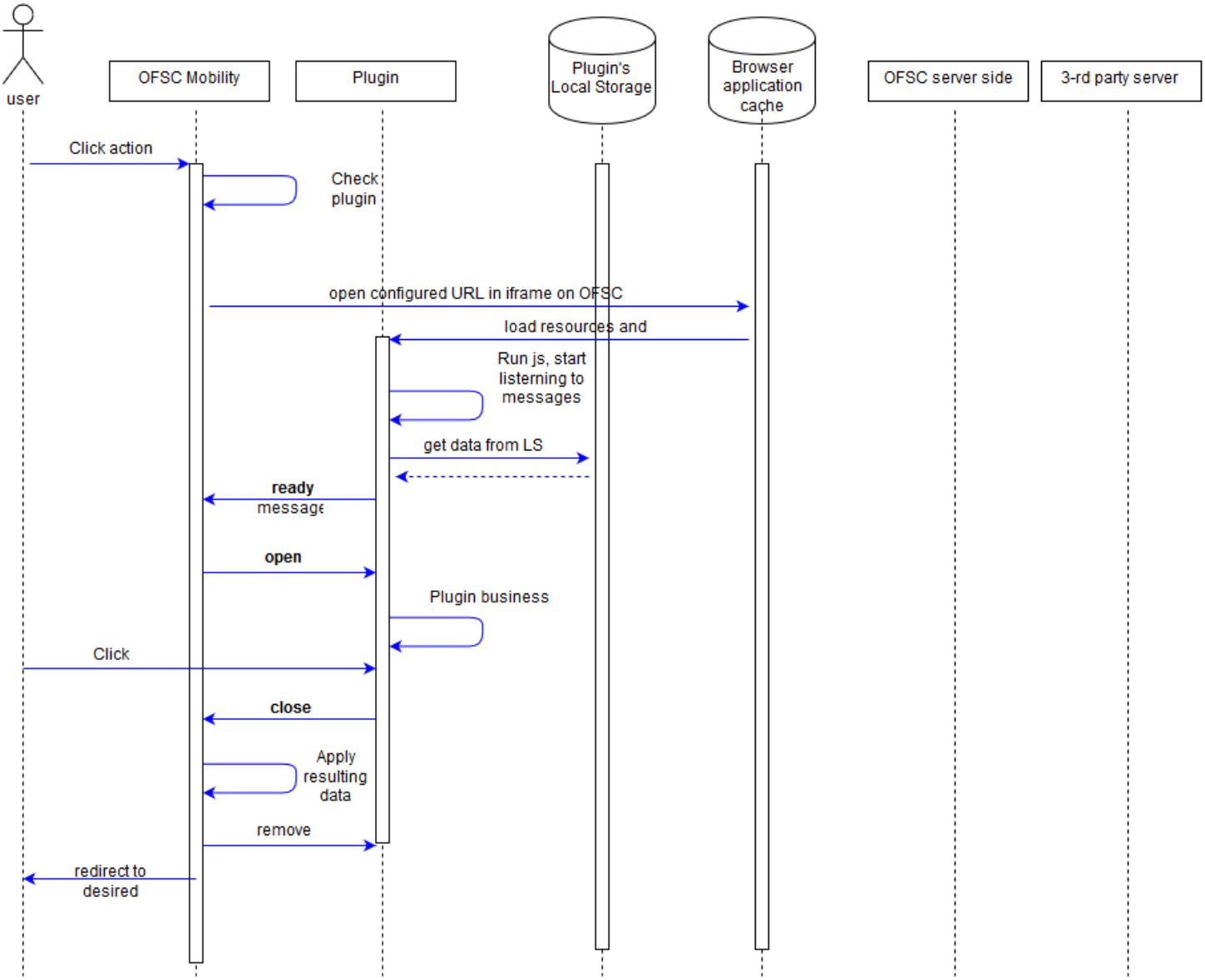
Flowcharts

This topic provides the flowcharts that show how the plug-in framework works.

This flowchart shows the plug-in's initialization flow:



This flowchart shows the plug-in'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 Plug-In API Messages

Message Formats

A message can be sent 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 plug-in must process the messages that it receives. Oracle Field Service always sends the data to the plug-in 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 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 Field Service entity that's to be processed by the plug-in. Available only for the 'open' method.
- resource, inventoryList: Entity data collections. Available only for 'open' and 'close' methods.

Common Fields

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

Related Topics

- Does the plug-in apply limits to property values?

Available Methods

Initiated by Oracle Field Service:

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

Initiated by the plug-in:

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

init Method

A message with the init method indicates that Oracle Field Service Core Application has started initializing the plug-in.

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 plug-in.

List of all buttons that are configured for a plug-in is sent to the plug-in 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.

This table provides the fields in the enum item of the invtype property:

Key	Type	Is Optional	Description
nonSerialized	boolean	No	**false:** Inventory of this type is serialized (that is, it has a serial number and its quantity is always 1). Examples: Network router, mobile phone, air conditioner **true:** Inventory of this type is not serialized (that is, it doesn't have a serial number and its quantity is an arbitrary number). Examples: Ethernet cable, connector, coolant
quantityPrecision	integer	Yes	The key is present only if the value of the 'nonSerialized' field is true. The value 0 means that the 'Decimal quantity' option is not enabled for the inventory type, so the quantity is always an integer.
unitOfMeasurement	string	Yes	The unit of measurement text which is shown next to the quantity in the GUI. Doesn't affect any calculations. The key is present only if the value of the 'nonSerialized' field is 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",
"features": {
"isSegmentingEnabled": true
}
},
"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"
}
}
}
```

initEnd Method

The messages with this method indicate that the plug-in has finished processing the initialization data.

When Oracle Field Service Core Application receives the initEnd message from a plug-in, it destroys the plug-in's iframe. The message may also contain the optional 'wakeupNeeded' field, which allows to continue background activity of the plug-in after a page is reloaded. See the description of the wakeup method for details. You can change the appearance of the plug-in 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 Field Service ignores all other fields.

Related Topics

- wakeup Message

open Method

When a user opens a plug-in through a button, a message with the open method is sent to the plug-in after Oracle 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 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 plug-in 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

```
{
 "manifestVersion": "1",
 "pluginApiVersion": "1",
 "productVersion": "19.5.0",
 "label": "debriefing",
 "name": "Debriefing",
 "description": "Debriefing is the process of reporting time and materials that were used while performing
 the work order.\nThe reporting is usually done by a field technician.",
 "icon": "oj-ux-ico-adapter",
 "buttonEntity": "activity",
 "secureParameters": ["ofsRestApiEndpoint", "ofsRestApiClientId"],
 "properties": {
 "csign": {
 "entity": "activity",
 "type": "file",
 "gui": "signature",
 "translations": [
 {
 "language": "en",
 "name": "Signature"
 }
 ]
 },
 "labor_service_activity": {
 "entity": "inventory",
 "type": "enumeration",
 "gui": "combobox",
 "translations": [
 {
 "language": "en",
 "name": "Labor Activity"
 }
 ],
 "items": [
```

```
{
"label": "com",
"active": true,
"translations": [
{
"language": "en",
"name": "Commute"
}
]
},
{
"label": "drp",
"active": true,
"translations": [
{
"language": "en",
"name": "Diagnose and Repair"
}
]
}
]
},
"labor_end_time": {
"entity": "inventory",
"type": "string",
"gui": "text",
"lines": 1,
"translations": [
{
"language": "en",
"name": "Labor End Time"
}
]
},
"invoice": {
"entity": "activity",
"type": "file",
"gui": "file",
"mimeTypes": "application/pdf",
"fileSizeLimit": "5",
"translations": [
{
"language": "en",
"name": "Invoice"
}
]
},
"pid": {
"entity": "provider"
},
"pname": {
"entity": "provider"
},
"astatus": {
"entity": "activity"
},
"aid": {
"entity": "activity"
},
"invid": {
"entity": "inventory"
},
"invsn": {
"entity": "inventory"
}
}
```

```
}
```

Example of the open message for a plug-in opened from the Main menu

```
{
 "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 plug-in opened from the Parts Catalog

```
{
 "apiVersion": 1,
 "method": "open",
 "entity": "partsCatalogItem",
 "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"
 }
 }
}
```

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 Field Service Core Application page from which the user opens the plug-in. 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 inventoryList
Activity List -> Inventory List	inventoryList	
Activity List -> Activity Details	activity	user team queue resource activityList activity inventoryList
Activity List -> Activity Details -> Inventory List	activityInventoryList	
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

Page	Entity Field Value	Available Collections
		resource activityList activity inventoryList inventory
Inventory Search -> Parts Catalog Item Details	partsCatalogItem	user team 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 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 Field Service Core Application and opens the plug-in
- 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 plug-in API and are ignored if they're sent with the 'close' message.

Order of Applying Changes to Entity Data Collections

If a plug-in 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 plug-in 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 plug-in. 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 plug-in 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 plug-in. 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
 }
 }
}
```

Availability of activity, inventory, and resource properties depends on the configuration of the plug-in. 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 plug-in. 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

Field	Type	Example Value	Description
uname	String	Admin	Name

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

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

Field	Description
customer_number	Account Number
c_zid	Time Zone
cphone	Phone
cemail	Email
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

Field	Description
atime_of_assignment	Activity Time of Assignment

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

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 plug-in 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

ready Method

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

The plug-in sends the *ready* message every time it's loaded. When Oracle 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 plug-in sends the *ready* message. If a plug-in doesn't send the *ready* message within 120 seconds, Oracle Field Service Core Application marks the plug-in as 'Failed to init' and shows the corresponding message to the user ("This plugin has not been loaded: ..."). If a plug-in is marked as 'Failed to init', Oracle Field Service Core Application tries to re-initialize it when the user opens the plug-in by clicking a button. When a user opens the plug-in through a button, the message, 'Screen is loading. Please wait.' is shown. This message is shown until the plug-in sends the *ready* message. If a plug-in doesn't send the *ready* message within 120 seconds, Oracle Field Service Core Application displays the message, 'The plug-in 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 Field Service Core Application has loaded the plug-in while initializing (not when the user opens the plug-in), Oracle Field Service Core Application sends an additional *init* message with initialization data to the plug-in such as attribute description. It destroys the plug-in's iframe only when the plug-in sends the *initEnd* message. If you have not set the field 'sendInitData' or set it as not equal to true, and Oracle Field Service Core Application loads the plug-in while initializing Oracle Field Service Core Application (not when the user opens the plug-in), Oracle Field Service Core Application destroys the plug-in's iframe immediately after the ready message received. If the user opens the plug-in, the field 'sendInitData' is ignored.

Amount of Data Sent to Plug-In

Oracle Field Service Core Application sends the *open* message with all data available for entity collections, according to the context layout, where the plug-in's button is placed and depending on how the Available Properties section is configured. For example, if a plug-in 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 Field Service Core Application collects and serializes the data, and the plug-in un-serializes it, thereby increasing the loading time for the

plug-in. 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 plug-in can prevent Oracle 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 plug-in. This set is predefined and depends on the page from which the plug-in 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 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 plug-in. Here is the list of available keys and data subsets:

Key	Affected Collections	Description
queue	queue	Current queue state
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": [
 "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 plug-in 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,

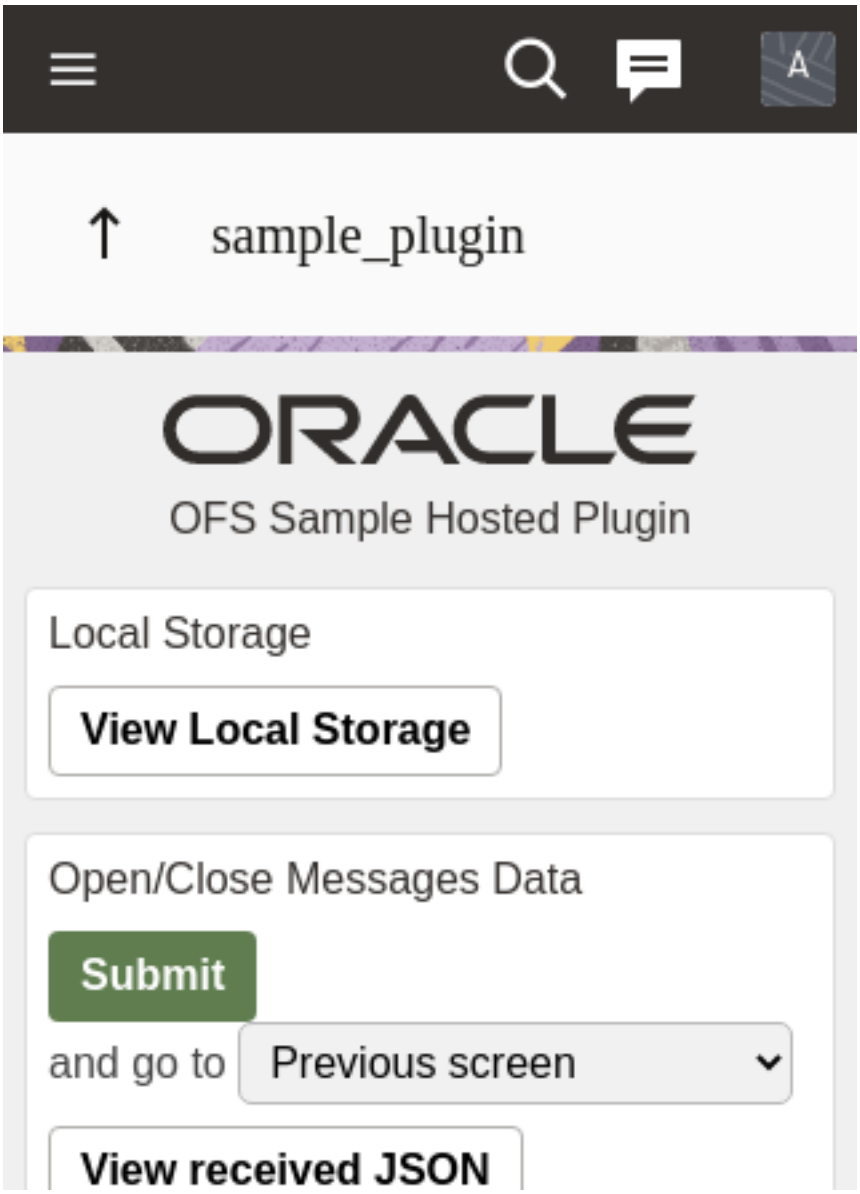
the parameters "showHeader" and "enableBackButton" are available for the ready method. Here are the details of the “showHeader” and “enableBackButton” parameters:

Param Name	Mandatory	Default Value	Type	Description
enableBackButton	No	true	boolean	If the flag value is set to true, then the Oracle 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 Field Service Core Application "< Back" button is hidden and the navigation is 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 Field Service Core Application header is shown. If the flag value is set to false, then the header is hidden.

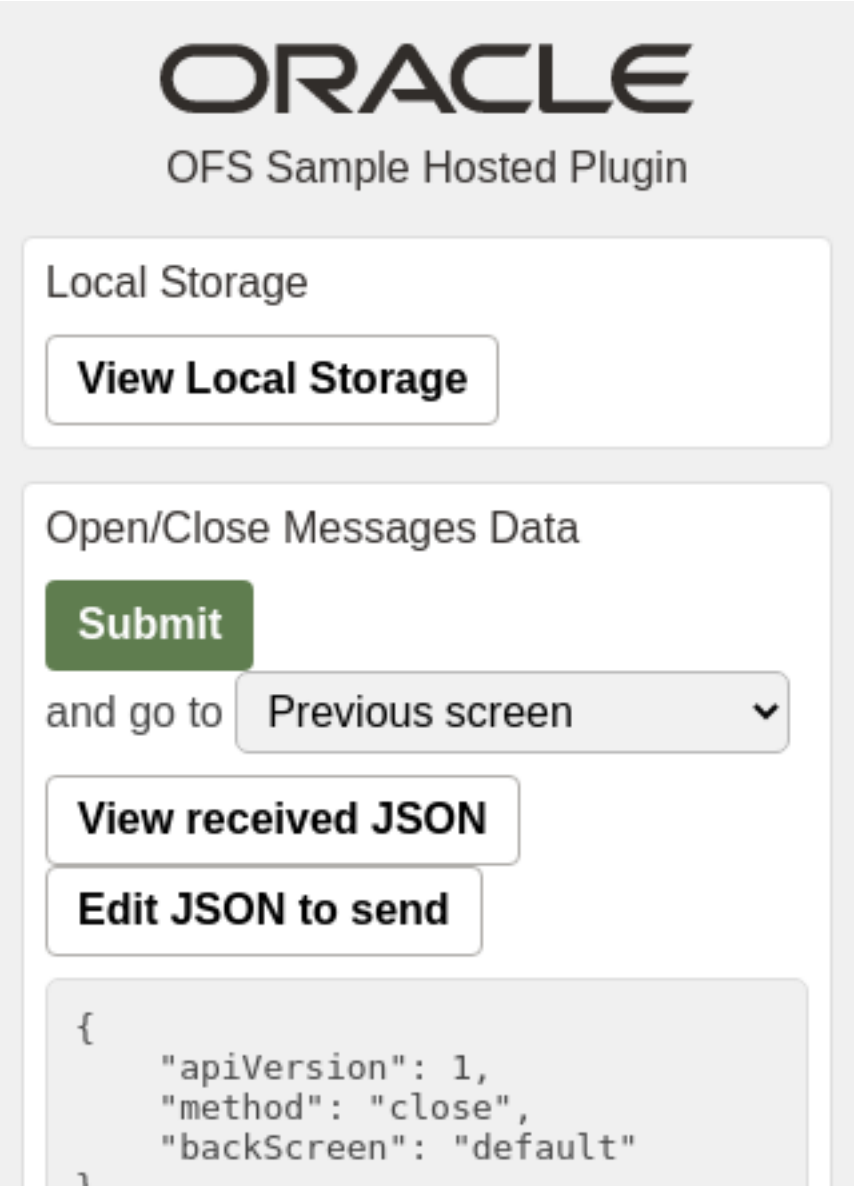
Users can be redirected to a specific page after the plug-in is closed. See the *close* method for details. This table shows the behavior of the plug-in for different values of the showHeader param.

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

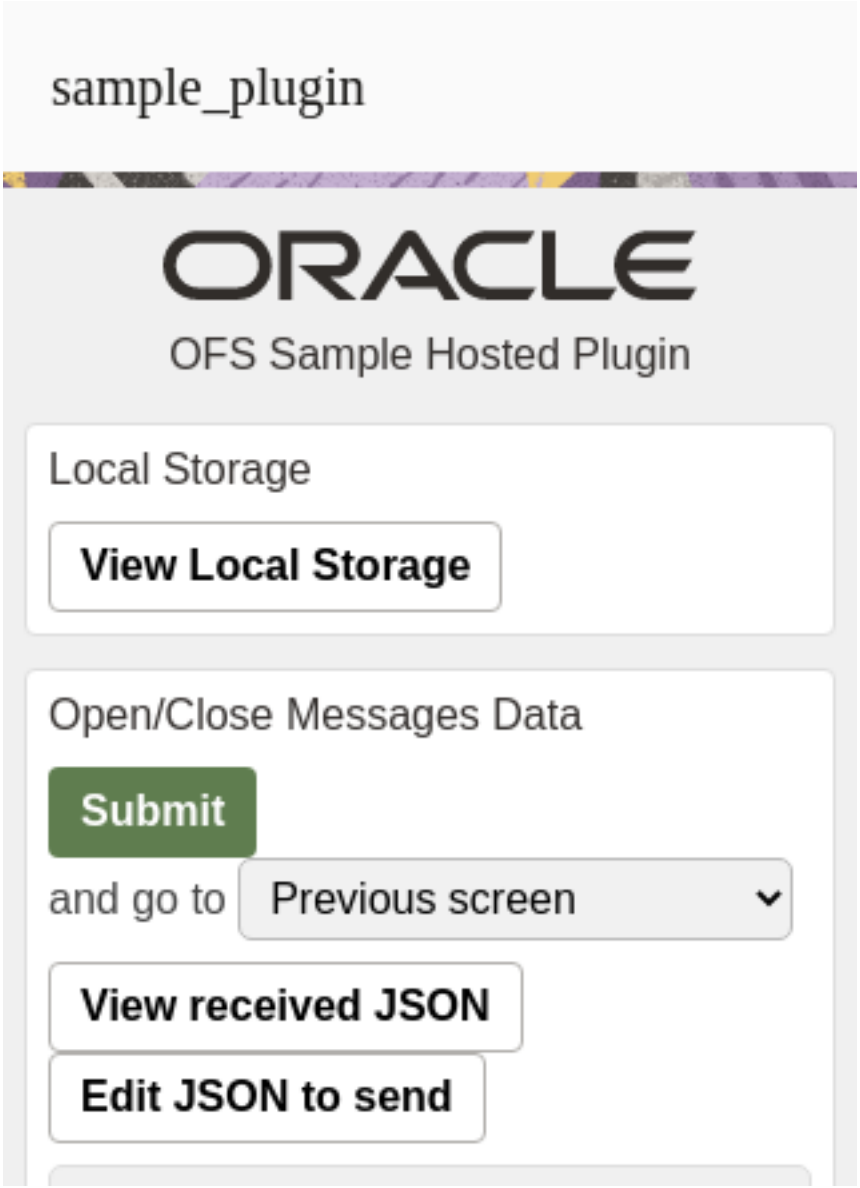
This screenshot shows the scenario when enableBackButton is True and showHeader is True.



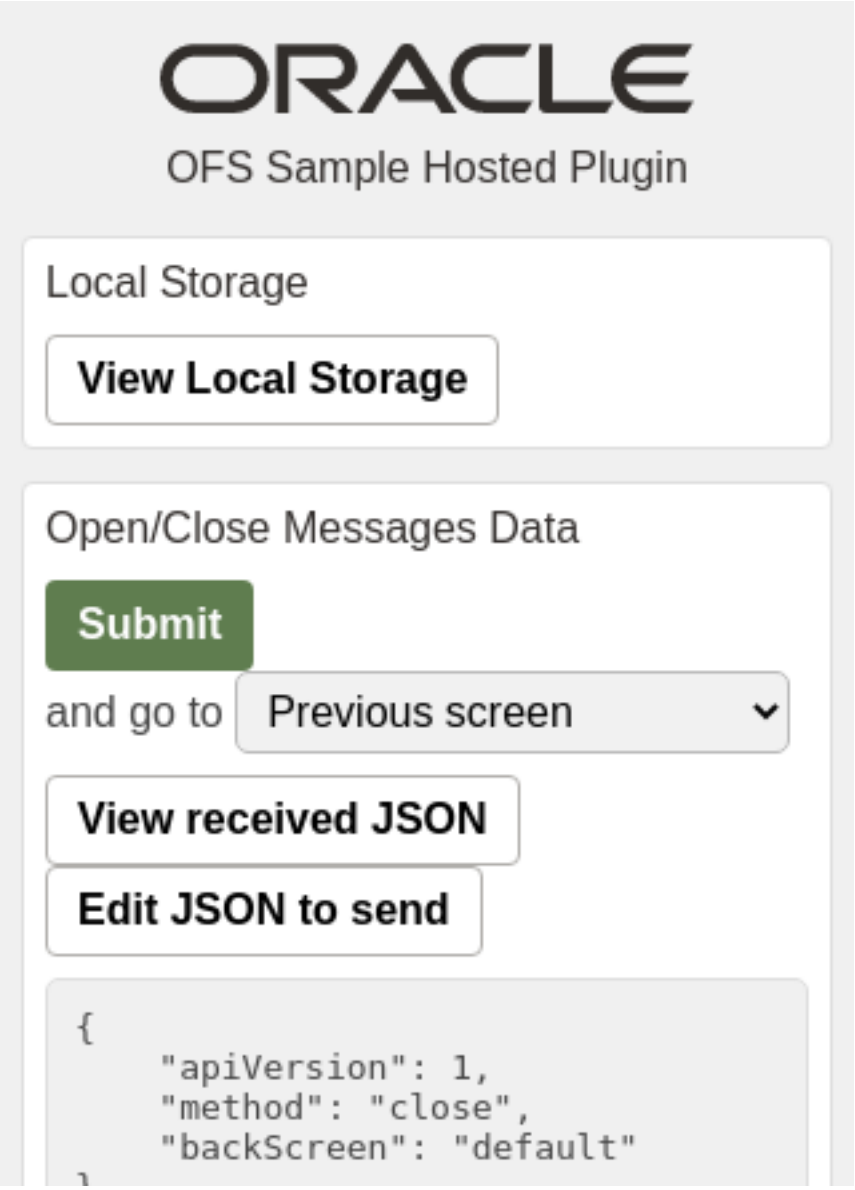
This screenshot shows the scenario when enableBackButton is True and showHeader is False.



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
}
```

Related Topics

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

Available Fields for the 'queue' Entity Collection

You can implement the activate and deactivate route functions in your custom plug-ins using the Plug-in API framework. You can activate and deactivate the route and get the state of the currently selected route. You can use plug-ins instead of the standard Activate/Deactivate dialog box. You can implement scenarios such as gathering data before activating and deactivating the route.

This table lists all the available fields for the 'queue' entity collection:

Field	Example Value	Description	Mandatory (Yes/No)
date	2021-02-03	Date of the queue. Is a key field for the queue.	Yes
status	notActivated	Status of the queue. Possible values: • notActivated - the queue hasn't been activated or deactivated yet. • activated - the queue has been activated • deactivated - the queue has been deactivated	Yes
isActual	true	Boolean, true if the current queue can be operated (not archived, not in future)	Yes
activationTime	2021-02-03 04:05:06	Date and time of queue activation. Present if the queue has been activated.	No
deactivationTime	2021-02-03 04:05:06	Date and time of queue deactivation. Present if the queue has been deactivated and not activated after.	No

Field	Example Value	Description	Mandatory (Yes/No)
reactivationTime	2021-02-03 04:05:06	Date and time of repeated queue activation (re-activation). Present if the queue has been activated after deactivation.	No

Examples

```
"queue": {
 "date": "2021-02-03",
 "status": "notActivated",
 "isActual": true
}
```

```
"queue": {
 "date": "2021-02-03",
 "status": "activated",
 "isActual": true,
 "activationTime": "2021-02-03 04:05:06"
}
```

```
"queue": {
 "date": "2021-02-03",
 "status": "deactivated",
 "isActual": false,
 "activationTime": "2021-02-03 04:05:06",
 "deactivationTime": "2021-02-03 06:07:08",
 "reactivationTime": "2021-02-03 05:06:07"
}
```

Supported Queue Actions

Here are the actions supported for the queue key.

activate_queue

This action activates the currently selected queue. If the queue is not in the present (it is in the past or future), an error is shown. If the queue is deactivated, the queue gets re-activated and the deactivation time is truncated. This table describes the parameters supported for the activate_queue action:

Parameter	Mandatory	Type	Description
action_time	no	String	Time and date of the action in current resource's time zone. If not set - the current time is used (on the moment of close message processing). Date/time in ISO 8601 format with possible forms: • YYYY-MM-DD hh:mm:ss

Parameter	Mandatory	Type	Description
			• YYYY-MM-DD hh:mm • hh:mm:ss • hh:mm If the "date" part is omitted then the current resource's date is used.

Examples of the "close" message

Without specifying the action time:

```
{
 "apiVersion": 1,
 "method": "close"
 "actions": [
 {
 "entity": "queue",
 "action": "activate_queue"
 }
 ]
}
```

Specifying the action time:

```
{
 "apiVersion": 1,
 "method": "close"
 "actions": [
 {
 "entity": "queue",
 "action": "activate_queue",
 "action_time": "2021-01-02 03:04:05"
 }
 ]
}
```

deactivate_queue

This action deactivates the current selected queue. If the queue is not in the present (it is in the past or future) or the queue has started or pending activities, an error is thrown. This table describes the parameters supported for the deactivate queue action:

Parameter	Mandatory	Type	Description
action_time	no	String	Time and date of the action in current resource's time zone. If not set - the current time is used (on the moment of close message processing). Date/time in ISO 8601 format with possible forms:

Parameter	Mandatory	Type	Description
			• YYYY-MM-DD hh:mm:ss • YYYY-MM-DD hh:mm • hh:mm:ss • hh:mm If the "date" part is omitted then the current resource's date is used.

Examples of the "close" message

Without specifying the action time:

```
{
 "apiVersion": 1,
 "method": "close"
 "actions": [
 {
 "entity": "queue",
 "action": "deactivate_queue"
 }
 ]
}
```

Specifying the action time:

```
{
 "apiVersion": 1,
 "method": "close"
 "actions": [
 {
 "entity": "queue",
 "action": "deactivate_queue",
 "action_time": "2021-02-03 04:05:06"
 }
 ]
}
```

Error Codes for Queue Actions

You can encounter validation, execution, or internal type of errors when your plug-in 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 plug-in to be updated in the "properties" param, has an invalid value	• actionId • entity • entityId • propertyLabel

Execution Errors

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

If the validation passes successfully, but the entity collection update or some actions fail, then you get internal errors. This means that the entity collection update may not be applied for the actions that didn't cause errors. In this case, the entities are in a state that is unknown to the plug-in, 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 Field Service is unable to process the message due to an unexpected change of the system 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.	

This table lists the error codes that are available for queue-related actions such as activate queue and deactivate queue.

Code	Caused by Action	Cause
TYPE_ACTION_ERROR		
CODE_ACTION_ON_PAST_DATE_NOT_ALLOWED	• activate_queue • deactivate_queue	A plug-in attempts to activate or deactivate a queue that is in the past (archived).
CODE_ACTION_ON_FUTURE_DATE_NOT_ALLOWED	• activate_queue • deactivate_queue	A plug-in attempts to activate or deactivate a queue that is in future.
CODE_ACTION_STARTED_ACTIVITY_EXISTS	• deactivate_queue	A plug-in attempts to deactivate a queue that has a started activity.
CODE_ACTION_ENROUTE_ACTIVITY_EXISTS	• deactivate_queue	A plug-in attempts to deactivate a queue that has an en route activity.
CODE_ACTION_PENDING_ACTIVITY_EXISTS	• deactivate_queue	A plug-in attempts to deactivate a queue that has one or more scheduled pending activities.
CODE_ACTION_UNKNOWN	—	"action" param is not equal to any of the supported queue actions.
CODE_ACTION_ENTITY_UNKNOWN	—	"entity" param is not equal to "queue".
TYPE_ACTION_PARAM		
CODE_ACTION_PARAM_VALUE_INVALID	• activate_queue • deactivate_queue	"action_time" has incorrect format.
TYPE_INTERNAL		
CODE_UNKNOWN	• activate_queue • deactivate_queue	Oracle Field Service is unable to process the message due to an unexpected change of the system state.

Example

```
"resource": {
 "currentTime": "2021-02-03 04:05:06",
 "deviceUTCDiffSeconds": 42,
 "timeZoneDiffSeconds": -18000
}
```

close Method

The message format for the *close* method is similar to the *open* method. When Oracle Field Service receives a message from a plug-in 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 plug-in. If there are violations, Oracle 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 plug-in can send only those entities and properties that have to be updated. The remaining entities and properties are left unchanged. Oracle Field Service Core Application updates only those entity collections that were sent with the *open* message. If the plug-in sends extra entities with the *close* message, they're ignored. Oracle Field Service Core Application updates only the properties that are configured for the plug-in. If the plug-in 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 Plug-In 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 plug-in to be updated, has an invalid value or violates a business rule for the given entity and conditions	• entity • entityId • propertyLabel
TYPE_INTERNAL	Oracle 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 plug-in tries to update the activity properties that are in the past and overnight or overtime limit is elapsed • The plug-in 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: 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	Occurs When
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_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 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 plug-in through the Plugin API for update. If a value length exceeds the limit, Oracle 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 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 plug-in 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 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 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:
 {
 ccity: 'Cleveland',
 door_photo:
 {
 fileName: 'DCIM_20170425_203115.jpg',
 fileContents: file
 }
 }
 },
 targetOrigin
);
```

Sending Generated Content

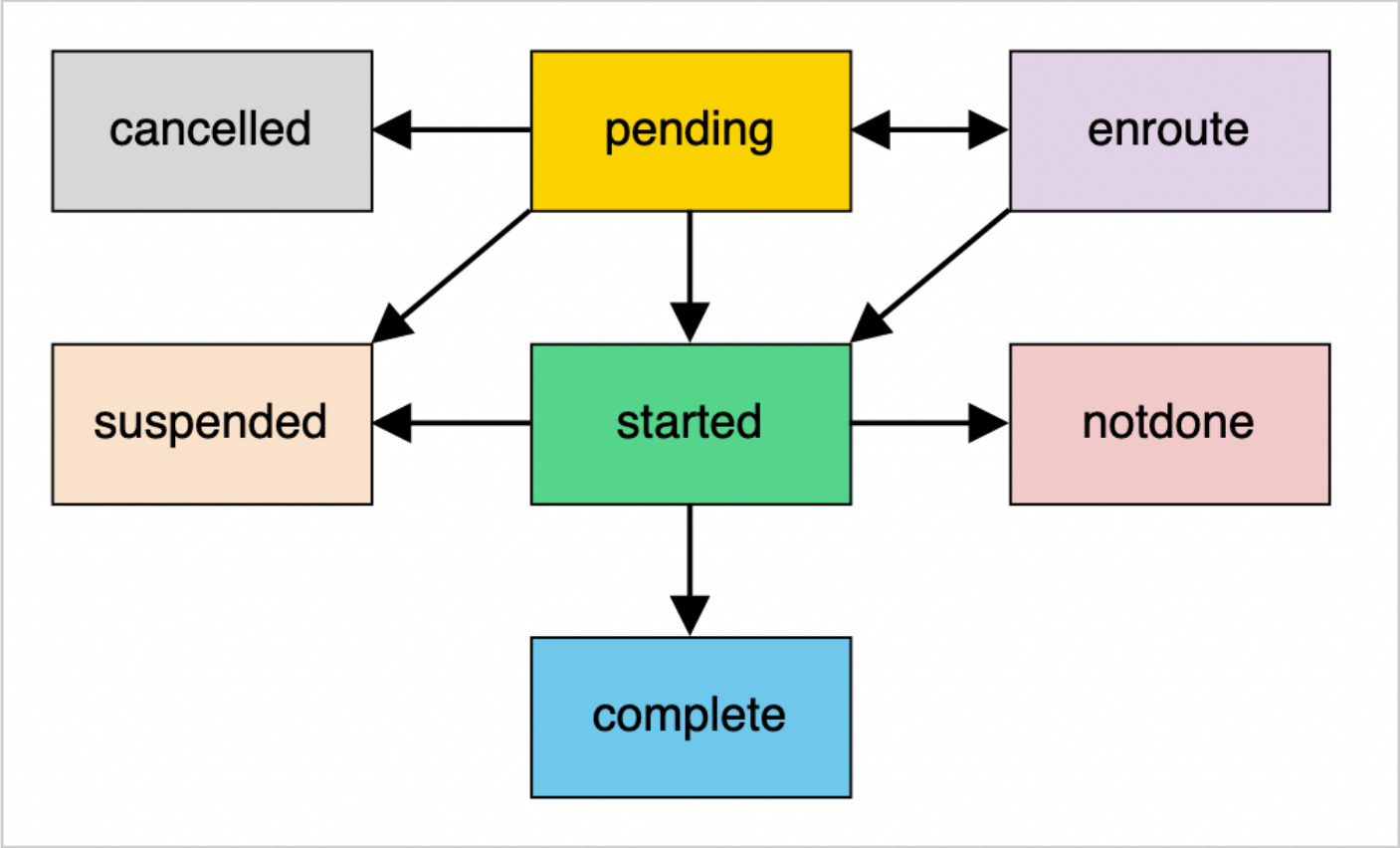
```
var text =
 '<?xml version="1.0" encoding="UTF-8"?>' +
 '<test></test>';

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
  );
```

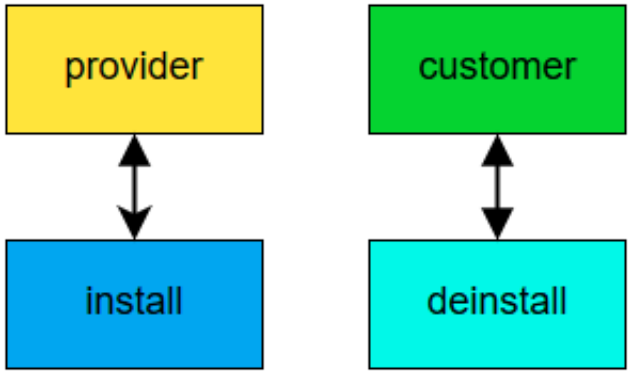
Activity Status and Inventory Pool Changes

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).

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 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.

Term	Description
	• 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 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 requirements:

- Is a Number or a String for which the parseFloat() function returns the Number (not NaN)
- Is less than 9999999999.99999
- Is greater that -9999999999.99999 for "inventory" and "inventoryList" collections
- Is greater that 0 for inventory actions

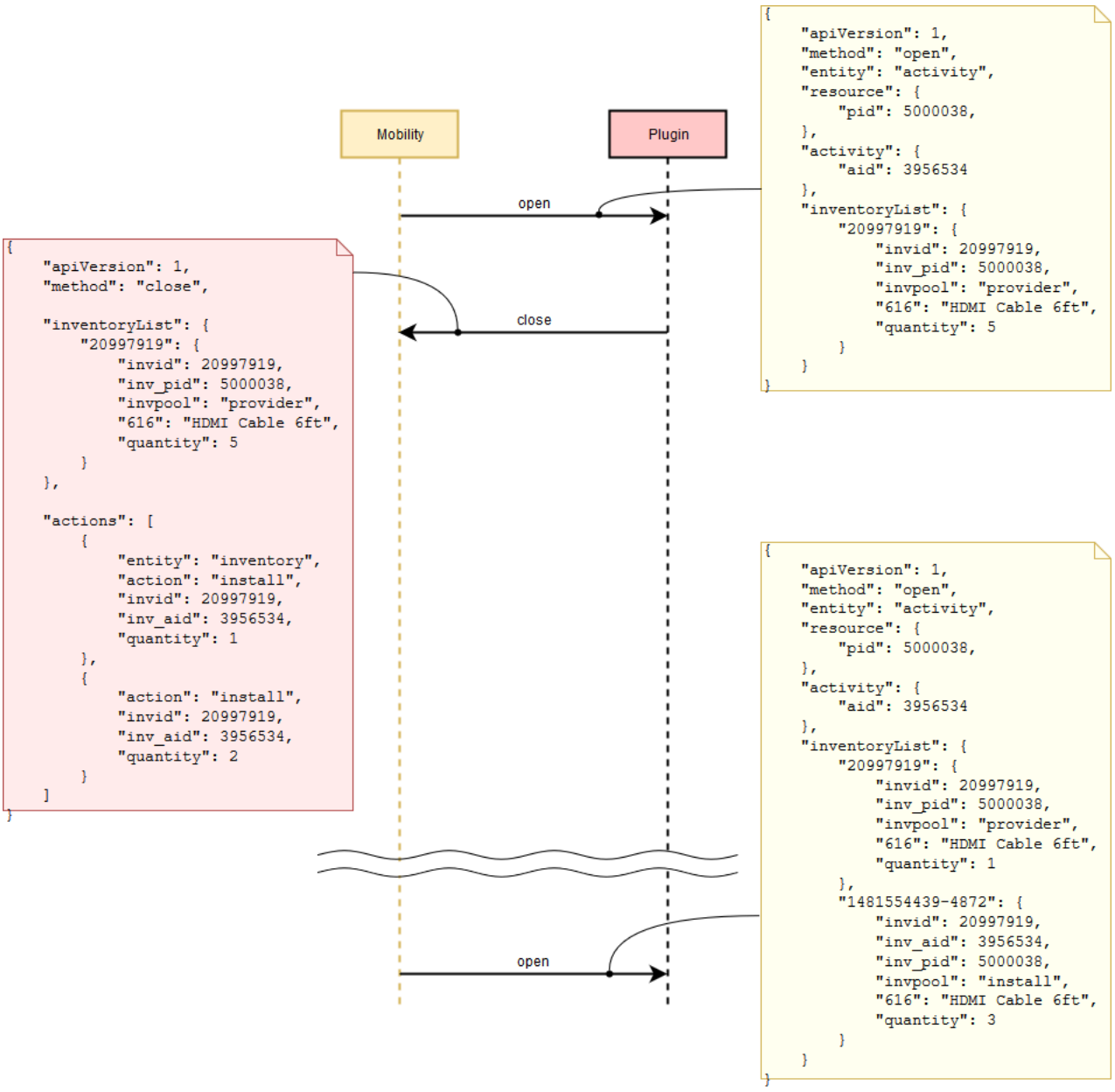
If these requirements are not satisfied, these error messages are shown:

- 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 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 plug-in in the "actions" array.

Actions are run after applying all the data collection updates that are sent by plug-in 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 plug-in within the *error* message. If some actions fail to run, the remaining actions are applied anyway, and the

processing errors are sent to the plug-in 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 plug-in.

Here is how Oracle 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 plug-in; 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 plug-in; if not, proceed to the next step.
7. Close the plug-in 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 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:

```
{
 language js
 theme Eclipse

 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 Field Service inventory properties to be updated. Properties are validated and processed according to the plug-in 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 Field Service inventory properties to be updated. Properties are validated and processed according to the plug-in 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. Is forbidden for serialized inventory.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Field Service inventory properties to be updated. Properties are validated and processed according to the plug-in 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 Field Service inventory properties to be updated. Properties are validated and processed according to the plug-in 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 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".

Parameter	Mandatory	Type	Description
			Is forbidden for invpool equal to "provider".
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 plug-in, then it is set to "1" for non-serialized inventory by Oracle Field Service Core Application itself.
properties	No	Object	Is a key-value object, where keys are the labels of Oracle Field Service inventory properties to be updated. Properties are validated and processed according to the plug-in 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",
 "inventoryList": {
 "21064435": {
```

```
"invtype": "RJ45",
"quantity": 75
},
"21258549": {
"invtype": "KPT-8",
"quantity": 0.12937
}
},
"actions": [
{
"entity": "inventory",
"action": "install",
"quantity": "2.71",
"inv_aid": "4224031",
"invid": "21258549",
"properties": {}
},
{
"entity": "inventory",
"action": "create",
"inv_pid": "3000001",
"invtype": "HD12",
"quantity": "5",
"invpool": "provider",
"properties": {}
}
]
}
```

Error Types for Inventory Actions

You can encounter validation, run, or internal type of errors when your plug-in 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 plug-in 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 plug-in, 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 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. Both actions and entity collection updates may lead to such errors.	NA

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 plug-in. 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".

What are the error codes displayed for inventory actions?

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

Code	Caused by Action	Cause
TYPE_ACTION_ERROR		

Code	Caused by Action	Cause
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_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 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"

Code	Caused by Action	Cause
		• "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 • 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 an invalid MIME-type • Property type is 'enumeration', and its value is not a valid enumeration item's index
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

Code	Caused by Action	Cause
		• "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_UNKNOWN	• install • deinstall • undo_install • undo_deinstall • create	Oracle 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 plug-in API support?

Supported Activity Actions

You can create scheduled and non-scheduled activities through custom plug-ins. For example, a custom plug-in 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 plug-in 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 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.
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 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 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.

Parameter Name	Mandatory	Type	Description
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).
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 plug-in 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).

Code	Caused by Action	Cause
		• "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 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_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

Code	Caused by Action	Cause
		• "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 • 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 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 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 plug-in 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 plug-ins 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 plug-in perform any operation on activities created on the client-side (using the "create" action of the plug-in API or the standard Add activity page), both in offline in online, the plug-in 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 plug-in may generate a temporary id for activities on its own and send it in the "temporaryAid" param of the "create" activity action. The plug-in may store this value and use it to reference a created activity in later acctions not even knowing the actual aid.
- The plug-in 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 Field Service Application closes the plug-in and redirects the user to another page. By default, the user is redirected to the same page from which the plug-in 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".
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.
activity_list	None	None	Activity List page
start_activity	backActivityId	None	Start Activity page
end_activity	backActivityId	None	End Activity (Complete) 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.

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

Additional Parameters for Plug-in on Redirection

List of all buttons that are configured for a plug-in is sent to the plug-in 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 plug-in using the *backScreen* param of the *close* message. With this, you can open the plug-in in various states or show different pages of the plug-in depending on the context. You can also use redirection to implement strict business flows using several plug-ins. Here are some examples:

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

Navigation flow from plug-in A to plug-in 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 plug-in B. backpluginOpenParams is an object, which contains the data needed by plug-in B.
- After successful processing of the close message, plug-in 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 plug-in 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 plug-in A

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

Example of the open message received by plug-in 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 plug-in.

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

How do I use the update method in the Plug-In API?

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

Oracle 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 plug-in multiple times before closing. However, the plug-in cannot send the next *update* message until the previous *update* is applied and the *updateResult* message is sent to the plug-in.
- The plug-in 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 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 plug-in 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 plug-in page won't be recreated and the plug-in 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 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 plug-in to work with actual data without having to close and reopen the plug-in page.

If plug-in 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"
}
```

sleep Message

For the description of the *sleep* method, see the *wakeup* method. The appearance of a plug-in 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.

```
{
 "apiVersion": 1,
 "method": "sleep",
 "wakeupNeeded": true,
 "iconData": {
 "color": "highlight"
 }
}
```

Related Topics

- Change the Plug-In Tile Appearance

wakeup Message

Oracle Field Service destroys a plug-in'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 plug-in is closed. However, the plug-in may have data, which must be synchronized with its server side, Oracle 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 plug-in synchronize its data as mentioned earlier. If *wakeupNeeded* is set to true, the Oracle Field Service Core Application opens the plug-in's hidden iframe in the background, as soon as Oracle Field Service Core Application is online, but no earlier than 10 seconds after the plug-in is closed. After the plug-in iframe is opened, Oracle Field Service Core Application sends the *wakeup* message to the plug-in, in response to the *ready* message. The plug-in then sends the *sleep* message back to Oracle Field Service Core Application, when it finishes synchronization. This lets Oracle Field Service Core Application destroy the iframe.

If the plug-in 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 Field Service Core Application opens the plug-in's iframe in the background again, as soon as Oracle Field Service Core Application is online, but no earlier than five minutes after the plug-in is closed. If the plug-in doesn't send the *sleep* message in two minutes (120 s) after the *wakeup* message is sent, Oracle Field Service Core Application destroys its iframe and reopens it again, as if the plug-in 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 plug-in synchronize even after refreshing Oracle Field Service Core Application or closing the browser. If the plug-in 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 Field Service Core Application opens the plug-in's iframe in the background, as soon as Oracle Field Service Core Application is online, but no earlier than five minutes after it receives the *initEnd* message.

The plug-in 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 Field Service Core Application doesn't detect the offline mode, when the plug-in is opened or closed. If the user opens the plug-in by clicking its button, the background iframe is destroyed without sending any messages to the plug-in. If the plug-in 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 plug-in'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 plug-in must be opened for background operation:

- online: If this field is set and is not null, the plug-in is opened in the background only when Oracle Field Service is online, as if *wakeOnEvents* field was not sent.
- timer: If this field is set and is not null, the plug-in 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 Field Service opens the plug-in in

Field	Type	Min Value	Max Value	Default Value	Description
					the background and sends a *wakeup* message. Oracle Field Service *wakes* a plug-in 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.
sleepTimeout	Number (integer)	10	3600	120	Duration in seconds, after which Oracle Field Service forcibly closes a background frame of the plug-in if it hasn't sent a *sleep* message explicitly. This period starts when Oracle Field Service sends the *wakeup* message to the plug-in.

If both *online* and *timer* are set, the plug-in is opened on the first event for which all conditions are met (*wakeupDelay* period has passed, Oracle Field Service is online (for the *online* event)).

If both fields have the same value for *wakeupDelay* and Oracle Field Service is online, then there's no guaranteed order of *wakeup* events.

The default value for *wakeOnEvents* is { online: {} }. That is, the plug-in 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 plug-in sends the *close*, *initEnd*, or *sleep* message. So if a plug-in 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 plug-ins 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 plug-in'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).

callProcedure Method

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

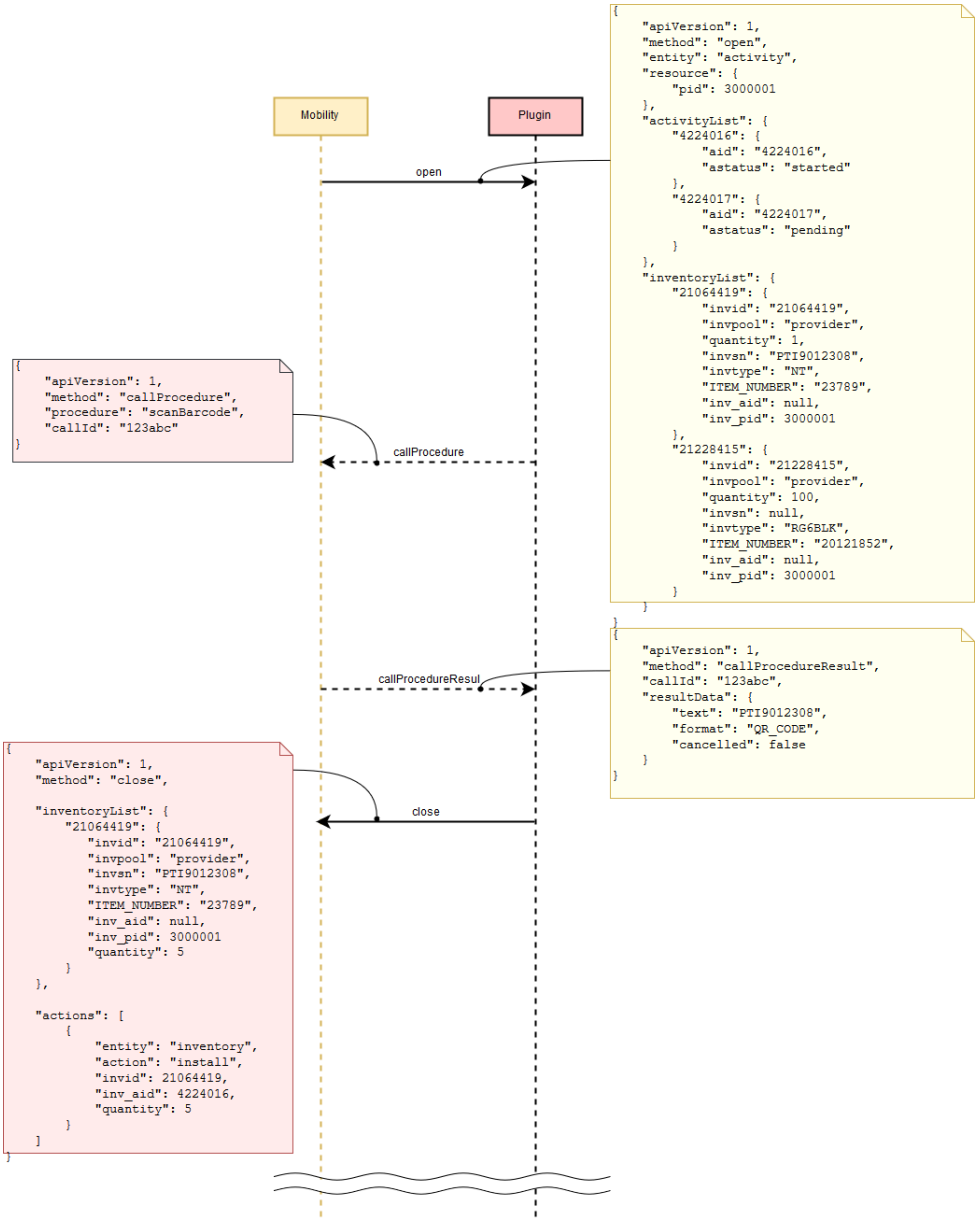
The *callProcedureResult* method returns the result of running the *callProcedure*. As you can run multiple procedures simultaneously, it is possible that the plug-in sends the next *callProcedure* messages to Oracle Field Service before it sends the *callProcedureResult* message back. These results 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 plug-in through the *callProcedure* method:

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

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 plug-ins that are opened in background after they receive a *wakeup* message. You can use only the *updateIconData* and *updateButtonsIconData* procedures with the plug-ins that are opened in background. These procedures help the plug-in 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 plug-in 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 plug-in through the *callProcedureResult* method. If the barcode scanner is unavailable or Oracle Field Service Core Application isn't run inside the Oracle Field Service Mobile (for Android or iOS) application, an error code is returned to the plug-in through an error message.

> **Note:** You must have the Oracle Field Service Mobile (for Android or iOS) application to use the Barcode Scanner through the plug-in 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
EAN_8	Yes	Yes
EAN_13	Yes	Yes

Barcode Type	Android	iOS
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
 }
}
```

Related Topics

- callProcedureResult Method

openLink Procedure

The openLink procedure provides a common way to open external URLs from Oracle Field Service Core Application run either in a web browser or in the Oracle Field Service Mobile for Android and iOS app. If Oracle Field Service Core Application is run in the Oracle 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

Field Name	Type	Parts Catalog API Parameter	Description
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 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.
inventoryType	String	inventory_type	Inventory type according to Oracle 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 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"
}
]
}
}
```


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 plug-in'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 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 plug-in'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 plug-in'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 Field Service. Can be empty if the mapping of item types to

Field Name	Type	Parts Catalog API Parameter for 'item' Element of 'upload_ catalog' Request	Description
			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
}
}
```


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 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

documentType	Dependent Field
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 plug-in to update the appearance of its button (or multiple buttons at once to make it look the same) on **My Route** without closing the plug-in 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 plug-in to update the appearance of its buttons individually without closing the plug-in 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": {}
 }
 }
 }
}
```

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 Field Service application or the Oracle 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 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 plug-in is allowed to send before it's closed. With this list, a plug-in may check the procedures that are available for the current device without calling them. This helps the plug-in disable its functions and/or user interface elements that depend on some procedures (for example, scanBarcode, which is available only in Oracle Field Service Mobile native app).

Only *updateIconData* and *updateButtonsIconData* are available when plug-ins work in the background. To consider a procedure as available, the plug-in 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_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_WAKEUP_PARAM	Value of *wakeOnEvents* field of *close*, *initEnd*, or *sleep* message is invalid.	• procedure • callId (if available)

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 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	Error Type	Cause
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_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.

Code	Error Type	Cause
		• 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.

callProcedureResult Method

A message with the *callProcedureResult* method is sent by Oracle Field Service to a plug-in when Oracle 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": "zs2vF8f7",
 "resultData": {
 "text": "PT9012308",
 "format": "QR_CODE",
 "cancelled": false
 }
}
```

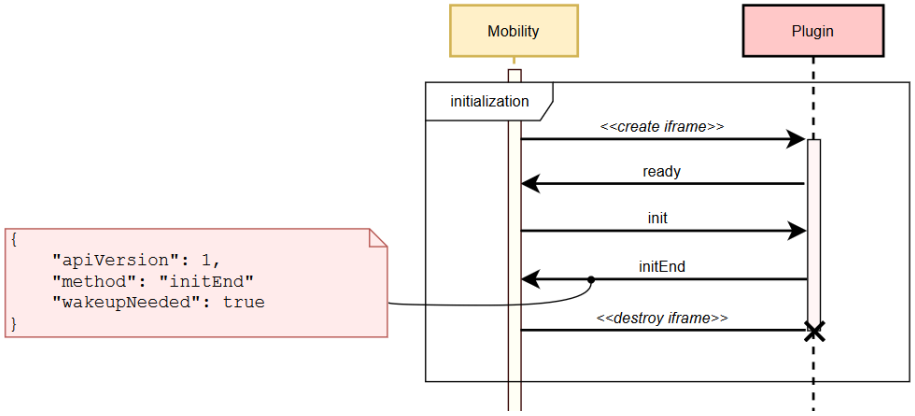
For more information about the possible responses, see the description of procedures in the callProcedure section.

Plug-In Lifecycle

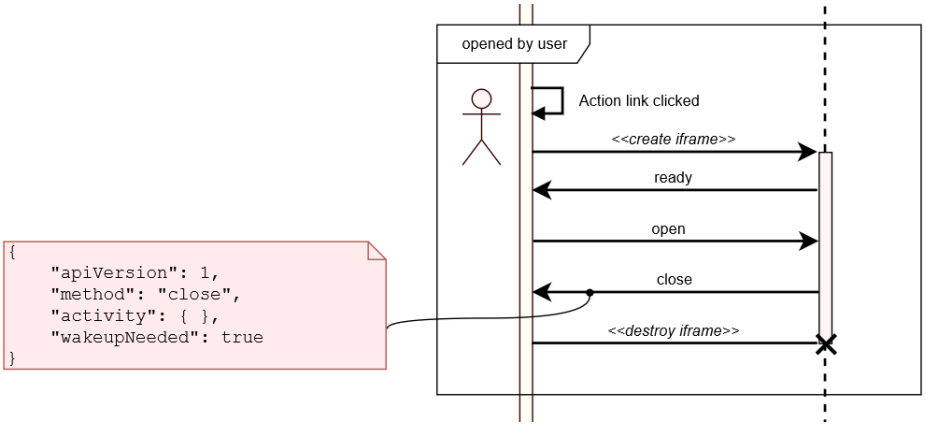
This topic provides the lifecycle diagram of a plug-in.

The plug-in lifecycle diagram is divided into four parts:

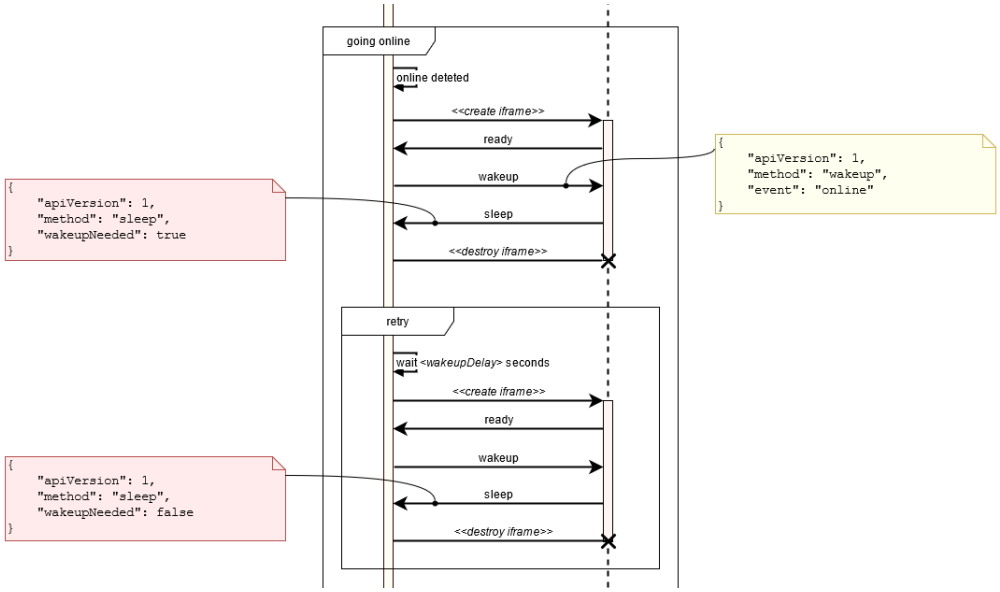
- Plug-in is initialized: This diagram shows the initialization of the plug-in:



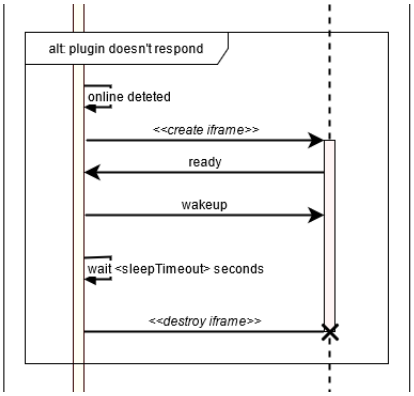
- Plug-in is opened by a user: This diagram shows the flow when a user opens the plug-in:



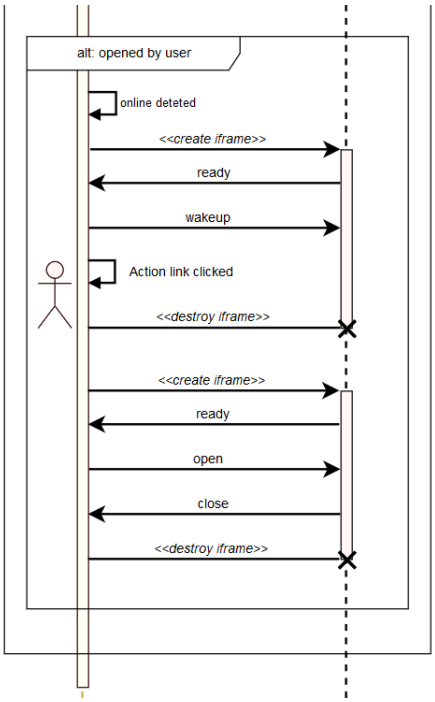
- Plug-in switches online with retries: This diagram shows the flow of background synchronization performed by the plug-in, when Oracle Field Service detects that the plug-in is online:



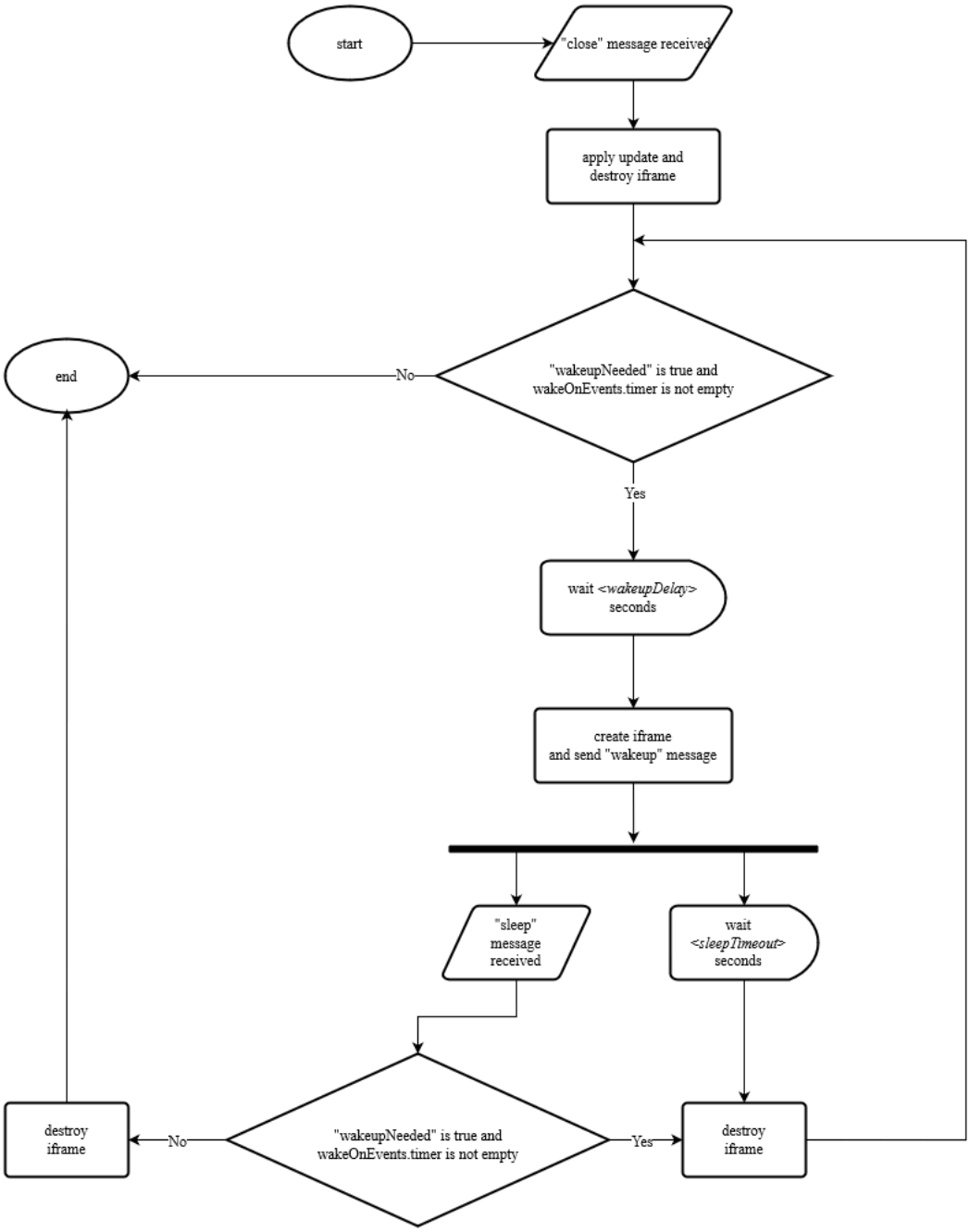
- Plug-in doesn't respond: This diagram shows the flow when the plug-in doesn't send the *sleep* message within two minutes:



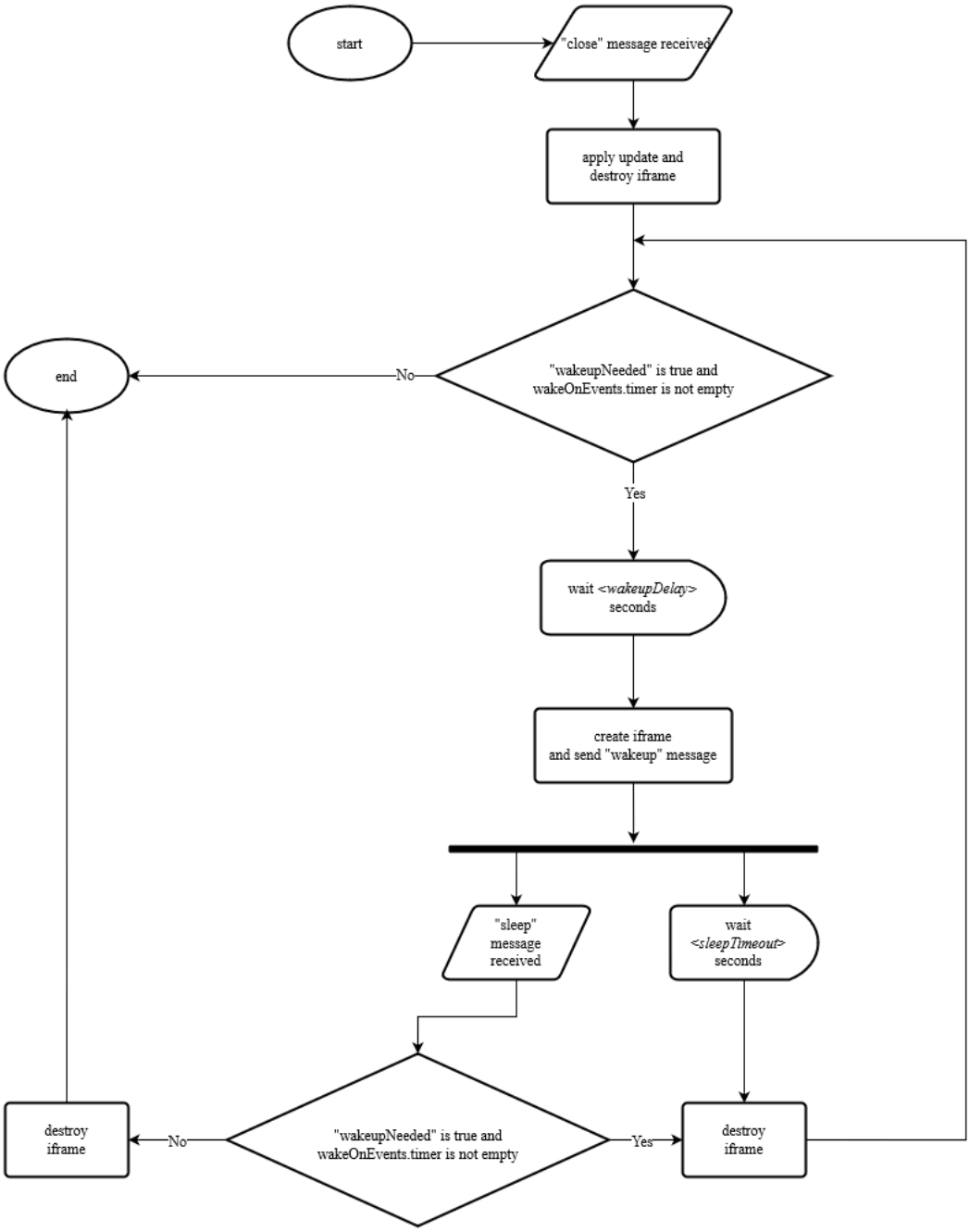
- Plug-in is opened by the user: This diagram shows the flow when the plug-in is opened by the user when the background synchronization is still in progress:



This flowchart shows the background synchronization of the plug-in for the *online* event:



This flowchart shows the background synchronization of the plug-in for the *timer* event:


start
"close" message received
apply update and destroy iframe
"wakeupNeeded" is true and wakeOnEvents.timer is not empty
No
end
Yes
wait <wakeupDelay> seconds
create iframe and send "wakeup" message
"sleep" message received
wait <sleepTimeout> seconds
"wakeupNeeded" is true and wakeOnEvents.timer is not empty
No
destroy iframe
Yes
destroy iframe

Avoid Cross-Domain Communication Blocking

To avoid cross-domain communication blocking when an Oracle Field Service API is called from a plug-in:

- For Oracle Field Service hosted plug-ins: When calling an Oracle Field Service API, the plug-in must use a plug-in 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 plug-ins: The plug-in must not call an Oracle Field Service API directly from the browser. Instead, the plug-in must call its server side. All the API calls must be performed by the server side and the call results transmitted to the plug-in.

3 Use a Standard Plug-In

Add a Standard Plug-In

A Standard plug-in is available out-of-the-box in Oracle Field Service and is supported by Oracle. It contains the logic that covers specific business scenarios (for example, Debrief) and can support integrations with other Oracle products such as Service Logistics. You cannot change a standard plug-in.

1. Click **Configuration** > **Forms & Plugins**.
   The **Forms & Plugins** page appears and displays the existing forms and plug-ins.
2. Click **Add Plugin**.
3. Click **Standard Plugin** and then click **Next**.
4. Click Debriefing or Sample Plugin.
   These sections are displayed:
   - **Properties will be installed.** These are the properties that are automatically installed with the plug-in. These properties will be available on the **Configuration** > **Properties** page. If you de-install this plug-in 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 plug-in and are currently present in Oracle 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 plug-in's documentation, find the property requirements, and change the property settings accordingly.
5. Click **Install** and confirm the installation.
   A message similar to, 'Debriefing Successfully installed.' is displayed on the **Forms and Plugins** page after the installation You can install a Standard plug-in only once. The **Install** button is disabled when you install the plug-in. Be aware that plug-ins are supported only in the English language.
6. To deinstall the plug-in that is not in use:
   a. Click **Configuration** > **Forms & Plugins**.
   b. Locate the plug-in that you want to deinstall.
   c. Click the actions icon and then click **Deinstall**.
      The plug-in is deinstalled and is no longer displayed on the **Forms and Plugins** page.

Modify the Settings of a Standard Plug-In

You can modify the settings of a Standard plug-in with some exceptions. You cannot change the required fields and the plug-in label. If a plug-in is designed to work with some specific properties and parameters, you cannot change them either.

1. Click **Configuration** > **Forms & Plugins**.
2. Search for the Standard plug-in 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 Standard Plug-In

You can change the code for a Standard plug-in to suit your business requirements and upload it back as a Hosted plug-in.

> **Note:** If you change the code for a Standard plug-in, the plug-in becomes your custom plug-in and it will no longer be supported by Oracle.

1. Click **Configuration** > **Forms & Plugins**.
2. Search for the Standard plug-in 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 plug-in back as a Hosted plug-in.
7. Follow the instructions in the Add a Hosted Plug-In topic and add the modified plug-in as a new hosted plug-in.
8. Add a button for the plug-in on the required page.
9. Open the plug-in and test your scenarios.

Install the Sample Plug-In

The Sample plug-in is available and accessible out-of-the-box within Oracle 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 Standard Plug-In topic and install the plug-in.
2. Verify the available features of the plug-in.
3. Add the required settings to the plug-in (properties, secure parameters, and so on).
4. Add a button for the plug-in on the required page.
5. Open the plug-in and test your business scenarios.
6. Learn the source code:
    a. Go to the Standard Plugins page and click Sample plug-in.
    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 plug-in either using the **Plugin archive** section on the **Edit Plugin** page, or through a REST API.
    g. Add a button for the plug-in on the required page.
    h. Open the plug-in and test your business scenarios.

7. Deinstall the plug-in if you don't need it for testing purposes.

Debrief Plug-In

Debriefing is the process of reporting time and materials used while performing an activity. Resources can use the Debrief button available on a started activity to add time, expense, or material information to an invoice report. They can save this information and obtain the customer's signature.

A field resource 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 Plug-In

Here are the high-level steps to configure the Debrief plug-in. You must follow the order given here strictly to configure the plug-in.

1. Install the plug-in.
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.

How do I install the Debrief plug-in?

Debrief is available as a Standard plug-in in Oracle Field Service. You can install it from the **Forms & Plugins** page.

You can add additional secure parameters to the plug-in, but you can't change the label, or available properties.

1. Click **Configuration** > **Forms & Plugins**.
   The **Forms & Plugins** page appears and displays the existing forms and plug-ins.
2. Click **Add Plugin**.
3. Click **Standard Plugin** and then click **Next**.

4. Click Debriefing.
   These sections are displayed:
   - **Properties will be installed.** These are the properties that are automatically installed with the plug-in. These properties will be available on the **Configuration** > **Properties** page. If you de-install this plug-in 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 plug-in and are currently present in Oracle 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 plug-in's documentation, find the property requirements, and change the property settings accordingly.

5. Click **Install** and confirm the installation.
   A message similar to, 'Debriefing Successfully installed.' is displayed on the **Forms and Plugins** page after the installation You can install a Standard plug-in only once. The **Install** button is disabled when you install the plug-in. Be aware that plug-ins are supported only in the English language. Further, the Debrief plug-in doesn't work in offline mode.
   Oracle Field Service creates the required properties automatically or notifies you that some existing properties will be used by the plug-in, 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 install the plug-in again. Here are the properties for resource, activity, and inventory entities that the plug-in 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 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.

Name	Label	Type	GUI	Description
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.
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

Name	Label	Type	GUI	Description
				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.
Expense Item Description	expense_item_ desc	Enumeration	Combobox	The description of expense subtypes. The indices must be the same as in the

Name	Label	Type	GUI	Description
				expense_item_numb er property. The values must describe the corresponding expense_item_numb er element.
Labor End Time	labor_end_time	String	Text	The time when a technician stops working on particular service activity. It must be not later than the end time of the work order (Oracle 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 technician starts working on a particular service activity. It must be not be earlier than the start time of the work order (Oracle 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.
Inventory ID	invid			Internal ID of the inventory.

Name	Label	Type	GUI	Description
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.
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.

6. To add your company logo in the Time & Labour 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 plug-in 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 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. Doublecheck 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 Plug-In to a Page

To make the plug-in 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 **Edit/View Activity** 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 *How do I add a plug-in to a page?* topic.
2. Be sure to add the visibility condition as `Activity status in (equal) Started`.

4 Use a Custom Plug-In

Types of Plug-Ins

You can add a hosted plug-in, an external plug-in, or an external application as a plug-in. You can also use the standard plug-ins that are shipped with Oracle Field Service.

Standard Plug-In: A standard plug-in is available out-of-the-box in Oracle Field Service and is supported by Oracle. It contains the logic that covers specific business scenarios (for example, Debrief) and can support integrations with other Oracle products such as Service Logistics. You cannot change a standard plug-in.

Hosted Plug-In: A hosted plug-in is hosted in Oracle Field Service and uses the Plugin API to interact with Oracle Field Service. This means, if your plug-in consists only of HTML, CSS, and JavaScript files and doesn't contain server-side files, then you can host it in Oracle Field Service. No additional hosting is required. The plug-in framework handles the communication between the hosted plug-in and Oracle Field Service.

External Plug-In: An external plug-in is hosted elsewhere and communicates with Oracle Field Service through the Plugin API. You add only a link to the plug-in here.

External Application: An external application can be added as a plug-in and it will be opened as a web page in a new window, or the same window within Oracle Field Service.

Add a Plug-In Archive

A plug-in archive is hosted in Oracle Field Service and uses the Plugin API to interact with it. The hosted plug-in can contain multiple pages and have its own navigation flow. This topic describes the high-level steps to configure and host a plug-in.

1. Determine whether you want to host the plug-in in Oracle Field Service. If yes, prepare your plug-in for upload.
2. Configure the plug-in.
    a. Upload the hosted plug-in.
    b. Add the available properties.
3. Add the plug-in to the required page.

How Plug-Ins are Hosted

If your plug-in consists only of HTML, CSS, and JavaScript files and doesn't contain server-side files, then you can upload it in Oracle Field Service. No additional hosting is required. The plug-in framework handles the communication between the hosted plug-in and Oracle Field Service.

You can host a maximum of 35 plug-ins per instance. This limitation does not include the Standard plug-ins.

The steps to host a plug-in are:

- Complete the prerequisites to upload the plug-in.
- Upload the plug-in.

After hosting a plug-in, you can:

- Use it on a page
- Move between instances
- Modify
- Rollback to a previous version
- Delete

Note: A hosted plug-in works only with Oracle Field Service Mobility Cloud Service.

Prerequisites to Upload a Plug-In

The plug-in must be in a specific format to be uploaded. If not, you cannot upload it, you must host it elsewhere.

The plug-in files must meet these requirements:

- You must upload a ZIP archive of the plug-in 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 plug-in files uploaded in Oracle 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 plug-in archive. If you don't want your code to be available without authentication, we recommend that you don't use the hosted plug-in functionality. Be aware that the communication between the plug-in and Oracle Field Service starts only when a user successfully logs in to Oracle Field Service.

Add a Hosted Plug-In

A Hosted plug-in is hosted in Oracle Field Service and uses the Plugin API to interact with Oracle Field Service. You can host a maximum of 35 plug-ins, excluding Standard plug-ins.

1. Click **Configuration** > **Forms & Plugins**.
   The Forms & Plugins page appears and displays the existing forms and plug-ins.
2. Click **Add Plugin**.
3. Click **Plugin Archive** 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 plug-in.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plug-in 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 plug-in is to be derived, if needed. When a base action is selected, the resulting plug-in functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plug-in. 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 plug-in, the plug-in 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 plug-in name in the English language. The action or plug-in appears under this name in the actual context.
Name (other languages)	Plug-in name translations to other languages, if used.
Plugin settings section	
Plugin archive	The zip file for the Hosted plug-in, which contains HTML, CSS, and JavaScript files. Click the field to browse and select file, or drag and drop a file.
Disable plug-in in offline	Determines whether you want to disable the plug-in when Oracle Field Service is offline. Clear this check box for the plug-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 Click **Add**. The parameter is added to the plug-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

Field Name	Description
	to Oracle Field Service during the next synchronization. The data is sent to the plug-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 to the plug-in or updated by the plug-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 o travel_estimation_method o service_window_end o service_window_start o eta_end_time o pid (it's still available for the Resource entity) o ctime_delivered_start o 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 located in the root of the archive.
   - The archive includes a maximum of 10 entries, including empty directories.

   If any of these conditions is not met, an error message is displayed and the archive is not uploaded.
   To be able to use the plug-in, you must add it to a button or a link. See the Add the Plug-In to a Page topic.

Working Offline

You can create the plug-in 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 plug-in's resources.

  b. Create the Service Worker javascript file (for example, service-worker.js). This file must implement the network behavior of your plug-in 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 plug-in, 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 plug-in archive, you might want to modify it, download it, or delete it.

1. To modify a hosted plug-in, 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 plug-in at any time. Whenever you upload a newer version of a plug-in:
   - 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 plug-in, 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 plug-in, first unassign it from all the buttons it is added to. Then, click **Delete** on the **Forms & Plugins** page.

   The plug-in is deleted with all its historical versions.
5. To move all the uploaded plug-ins between instances, export from the required instance using the **Export** function on the **Forms & Plugins** page. Import the exported files using the **Import** function in the target instance.
6. To move a single plug-in between instances, download it from the required instance. Upload it in the target instance. You can use the **Export** option here as well.

Add an External Plug-In

An external plug-in is hosted elsewhere and communicates with Oracle Field Service through the Plugin API. You add only a link to the plug-in 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 plug-in.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plug-in 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 plug-in is to be derived, if needed. When a base action is selected, the resulting plug-in functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plug-in. 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 plug-in, the plug-in 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 plug-in name in the English language. The action or plug-in 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 plug-in. This URL processes the HTML5 application and it runs the plug-in in the entire browser window. The URL must start with the protocol (https) and must point to the main file of the plug-in. Oracle Field Service adds the backUrl parameter to the URL automatically. This parameter contains the address of the current page of Oracle Field Service.
Disable button in Offline	Determines whether you want to disable the plug-in when Oracle Field Service is offline. Clear this check box for the plug-in to work in offline mode.
Authentication	The type of authentication used by the external server hosting the plug-in source to verify access to the plug-in. Select one of these options: o 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 plug-in content. o 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 plug-in. 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 plug-in. 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:

Field Name	Description
	○ 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 plug-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 Field Service during the next synchronization. The data is sent to the plug-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 to the plug-in or updated by the plug-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: ○ 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 ○ 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 plug-in and it will be opened as a web page in a new window, or the same window within Oracle 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 plug-in label.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plug-in 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 plug-in is to be derived, if needed. When a base action is selected, the resulting plug-in functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plug-in. 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 plug-in, the plug-in 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 plug-in name in the English language. The action or plug-in 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 plug-in.
URL	The path to a URL for the external plug-in. This URL processes the HTML5 application and it runs the plug-in in the entire browser window. The URL must start with the protocol (https) and must point to the main file of the plug-in. Oracle Field Service adds the backUrl parameter to the URL automatically. This parameter contains the address of the current page of Oracle Field Service.
POST Data	The data that you want to be sent to the external plug-in.
Disable button in Offline	Determines whether you want to disable the plug-in when Oracle Field Service is offline. Clear this check box for the plug-in to work in offline mode.
Open inside Field Service	Determines whether the plug-in uses the iframe layout. If the field is cleared, the plug-in's URL is opened in a new browser tab or window.
Show scrollbars on Dialog	Determines whether the window in which the plug-in 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 plug-in 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 plug-in.

Field Name	Description
Native application name	The name of the application to be launched by the plug-in.
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 Plug-In to Add to the Main Menu

You can add plug-ins that are created as HTML5 applications to the Main menu. You cannot add native application plug-ins.

1. Click **Configuration** > **Forms & Plugins**.
2. Click **Add Plugin**.
3. Complete these fields on the **Add Plugin** page:

Field Name	Description
General Information section	
Name (English)	A mandatory field defining the plug-in name in the English language. The action or plug-in appears under this name in the actual context.
Name (other languages)	Plug-in name translations to other languages, if used.
Label	A mandatory field defining a unique action or plug-in label.
Entity	Entity (activity, inventory, required inventory, resource, service request, user) to which the action or plug-in 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 plug-in is to be derived, if needed. When a base action is selected, the resulting plug-in functions per the same rules as the base action. The base action affects only the visibility of buttons and not the functioning of the plug-in. 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 plug-in, the plug-in 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 plug-in you want to use. Select HTML5 application. This means, the plug-in 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.

Field Name	Description
Fields for the **HTML 5 Application** Option	
Use Plug-in API	Determines whether you want the plug-in to communicate with Oracle Field Service using the Plug-in API. Clear this check box. When you do not use the Plug-In API, the URL is opened in a new tab, window, or iframe. To interact with Oracle Field Service you must pass some data (such as activity id, resource name) to the plug-in using the placeholders in the "POST Data" and "URL" fields.
URL	The path to a URL (for external plug-ins). This URL processes the HTML5 application and it runs the plug-in 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 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 plug-in. You can use only User entity fields as placeholders.
Disable plug-in in offline	Determines whether you want to disable the plug-in when Oracle Field Service is offline. Clear this check box for the plug-in to work in offline mode.
Open in iframe	Determines whether the plug-in uses the iframe layout. If you clear the field, the plug-in’s URL is opened in a new browser tab or window.

4. Click **Save.**
   If you set the external plug-in as the first item on the Main menu context layout, the menu item is displayed in the correct order. However, the plug-in does not open when a user logs in to the application. Instead, a standard page or a plug-in 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

Placeholder	User Property
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.

Change the Plug-In Tile Appearance

Use the *iconData* parameter to change the appearance of the plug-in button on the Landing page. You can change the status text, icon image, and color of the plug-in 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 plug-in is woken up when the user is on the Landing page, the icon is updated immediately after the plug-in 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 plug-in's button on the Landing page changes its color to get the user's attention.
image	Blob	o Max size: 64 KiB	Icon picture. It's recommended to use a scalable monochrome white icon in SVG format.

Field	Type	Limits	Description
		○ Allowed types: - image/svg+xml - image/png	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 plug-in'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 plug-in's URL. The placeholders are replaced with the values of the corresponding properties and are processed by the server that hosts the plug-in

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

Placeholder	Description
{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 plug-in'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 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 Field Service database.
Server Side	Configure the web server on which the plug-in sources are hosted to return the HTTP 401 Unauthorized status, if you are requesting the configured plug-in URL without the credentials. See the NGINX and Apache documents for details. The server must return the plug-in 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 plug-in in the **Add plugin** and **Modify plugin** pages must be accepted as valid.

Client Side	When the user logs in, Oracle Field Service reads the credentials from the database and loads the plug-in URL into the hidden iframe as follows: `<iframe src="https://mylogin:mypass@example.com/myPlugin.php"/>` This way, the browser loads the plug-in 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 plug-in in the **Add plugin** and **Modify plugin** pages in Oracle 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 Field Service on both, client and server sides. Examples of sensitive data include passwords and endpoints for Oracle Field Service Core API or external APIs.

Sensitive data is passed to the plug-in through the plug-in API in decrypted form, so the plug-in can access APIs or third-party services without having to store and secure the credentials. The plug-in stores the sensitive data in a JavaScript variable every time it receives a message from Oracle Field Service. Changes to this data are sent to Oracle Field Service during the next synchronization. This data is sent to the plug-in when the next message is sent. The plug-in 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 plug-in 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 Field Service Core Application to a plug-in, 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.

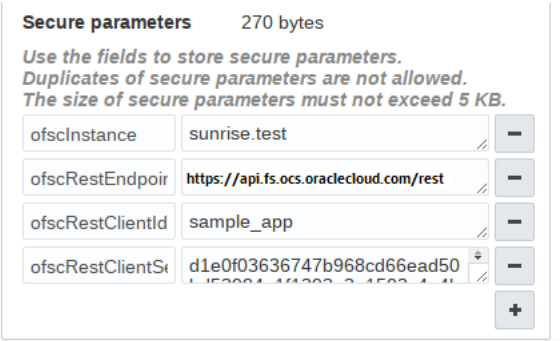
Format of *securedData*

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 plug-in 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"
 }
}
```

How do I add a plug-in to a page?

You add a plug-in to a context layout page, so that Field Resources can open it. You can configure the parameters for a button to send the parameters to the plug-in, or to open a specific page, or another plug-in.

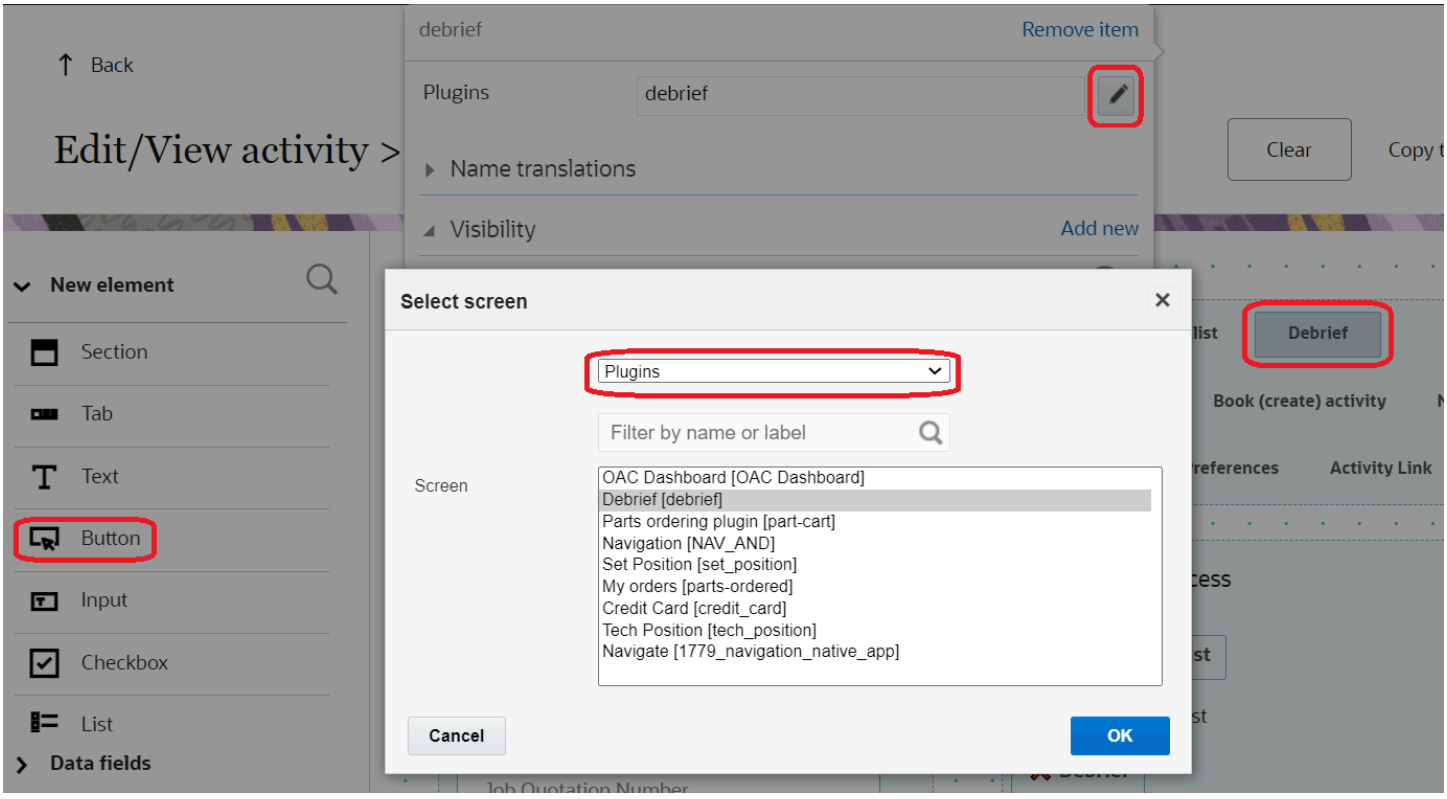
1. 1. Click **Configuration** > **User Types**.
2. Select the type of user for which you want to add the plug-in.
3. Click **Screen configuration**.
4. Find and click the page to which you want to add the plug-in.

   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 plug-in from the list.

5. Drag-and-drop the **Button** element to the section from where you want to invoke the plug-in.

> **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 plug-in that you want to open and click **OK**.
   The label of the plug-in is displayed in the **Plug-in** field. By default all plug-ins have a visibility of Read-only.
10. In the **Visibility** section, add the conditions based on which the plug-in is visible.
11. In the **Translations** section, add a name for the plug-in. [Optional]
    This name is displayed on the page from which the plug-in will be invoked. This screenshot shows the Visual Form Editor page where a plug-in is added to a Button element:



If you retain the default name and if you happen to change the name of the plug-in 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 plug-in. 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 plug-in. 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 plug-in.
13. Click **Save** on the **Visual Form Editor** page.

    The plug-in is added to the selected page.

Plug-In Buttons and Icons

Let's say you use a plug-in that contains several pages and you want to configure links that open different pages of the plug-in. Or, you want to configure different icons for different tiles of the plug-in on the Landing page and change the icon of the tile according to the data in the plug-in.

You can configure the plug-in for all these scenarios, using the Parameters option:

- Configure parameters for a button associated with a plug-in: You can specify custom parameters for each button of a plug-in. For example, when you click the button, you can open a specific page in the plug-in. You can implement this on multiple pages, by uploading a single plug-in 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 Field Service Core Application ('initEnd'), after closing ('close') a plug-in, or when a plug-in receives new data in the background mode ('sleep'). You can also associate each plug-in'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 plug-in.

How it Works

The data sent to the plug-in during initialization includes the list of all buttons configured for each plug-in with the key-value pairs of configured parameters. When a user opens the plug-in, the ID of the button is passed to the plug-in with the parameters configured for this button. When a user closes the plug-in, you can redirect the user to another plug-in, as if the plug-in has been opened through a button, and parameters that are configured for this button are passed to the plug-in.

Modify the Icons and Text of a Plug-In Tile

You can implement a flexible flow for the buttons of a plug-in 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 Field Service Core Application is opened ('initEnd'), after plug-in is closed ('close'), or when a plug-in 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 plug-in is sent to the plug-in 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 plug-in. openParams contains the parameters that are configured for this button.

If a plug-in is opened by sending the backScreen: "plugin_by_label" from another plug-in, 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 plug-in.
	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 plug-in.
	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

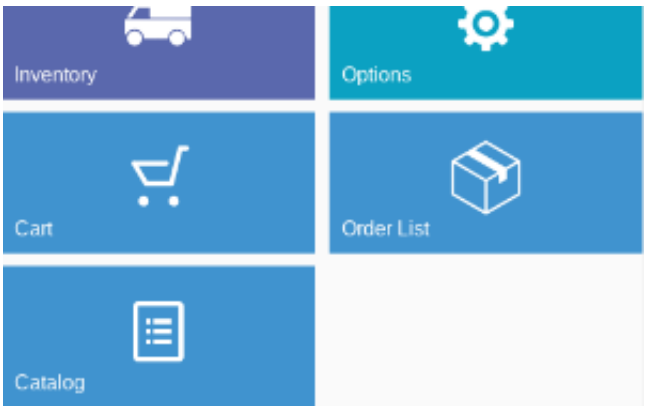
```
{
 "apiVersion": 1,
 "method": "close",
 "backScreen": "default",
```

```
 "wakeupNeeded": false,
 "buttonsIconData": {
 "17156": {
 "color": "highlight",
 "text": "123",
 "image": {}
 },
 "17155": {
 "color": "default",
 "text": null,
 "image": {}
 }
 }
}
```

Example of Changing Buttons in a Plug-In

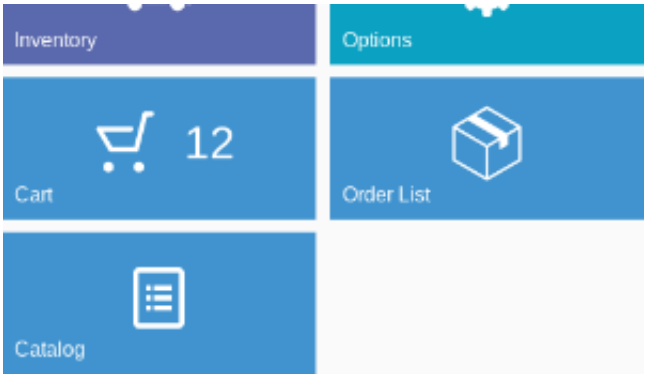
Consider the example of a catalog and a cart. The Order List Button changes its color when the supervisor approves the order.

1. There's a plug-in, which implements three pages: "Catalog", "Cart" and "Order List". Each button has a corresponding icon:

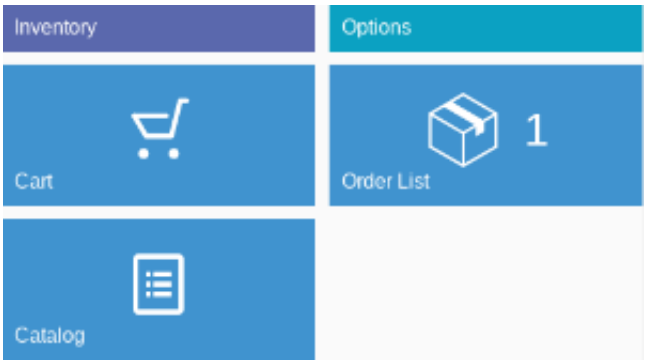


2. User clicks "Catalog" and the plug-in shows the list of items available for order.
3. User selects the items to order and closes the plug-in page.

4. The plug-in updates the "Cart", so that it shows the count of the items in the cart:

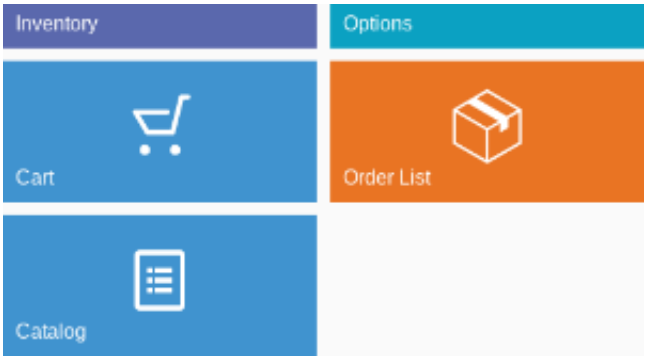


5. User clicks "Cart" and the plug-in shows the list of ordered items.
6. User confirms the order and closes the plug-in page.
7. The plug-in clears the counter on the "Cart" button and updates the "Order List" button so it shows the number of orders for approval:



8. The plug-in prompts Oracle 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 plug-in 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 plug-in.

Export and Import Plug-Ins

When you have plug-ins that work in a similar way, you can export the properties and configuration from one plug-in and import them into another. You can export or import multiple plug-ins 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 check box against the plug-ins that you want to export.
   A counter shows the total number of plug-ins available in the instance and the number you have selected.
5. Click **Export**.
   The selected plug-ins 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 plug-in configuration, click **Import** > **Plugins**.
7. Drag-and-drop the XML file that you have exported.
8. Click **Continue**.
   The plug-ins are validated and errors, if any, are displayed.
9. Fix the errors and import the file again.
10. Click **Apply**.
    The plug-ins 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	