Oracle Fusion Service

How do I use the UI Events Framework?

Oracle Fusion Service
How do I use the UI Events Framework?

F95864-02

Author: DYETTER

Contents

Get Help

There are a number of ways to learn more about your product and interact with Oracle and other users.

Get Help in the Applications

Use help icons to access help in the application. If you don't see any help icons on your page, click your user image or name in the global header and select Show Help Icons.

Get Support

You can get support at *My Oracle Support*. For accessible support, visit *Oracle Accessibility Learning and Support*.

Get Training

Increase your knowledge of Oracle Cloud by taking courses at *Oracle University*.

Join Our Community

Use *Cloud Customer Connect* to get information from industry experts at Oracle and in the partner community. You can join forums to connect with other customers, post questions, suggest *ideas* for product enhancements, and watch events.

Learn About Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program*. Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Share Your Feedback

We welcome your feedback about Oracle Applications user assistance. If you need clarification, find an error, or just want to tell us what you found helpful, we'd like to hear from you.

You can email your feedback to *oracle_fusion_applications_help_ww_grp@oracle.com*.

Thanks for helping us improve our user assistance!

1 Using the UI Events Framework

Getting Started with the UI Events Framework

The UI Events Framework is a JavaScript framework that you can use with a third-party application to interact with Service Center.

The framework has dedicated APIs to listen to events generated from the Service Center application and perform operations inside Service Center. The UI Events Framework manages these events and operations. For example, an input Text Field in Service Center can be updated from a third-party application through a UI Events Framework operation named setFieldValue. You can also subscribe to events such as field-value changes, so that whenever the field's value change occurs in Service Center, the callback function registered from the third party application is executed with the new and the old values of the field. More examples of what UI Events Framework can do follow.

Architecture Details of the UI Events Framework

UI Events Framework can:

- Subscribe to events from Service Center.

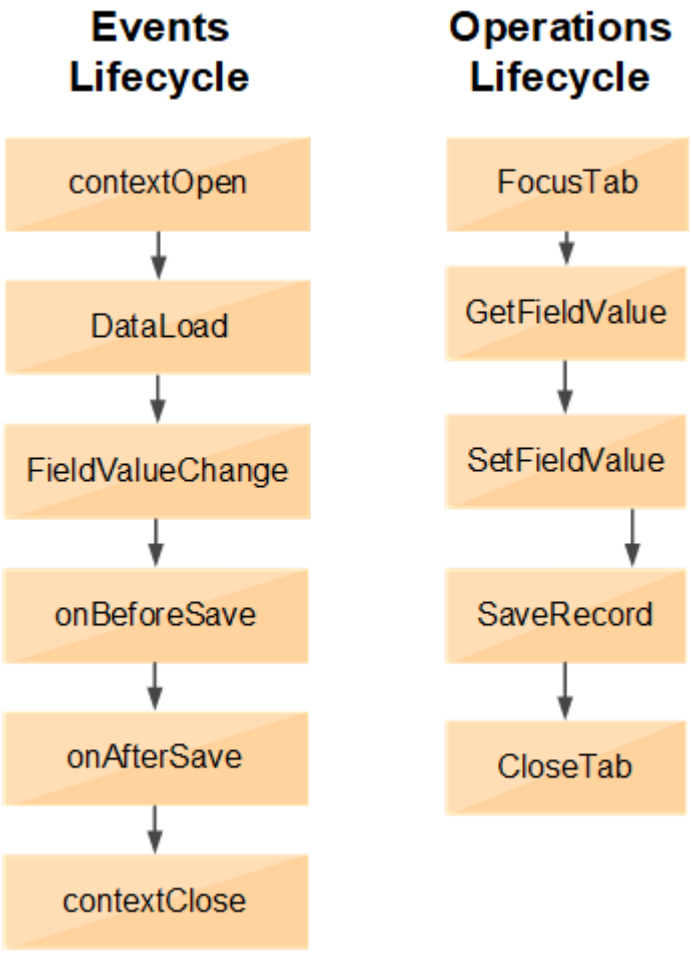
  Supported events include: TabOpen, TabClose, ContextOpen, DataLoad, FieldValueChange, OnBeforeSave, OnAfterSave, ContextClose, TabChange.
- Publish an operation to Service Center.

  Supported operations include: FocusTab, GetFieldValue, SetFieldValue, SaveRecord, CloseTab.

Understanding the UI Events Framework life cycle

Here's an overview of the Events life cycle.

The following graphic presents a life cycle overview:



TabOpen Event

TabOpen event subscription gives a notification about a browser tab or new window tab open event, or a new MSI tab open event happening in any already opened browser tabs with the Service Center application.

The event response will give the TabContext of the opened tab, on top of which you can call APIs to subscribe or publish APIs or any other tabContext supported APIs. Also, by calling getType API on tabContext, you can identify the type of the opened tab. The type could be the Browser tab, MSI tab, or MSI sub tab.

> **Note:** TabOpen is an event listenable from GlobalContext;

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>

 const frameworkProvider: IUiEventsFrameworkProvider =await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');

 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

```
 const payload: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(payload, (response: IEventResponse) => {
 const tabOpenResponse = response as ITabEventResponse;
 const tabContext: ITabContext = tabOpenResponse.getResponseData();
 const type = tabOpenResponse.getType();
 })
```

Here's a JavaScript example:

```
const frameworkProvider =await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const payload =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(payload, (response) => {
 const tabContext = response.getResponseData();
 const type = tabOpenResponse.getType();
 })
```

TabClose Event

The TabClose event subscription gives a notification about a browser tab, a new window tab close event, or a new MSI tab close event happening in any already opened browser tabs with the Service Center application loaded.

The event response will give the ITabInfo object of the closed tab, on top of which you can call gettabId(), getMSITabId(), and getMSISubTabId() to get the browser tabId, MSI tabid, and MSI sub tab id of the closed tab respectively.

Note: TabClose is an event listenable from GlobalContext.

Here's a TypeScript example:

```
const subscribeTabClose = async () => {

 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

 const payload: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 globalContext.subscribe(payload, (response: IEventResponse) => {
 let responseData = response as ITabCloseEventResponse;
 console.log(responseData.getResponseData().getTabId()); // Opened Tab's identifier
 console.log(responseData.getEventName()); // 'cxEventBusTabCloseEvent'
 console.log(responseData.getContext()); // contextObject info
 });
 }
```

Here's a JavaScript example:

```
const subscribeTabClose = async () => {

 const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const payload = frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabCloseEvent');
 globalContext.subscribe(payload, (responseData) => {
```

```
console.log(responseData.getResponseData().getTabId()); // Opened Tab's identifier
console.log(responseData.getEventName()); // 'cxEventBusTabCloseEvent'
console.log(responseData.getContext()); // contextObject info
});
}
```

TabChange Event

The TabChange event subscription gives a notification about a browser tab or an MSI tab switch happening in the Service Center application.

The user receives a notification of all tab-to-focus-out events once it adds a subscription to this event. It gives the newly focused tab's context in getCurrentTab() and the previous tab's context in getPreviousTab methods in the event response.

Note: TabChange is an event listenable from GlobalContext.

Here's a TypeScript example:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const payload: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabChangeEvent');
 globalContext.subscribe(payload, (response: IEventResponse) => {
 let responseData = response as ITabChangeEventResponse;
 let tabChangeResponse: ITabChangeResponse = responseData.getResponseData();
 let currentTabContext: ITabContext = tabChangeResponse.getCurrentTab();
 let previousTabContext: ITabContext = tabChangeResponse.getPreviousTab();
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const globalContext = await frameworkProvider.getGlobalContext();
 const payload =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabChangeEvent');
 globalContext.subscribe(payload, (responseData) => {
 let tabChangeResponse = responseData.getResponseData();
 let currentTabContext = tabChangeResponse.getCurrentTab();
 let previousTabContext = tabChangeResponse.getPreviousTab();
 });
```

Set Up and Try UI Events Framework

In this section, you'll learn how to set up a new Customer App and integrate it into the Service Center application.

This section covers only a basic setup, along with an example of a basic publish operation and a subscribe to an event use case.

Here's what we'll look at:

1. Setting TypeScript in your third-party application.
2. Add UI Events Framework library to your third party application.
3. Load the your application in Service Center.
4. Subscribe to a Tab Change event.
5. Publish a Set Field Value operation.

Set Up Your Third-Party Application

This example uses TypeScript to extend the JavaScript.

Make sure the Node.Js file is installed in your local development directory. The commands mentioned (in bold) can be executed in VS Code terminal.

1. Create a project folder and open it in any IDE - Recommended VS Code.
2. Initialize an npm project and install TypeScript and HTTP-server. npm init -y
3. Install TypeScript transpiler and HTTP-server globally. npm install tsc http-server -g . ' http-server' is used for local deployment.
4. Now run **`Create tsconfig.json`**:

```
{
 "compilerOptions": {
 "target": "es6",
 "module": "commonjs",
 "strict": true,
 "outDir": "out",
 "sourceMap": false,
 "lib": ["es2015.promise","es6","ES2016","dom"]
 }
}
```

5. Now run **`Create main.ts`**.

```
function run(name: string) {
 console.log('Hi ', name)
}
```

6. Transpile TypeScript to JavaScript using **`Command: tsc -w`**. This creates a file **`main.js`** in the out file. (Notice that typedefinition :string in main.ts is removed in main.js).
7. Create a file **`index.html`**. Notice that in index.html, the main.js file is loaded from "out/" directory, which will be generated during transpile.

```
<html>
<head>
 <script src="./out/main.js"></script>
</head>
<body>
 <button onclick="run('uef')">run</button>
</body>
</html>
```

The **`host index.html`** and **`out/main.js`** and go to the hosted URL in the browser.

Add the UEF Library to Your Application

The library name is ui-events-framework-client.js

A client application can be embedded in the Service Center application in 4 different ways:

- Load the client application in an IFrame
- Custom js in-app pages
- JavaScript in custom pages
- Custom Component CCA

Here's an example of how you load the application in an iFrame.

You add the `ui-events-framework-client.js` file to the script section of index.html file.

```
<head>
<script src="https://static.oracle.com/cdn/ui-events-framework/libs/ui-events-framework-client.js"></script>
<script s
```

Load the Third-Party Application Inside Service Center

You use the `<oj-cx-svc-common-ui-events-container>` component to load external applications in Service Center. You install the component from Oracle Component Exchange and configure it in Service Center according to the instructions in the included Readme file. Its `url` attribute should point to the application that we've hosted locally. It can be hard coded or added through page-level variables.

Here's an example:

```
<oj-cx-svc-common-ui-events-container url="[[ $page.variables.url ]]"></oj-cx-svc-common-ui-events-
container>
```

Subscribe to the Tab Change Event

Update the `run()` function in main.ts file with code to handle the Tab Change event. Now the code will look like the example which follows. More details are available in description of the Subscribe API in *Invoke the APIs*.

Add `uiEventsFramework.d.ts` to `main.ts` as shown in the following example:

```
/// <reference path="uiEventsFramework.d.ts"/> // Adding the type definitions for type checking
async function run () {
 // Creating an instance of UEF. This should be done only once in an app - usually during
the app's initialization.
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');

 // getting global context. GlobalContext has information about......
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

 // Currently, we can subscribe to ContextClose, ContextOpen, DataLoad, FieldValueChange, OnAfterSave,
OnBeforeSave, TabChange, TabClose, TabOpen
 const eventRequest: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusTabChangeEvent');

 // registering a subscription in the globalContext
 globalContext.subscribe(eventRequest, (response: IEventResponse) => {
 let responseData = response as ITabChangeEventResponse;
 let tabChangeResponse: ITabChangeResponse = responseData.getResponseData();
 let currentTabContext: ITabContext = tabChangeResponse.getCurrentTab();
 let previousTabContext: ITabContext = tabChangeResponse.getPreviousTab();
 console.log(tabChangeResponse, currentTabContext, previousTabContext);
 });
}
```

Because the compiler is started in watch mode (tsc -w), the main.js file will be autogenerated on saving main.ts. Reload the client application by right-clicking it and selecting Reload Frame from the drop-down list.

To test go to the Service Center home page and a service request. Keep the developer console open. Now switch the MSI tabs. The tabChangeResponse, currentTabContext, and previousTabContext details will be logged into the console on the tab switch.

Publish the Set Field Value Operation

Update the request type and subscription in the above code as given in the following example. Then, transpile the code and reload the iFrame.

```
async function run() {
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appName', 'v1');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: ISetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
ISetFieldValueOperationRequest);
 requestObject.field().setValue('ServiceRequest.Title', 'New Title');
 // publish event is happening here.
 // The result of the async operation is available on then(for success) and on catch ( for failure)
 recordContext.publish(requestObject).then((message) => {
 const response = message as ISetFieldValueResponse;
 console.log(response);
 }).catch((error: IErrorData) => {
 console.log(error);
 });
}
```

ContextOpen Event

This event is initiated when an object is opened in a VB application. In the event response, you can get the RecordContext object of the opened object and call all the object-related actions and events, such as setFieldValue, getFieldValue, fieldValueChange, an so on top of the RecordContext object.

Note: ContextOpen is an event listenable from TabContext.

The following code sample shows an example in TypeScript for subscribing to ContextOpen event:

```
/// <reference path="uiEventsFramework.d.ts"/>

 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
 tabContext.subscribe(requestObject, (message: IEventResponse) => {
 const response = message as IContextOpenEventResponse;
 const recordContext: IRecordContext = response.getResponseData();
 })
```

The following code sample shows an example in JavaScript for subscribing to ContextOpen event:

```
const frameworkProvider = await
```

```
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextOpenEvent');
 tabContext.subscribe(requestObject, (message) => {
 const recordContext = message.getResponseData();
 })
```

Context Close Event

This event is initiated when an object is closed in a VB application. This is a global-level event, so the user must call the subscribe API on top of global context to subscribe to this event.

Note: ContextClose is an event listenable from TabContext.

The following code sample shows an example in TypeScript for subscribing to ContextClose event:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (message: IEventResponse) => {
 const response = message as IContextResponse
 const context: IObjectContext = response.getResponseData();
 console.log(context.getObjectType());
 console.log(context.getObjectId());
 co
```

The following code sample shows an example in JavaScript for subscribing to ContextClose event:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusContextCloseEvent');
 tabContext.subscribe(requestObject, (response) => {
 const context = response.getResponseData();
 console.log(context.getObjectType());
 console.log(context.getObjectId());
 console.log(context.getTabId());
 });
```

DataLoad Event

This event is initiated when data is loaded for a particular object in a VB application. This event will always be fired after the ContextOpen event notification of a particular object. This is a record-specific event.

Note: DataLoad is an event listenable from RecordContext.

The following code sample shows an example in TypeScript for subscribing to DataLoad event:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusDataLoadEvent');
 recordContext.subscribe(requestObject, (response: IEventResponse) => {
 const message = response as IContextResponse;
 // custom code
 });
```

The following code sample shows an example in JavaScript for subscribing to DataLoad event:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusDataLoadEvent');
 recordContext.subscribe(requestObject, (response) => {
 // custom code
 });
```

Field Value Change Event

Here's a Typescript example of subscribing to FieldValueChange event where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: IFieldValueChangeEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
as IFieldValueChangeEventRequest;
 requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (message: IEventResponse) => {
 const response = message as IFieldValueChangeEventResponse;
 const fieldName: string = response.getResponseData().getFieldName();
 const newValue: string | boolean | number = response.getResponseData().getNewValue();
 const oldValue: string | boolean | number = response.getResponseData().getOldValue();
 });
```

The following code sample shows an example in JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
 requestObject.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (response) => {
 const fieldName = response.getResponseData().getFieldName();
```

```
const newValue = response.getResponseData().getNewValue();
const oldValue = response.getResponseData().getOldValue();
});
```

On After Save Event

This event is initiated after successfully saving a record in the VB application. This is a record level event, so the user must call the subscribe API on top of record context to subscribe to it.

This event response gives information about the saved object's type and its old and new identifiers. The old identifier and new identifier is different only when a save event happens on a new object. For example, while creating a new object UEF assigns a -ve identifier to it, and this -ve identifier is received in the ContextOpen event's subscription response. After saving, this object is assigned an actual identifier, and the new or actual identifier value can be tracked by adding the OnAfterSave Event listener to this new object.

Note: OnAfterSave is an event listenable from RecordContext.

The following code sample shows an example in TypeScript for subscribing to OnAfterSave event:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (message: IEventResponse) => {
 const response: IOnAfterSaveEventResponse = message as IOnAfterSaveEventResponse;
 const oldObjectId: string = response.getResponseData().getOldObjectId();
 const newObjectId: string = response.getResponseData().getObjectId();
 const objectType: string = response.getResponseData().getObjectType();
 });
```

The following code sample shows an example in JavaScript for subscribing to OnAfterSave event:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnAfterSaveEvent');
 recordContext.subscribe(requestObject, (response) => {
 const oldObjectId = response.getResponseData().getOldObjectId();
 const newObjectId = response.getResponseData().getObjectId();
 const objectType = response.getResponseData().getObjectType();
 });
```

OnBeforeSave Event

This event is initiated before initiating the API request to commit the data to the server. You can then do asynchronous operations in this callback or cancel this event. This event can wait for an asynchronous operation to be completed. With

this event, the user will get complete control of the save event that happens in VB by either approving or canceling this operation in VB.

> **Note:** OnBeforeSave is an event listenable from RecordContext

The following code sample shows an example in TypeScript for subscribing to OnBeforeSave event:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: IEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');

 recordContext.subscribe(requestObject, (response: IEventResponse) => {
 // custom code
 return new Promise((resolve, reject) => {
 });
 // the VB application save can be controlled by either resolving or rejecting this promise.
 // Resolving it would allow the save process, and rejecting the promise will cancel
the save process inside VB.
 // The save process in VB will wait until this promise is resolved or rejected.
 });
```

The following code sample shows an example in JavaScript for subscribing to OnBeforeSave event:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusOnBeforeSaveEvent');
 recordContext.subscribe(requestObject, (response) => {
 // custom code
 return new Promise((resolve, reject) => {
 });
 // the VB application save can be controlled by either resolving or rejecting this promise.
 // Resolving it would allow the save process and rejecting the promise will cancel the save process inside
 VB.
 // The save process will wait until this promise is resolved or rejected.
 });
```

InvokeServiceConnection Operation

Use this operation to invoke a service connection or REST call through the UI Events Framework.

> **Note:** InvokeServiceConnection Operation is an operation publishable from the GlobalContext level. Currently, only Internal Service connections marked as extensible are supported.

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
```

```
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();

 const restCallRequest: IServiceConnectionRequest =
(frameworkProvider.requestHelper.createPublishRequest('InvokeServiceConnection') as
IServiceConnectionRequest);
 restCallRequest.setServiceConnectionId('interactions/update_interactions');
 restCallRequest.setParameters({ "interactions_Id": "12345" });
 restCallRequest.setBody({ "StatusCd": "ORA_SVC_CLOSED" });

 globalContext.publish(restCallRequest).then((message: IOperationResponse) => {
 // custom code
 const response = (message as IServiceConnectionResponse).getResponseData();
 console.log(response.getStatus());
 console.log(response.getBody());
 }).catch((error: IErrorData) => {
 // custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();

 const restCallRequest =
(frameworkProvider.requestHelper.createPublishRequest('InvokeServiceConnection'));
 restCallRequest.setServiceConnectionId('interactions/update_interactions');
 restCallRequest.setParameters({ "interactions_Id": "12345" });
 restCallRequest.setBody({ "StatusCd": "ORA_SVC_CLOSED" });

 globalContext.publish(restCallRequest).then((message) => {
 // custom code
 const response = message.getResponseData();
 console.log(response.getStatus());
 console.log(response.getBody());
 }).catch((error) => {
 // custom code
 });
```

FocusTab Operation

This operation focuses on a particular browser tab, or an MSI tab opened in the current browser tab. This is an operation specific to the tab, so it should be executed on TabContext. The response will fetch the details of the current tab's context, meaning the focused tab's context and the previous tab's context.

> **Note:** FocusTab operation is publishable from the TabContext level.

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IOperationRequest =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const currentTab: ITabContext = (message as
IFocusTabResponseData).getResponseData().getCurrentTab();
```

```
 const previousTab: ITabContext = (message as
IFocusTabResponseData).getResponseData().getPreviousTab();
 }).catch((error: IErrorData) => {
 });
```

Here's a JavaScript example:

```
const publishFocusTab = async () => {

 const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const payload =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusFocusTabOperation');
 tabContext.publish(payload).then((message) => {
 const currentTab = message.getResponseData().getCurrentTab();
 const previousTab = message.getResponseData().getPreviousTab();
 }).catch((error) => {
 });
 };
```

CloseTab Operation

This operation is used to close a particular browser tab, or an MSI tab that's opened in the current browser tab. This is an operation specific to the tab and should be executed on TabContext. The response fetches the closed tab's identifier details in the ITabInfo object.

> **Note:** CloseTab Operation is an operation publishable from the TabContext level.

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const payload: IOperationRequest =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusCloseTabOperation');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const tabInfo: ITabInfo = (message as ITabCloseOperationResponse).getResponseData();
 const browserTabId: string = tabInfo.getTabId();
 const MSITabId: string = tabInfo.getMSITabId();
 }).catch((error: IErrorData) => {
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const payload =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusCloseTabOperation');
 tabContext.publish(payload).then((response) => {
 const tabInfo = response.getResponseData();
 const browserTabId = tabInfo.getTabId();
 const MSITabId = tabInfo.getMSITabId();
 }).catch((error) => {
 });
```

Pop Operations

Pop Operation

This operation is used to open any page from an external application and perform all the UEF supported operations and subscriptions on the TabContext object, which they will get as the response of this operation.

The response of this operation will have the TabContext object, which is a reference object of the opened browser tab or MSI tab in which the new page is opened. This operation enables the user to open the page either in a browser tab or in MSI tab. UEF supports opening Service Request, Case, Contact, Account, Lead, Activity, and Application UI Pages.

There's three methods used to open pages.

To open a Service Request, Case, Contact, Account, Lead, and Activity in IPopFlowInAppRequest, set the RecordType of the page. It can be a Service request, Case, Contact, Account, Lead, and Activity. RecordType is the only required value. In IPopFlowRequest, you can mention whether you need to open that in a new browser tab or the same tab, and if any parameter must pass to the page,. If you need to open the edit page of these pages, Service Request, Case, Contact, Account, Lead, and Activity, set the record in in **recordId**.

To open the Application page, In IPopFlowAppUIRequest, Set ApplicationUI Name, Flow, and Page. In IPopFlowRequest,you can mention whether you need to open that in a new browser tab or in the same tab, and if any parameter needs to pass to the page.

To open any page, In IPopFlowGenericRequest, Set Flow and Page. In IPopFlowRequest , you can mention whether you need to open that in a new browser tab or the same tab and if any parameter needs to pass to the page.

Open ServiceRequest Create Page using Pop

Here's a TypeScript example of IPopFlowInAppRequest opening the Create Service Request page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Open ServiceRequest Edit Page using Pop

Here's a TypeScript example of IPopFlowInAppRequest opening an Edit Service Request page with the detail view:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 requestObject.setRecordId('SR0000282245');
 requestObject.setInputParameters({view:'detail'});
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 requestObject.setRecordId('SR0000282245');
 requestObject.setInputParameters({view:'detail'});
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop Case Create Page

Here's a TypeScript example of IPopFlowInAppRequest used to open and Create a Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Case');
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop Case Edit Page

Here's a TypeScript example of IPopFlowInAppRequest opening an Edit Case page in a new browser tab:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Case');
 requestObject.setRecordId('CDRM2245');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 requestObject.setRecordId('CDRM2245');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop Contact Create Page

Here's a TypeScript example of IPopFlowInAppRequest used to open a Create Contact page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Contact');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

Pop Contact Edit Page

Here's a Typescript example of IPopFlowInAppRequest used to open an Edit Contact page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Contact');
 requestObject.setRecordId('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setRecordId('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

Pop Account Create Page

Here's a TypeScript example of IPopFlowInAppRequest used to open a Create Account page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

Pop Account Edit Page

Here's a TypeScript example of IPopFlowInAppRequest used to open an Edit Account page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setRecordId('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setRecordId('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await globalContext.publish(requestObject);
```

Pop Open AppUI Page

Here's a TypeScript example of the IPopFlowAppUIRequest used to open 360 pages.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowAppUIRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowAppUIRequest;
 requestObject.setApplicationUIName('advanced-customer-care');
 requestObject.setFlow('main');
 requestObject.setPage('main-start/main-dashboard');
 requestObject.setOpenPageInNewBrowserTab(true);
 requestObject.setInputParameters({ contactPartyNumber: "CDRM_943646", selectedAccountId: "9466225076" });
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setApplicationUIName('advanced-customer-care');
 requestObject.setFlow('main');
 requestObject.setPage('main-start/main-dashboard');
 requestObject.setOpenPageInNewBrowserTab(true);
 requestObject.setInputParameters({ contactPartyNumber: "CDRM_943646", selectedAccountId: "9466225076" });
 const response = await globalContext.publish(requestObject);
```

Pop Open Generic Page

Here's a TypeScript example of the IPopFlowGenericRequest to open the Article page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

```
const requestObject: IPopFlowGenericRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowGenericRequest;
requestObject.setFlow('service/ec/container/sr');
requestObject.setPage('view-article');
requestObject.setInputParameters({answerId: "10006003"});
requestObject.setOpenPageInNewBrowserTab(true);
const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setFlow('service/ec/container/sr');
requestObject.setPage('view-article');
requestObject.setInputParameters({answerId: "10006003"});
requestObject.setOpenPageInNewBrowserTab(true);
const response = await globalContext.publish(requestObject);
```

Pop Open Generic Page with Full URL

The following a TypeScript example for IPopFlowGenericRequest to open the Article page (full URL including https):

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IPopFlowUrlRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowUrlRequest;
requestObject.setUrl('https://url');
requestObject.setOpenPageInNewBrowserTab(true);
const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setUrl('https://url');
requestObject.setOpenPageInNewBrowserTab(true);
const response = await globalContext.publish(requestObject);
```

Pop Open New Object with Set Operation

The following TypeScript example shows an example of Pop with a Set operation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IPopFlowInAppRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
requestObject.setRecordType('ServiceRequest');
const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
const tabContext: ITabContext = response.getResponseData();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
```

```
const setFieldRequestObject: ISetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
ISetFieldValueOperationRequest);
setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
recordContext.publish(setFieldRequestObject).then((message) => {
const setFieldresponse = message as ISetFieldValueResponse;
//custom code
}).catch((error: IErrorData) => {
// custom code
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setRecordType('ServiceRequest');
const response = await globalContext.publish(requestObject);
const tabContext = response.getResponseData();
const recordContext = await tabContext.getActiveRecord();
const setFieldRequestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
recordContext.publish(setFieldRequestObject).then((message) => {
// custom code
}).catch((error) => {
// custom code
});
```

Pop Open Existing Object with Get Operation

The following TypeScript example shows an example of Pop with a Get operation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IPopFlowInAppRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
requestObject.setRecordType('ServiceRequest');
requestObject.setRecordId('SR0000282245');
const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
const tabContext: ITabContext = response.getResponseData();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const getFieldRequestObject: IGetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
IGetFieldValueOperationRequest);
getFieldRequestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

recordContext.publish(getFieldRequestObject).then((message) => {
const response = message as IGetFieldValueResponse;
const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
const problemDescriptionValue =
response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
// custom code
}).catch((error: IErrorData) => {
// custom code
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 requestObject.setRecordId('SR0000282245');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
 const recordContext = await tabContext.getActiveRecord();
 const getFieldRequestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
 getFieldRequestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);
 recordContext.publish(getFieldRequestObject).then((response) => {
 // custom code
 const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
 const problemDescriptionValue =
 response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
 }).catch((error) => {
 // custom code
 });
```

Pop Open New Object with a Set Mandatory Field and Save Operation

The following TypeScript example shows an example of Pop with the Set operation and Save operation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
 const recordContext: IRecordContext = tabContext.getActiveRecord();
 //set field value
 const setFieldRequestObject: ISetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest);
 setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
 recordContext.publish(setFieldRequestObject).then((message) => {
 const setFieldresponse = message as ISetFieldValueResponse;
 //save operation
 const saveRequestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
 recordContext.publish(saveRequestObject).then((message) => {
 const response = message as ISaveRecordResponse;
 const oldObjectId = response.getResponseData().getOldObjectId();
 const newObjectId = response.getResponseData().getObjectId();
 const objectType = response.getResponseData().getObjectType();
 }).catch((error: IErrorData) => {
 // custom code
 });
 }).catch((error: IErrorData) => {
 // custom code
 });
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
 const recordContext = await tabContext.getActiveRecord();
 //set field value
 const setFieldRequestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
 setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
 recordContext.publish(setFieldRequestObject).then((message) => {
 //save operation
 const saveRequestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
 recordContext.publish(saveRequestObject).then((response) => {
 const oldObjectId = response.getResponseData().getOldObjectId();
 const newObjectId = response.getResponseData().getObjectId();
 const objectType = response.getResponseData().getObjectType();
 }).catch((error) => {
 // custom code
 });
 }).catch((error) => {
 // custom code
 });
```

Field Name Mapping

Get Field Value Operation

This operation is used to fetch the current value of a field of a particular Record or Object (such as, to get the current value of the Title field of SR). You can set multiple fields in the request object of this operation to fetch multiple fields in one getFieldValue operation. This is a record-specific operation.

The following code sample shows an example in TypeScript of publishing GetFieldValue Operation where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IGetFieldValueOperationRequest = (frameworkProvider.requestHelper.
 createPublishRequest('cxEventBusGetFieldValueOperation') as IGetFieldValueOperationRequest);
 requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

 recordContext.publish(requestObject).then((message) => {
 const response = message as IGetFieldValueResponse;
 const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
 const problemDescriptionValue =
 response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in JavaScript for publishing GetFieldValue operation:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
 requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);
 recordContext.publish(requestObject).then((response) => {
 // custom code
 const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
 const problemDescriptionValue =
 response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
 }).catch((error) => {
 // custom code
 });
```

Set Field Value Operation

This operation is to update the value of a particular field of a particular Record or Object (for instance, updating the Title field of SR). This is a record-level operation; so, you must call the publish API on top of the record context.

The following code sample shows an example in TypeScript of publishing SetFieldValue Operation where field names are passed.:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: ISetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest);
 requestObject.field().setValue('ServiceRequest.Title', 'New Title');
 requestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');

 recordContext.publish(requestObject).then((message) => {
 const response = message as ISetFieldValueResponse;
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in JavaScript for publishing SetFieldValue operation:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 requestObject.field().setValue('ServiceRequest.Title', 'New Title');
 requestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');

 recordContext.publish(requestObject).then((message) => {
 // custom code
 }).catch((error) => {
 // custom code
```

```
});
```

Save Record Operation

This operation is used to save an open Record or Object in the UI. (such as save an SR form from the Edit page or save an SR from Create page).

The response of SaveRecordOperation gives information about the oldObjectId and actual object identifier after saving. This response is similar to the response from OnAfterSave event subscription response. The old identifier and new identifier is different only when a save event happens on a new object. For example, while creating a new object UEF assigns a -ve identifier to it, and this -ve identifier is received in the ContextOpen event's subscription response. After saving, this object is assigned an actual identifier from the database.

Note: The SaveRecord Operation is an operation publishable from RecordContext level.

The following code sample shows an example in TypeScript for publishing SaveRecord operation:

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
 recordContext.publish(requestObject).then((message) => {
 const response = message as ISaveRecordResponse;
 const oldObjectId = response.getResponseData().getOldObjectId();
 const newObjectId = response.getResponseData().getObjectId();
 const objectType = response.getResponseData().getObjectType();
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in JavaScript for publishing SaveRecord operation:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');

 recordContext.publish(requestObject).then((response) => {
 // custom code
 const oldObjectId = response.getResponseData().getOldObjectId();
 const newObjectId = response.getResponseData().getObjectId();
 const objectType = response.getResponseData().getObjectType();
 }).catch((error) => {
 // custom code
 });
```

Field Name Mapping

There are certain events and operations in the UI Events Framework which accept field names of an object or record and perform certain operations or listen to certain events.

Field level operations and events such as the FieldValueChange event, the GetFieldValue operation, and the SetFieldValue operation require field names' details contained in an object or record. An object's field name details can be obtained using the following API:

```
# metadata of a specific <object>
curl -X GET -u username:password https://servername.fa.us2.oraclecloud.com/crmRestApi/resources/11.13.18.05/
<object>/describe
```

Here are some examples of API usage for ServiceRequest, Case and Contact objects:

```
# serviceRequest object
curl -X GET -u username:password https://servername.fa.us2.oraclecloud.com/crmRestApi/resources/11.13.18.05/
serviceRequests/describe

# case object
curl -X GET -u username:password https://servername.fa.us2.oraclecloud.com/crmRestApi/resources/11.13.18.05/
cases/describe

# contact object
curl -X GET -u username:password https://servername.fa.us2.oraclecloud.com/crmRestApi/resources/11.13.18.05/
contacts/describe
```

Here the API to obtain metadata for all objects:

```
# metadata of all the objects' names with field details
curl -X GET -u username:password https://servername.fa.us2.oraclecloud.com/crmRestApi/resources/11.13.18.05/
describe
```

All fields must be mapped using the following format:

```
ObjectType.FieldName
```

As an example, an order to subscribe or perform certain operations on SR Title form, one must pass the field name as ServiceRequest.Title in the event, operation request object.

> **Note:** As part of the current release of ui-events-framework, we only support simple and LOV fields. Complex fields is supported in the coming releases.

Field Value Change Event

The following code sample shows an example in TypeScript for subscribing to FieldValueChange event where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
```

```
const requestObject: IFieldValueChangeEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent') as
IFieldValueChangeEventRequest;
requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

recordContext.subscribe(requestObject, (message: IEventResponse) => {
const response = message as IFieldValueChangeEventResponse;
const fieldName: string = response.getResponseData().getFieldName();
const newValue: string | boolean | number = response.getResponseData().getNewValue();
const oldValue: string | boolean | number = response.getResponseData().getOldValue();
});
```

The following code sample shows an example in JavaScript for subscribing to FieldValueChange event where field names are passed.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
requestObject.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

recordContext.subscribe(requestObject, (response) => {
const fieldName = response.getResponseData().getFieldName();
const newValue = response.getResponseData().getNewValue();
const oldValue = response.getResponseData().getOldValue();
});
```

Get Field Value operation

This operation is use to fetch the current value of a field of a particular Record or Object (for example, to get the current value of the Title field of SR). This is a record level operation and thus the user has to call the Publish API on top of record context in order to perform this operation.

The following code sample shows an example in Typescript for publishing GetFieldValue operation where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext: ITabContext = await frameworkProvider.getTabContext();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const requestObject: IGetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
IGetFieldValueOperationRequest);
requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

recordContext.publish(requestObject).then((message) => {
const response = message as IGetFieldValueResponse;
const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
const problemDescriptionValue =
response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
// custom code
}).catch((error: IErrorData) => {
// custom code
});
```

The following code sample shows an example in JavaScript for publishing GetFieldValue Operation where field names are passed.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
```

```
const tabContext = await frameworkProvider.getTabContext();
const recordContext = await tabContext.getActiveRecord();
const requestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);
recordContext.publish(requestObject).then((response) => {
// custom code
const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
const problemDescriptionValue =
response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
}).catch((error) => {
// custom code
});
```

Set Field Value operation

This operation is used to update the value of a particular field of a particular record, object (For example, to update the Title field of SR). This is a record level operation and thus the user has to call the publish API on top of record context in order to perform this operation.

The following code sample shows an example in Typescript for publishing SetFieldValue operation where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: ISetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest);
 requestObject.field().setValue('ServiceRequest.Title', 'New Title');
 requestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');

 recordContext.publish(requestObject).then((message) => {
 const response = message as ISetFieldValueResponse;
 // custom code
 }).catch((error: IErrorData) => {
 // custom code
 });
```

The following code sample shows an example in JavaScript for publishing SetFieldValue Operation where field names are passed.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();

 const requestObject =
 frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
 requestObject.field().setValue('ServiceRequest.Title', 'New Title');
 requestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');

 recordContext.publish(requestObject).then((message) => {
 // custom code
 }).catch((error) => {
 // custom code
 });
```

Field Value Change Event

Here's a Typescript example of subscribing to FieldValueChange event where field names are passed.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext: ITabContext = await frameworkProvider.getTabContext();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();

 const requestObject: IFieldValueChangeEventRequest =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
as IFieldValueChangeEventRequest;
 requestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (message: IEventResponse) => {
 const response = message as IFieldValueChangeEventResponse;
 const fieldName: string = response.getResponseData().getFieldName();
 const newValue: string | boolean | number = response.getResponseData().getNewValue();
 const oldValue: string | boolean | number = response.getResponseData().getOldValue();
 });
```

The following code sample shows an example in JavaScript example:

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent')
 requestObject.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (response) => {
 const fieldName = response.getResponseData().getFieldName();
 const newValue = response.getResponseData().getNewValue();
 const oldValue = response.getResponseData().getOldValue();
 });
```

Use the UEF Client Object in the Service Center application window

enableUefClient is an application level variable that can be exposed in Service Center.

The variable is used to enable the application window to consume UEF capabilities.

You enable or disable the application variable from the Variables tab in VB Studio as shown in the following screenshot.
<iINCLUDE SCREENSHOT>

With variable enabled, you can use the same steps you use to access the UEF capabilities from an external application directly in application window. You can use any custom JavaScript code inside Service Center to access it. This feature coexists with UEF objects consumed inside external applications.

Here's a JavaScript sample added in the Service Center container page to subscribe to TabOpen event using UEF:

```
UIEventsAPPFramework.UefClient.getUEFProvider().then((CX_SVC_UI_EVENTS_FRAMEWORK) => {
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID').then((uefProvider) => {
 uefProvider.getGlobalContext().then((globalContext) => {
 const requestObject = uefProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(requestObject, (response) => {
 console.log('Response from UEF API used in APP Window___', response);
 });
 });
 });
});
```

And here's an example of Subscribe Field Value Change in the VB application window using UEF:

```
async listenFieldValueChange() {
 const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
 const frameworkProvider = await uefProvider.uiEventsFramework.initialize('FVC');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent');
 requestObject.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (response) => {
 const fieldName = response.getResponseData().getFieldName();
 const newValue = response.getResponseData().getNewValue();
 const oldValue = response.getResponseData().getOldValue();
 console.log(`Field Name: ${fieldName}, oldValue: ${oldValue}, newValue: ${newValue}`);
 });
}
```

Extend the Service Center Side Pane

Service Center includes an extensible variable called `sidePaneButtons`. in service centre, This enables you to extend the sidePane buttons by assigning their desired sidePane icons to this variable .This variable is an array, of type `SidePaneIconProperties`. *SidePaneIconProperties*

```
"SidePaneIconProperties": {
 "ariaText": "string",
 "icon": "string",
 "label": "string",
 "sectionId": "string",
 "tooltip": "string",
 "visibility": "boolean",
 "buttonId": "string"
}
```

You can attach sections to these customised sidePane icons by creating sections in the extensible dynamic container available in container-page of service centre application.

> **Note:** The `sectionId` that you create section must match the sectionId you provide in the `sidePaneButtons` variable value.

Side Pane Operations

SidePane Operations

SidePaneContext is a UEF provider object that enables you to access the SidePane's events and perform actions on the SidePane in Service Center. You can get the sidePaneContext by calling the `getSidePaneContext` API provided in the UEF provider object.

Here's the syntax:

```
getSidePaneContext(sidePaneId: string): Promise<ISidePaneContext>;
```

Here's a TypeScript example of how you can access `SidePaneContext` by passing the `SidePane` ID:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const sidePaneContext:ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
```

Here's a JavaScript example of how you can access `SidePaneContext` by passing the `SidePane` ID:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
```

Supported Operations

UEF supports the following five sidePane operations:

- ExpandSidePane
- CollapseSidePane
- UpdateSidePane - setVisibility
- UpdateSidePane - setIcon
- UpdateSidePane - setSectionId

ExpandSidePane Operation

This operation is used to expand a particular side pane in Service Center. The ID of that sidePane should be passed while creating the SidePaneContext object.

Here's a TypeScript example:

```
const expandSidePaneContext = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
```

```
 const payload: IOperationRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('ExpandSidePane');
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response);
 }).catch(() => { })
}
```

Here's a JavaScript example:

```
const expandSidePaneContext = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('ExpandSidePane');
 sidePaneContext.publish(payload).then((response) => {
 console.log(response);
 }).catch((error) => { console.log(error); })
}
```

CollapseSidePane operation

This operation is used to close a particular side pane in Service Center. The ID of that sidePane should be passed while creating the SidePaneContext object.

Here's a TypeScript example:

```
const collapseSidePaneContext = async () => {
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const sidePaneContext: ISidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
const payload: IOperationRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CollapseSidePane');
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response);
}).catch((error) => { console.log(error); })
}
```

Here's a JavaScript example:

```
const collapseSidePaneContext = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CollapseSidePane');
 sidePaneContext.publish(payload).then((response) => {
 console.log(response);
 }).catch(() => { })
}
```

UpdateSidePane operation - setVisibility

This operation is used to update a particular side pane's section or icon or visibility in Service Center. The ID of that sidePane should be passed while creating the SidePaneContext object.SidePane visibility can be turned True or False by calling the `setVisibility` API.

Here's a TypeScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload:IUpdateSidePaneRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane') as IUpdateSidePaneRequest;
 payload.setVisibility(true);
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response)
 }).catch(() => { })
}
```

Here's a JavaScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane');
 payload.setVisibility(true);
 sidePaneContext.publish(payload).then((response) => {
 console.log(response);
 }).catch(() => { })
}
```

UpdateSidePane operation - setIcon

This operation is used to update a particular side pane's section or icon or visibility in Service Center. The ID of that sidePane should be passed while creating the SidePaneContext object.SidePane icon can be set to a different icon than what user has already customised, by calling the `setIcon` API.

Here's a TypeScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload:IUpdateSidePaneRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane') as IUpdateSidePaneRequest;
 payload.setIcon('newIcon');
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response)
 }).catch(() => { })
```

```
}
```

Here's a JavaScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane');
 payload.setIcon('newIcon');
 sidePaneContext.publish(payload).then((response) => {
 console.log(response);
 }).catch(() => { })
}
```

UpdateSidePane operation - setSectionId

This operation is used to update a particular side pane's section or icon or visibility in Service Center. The ID of the sidePane should be passed while creating the SidePaneContext object. The SidePane section can be set to a different section than what you may have already extended by calling the `setSectionId` API. The Section ID should match the Section IDs which used when the SidePane was extended.

Here's a TypeScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload:IUpdateSidePaneRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane') as IUpdateSidePaneRequest;
 payload.setSectionId('newSectionId');
 sidePaneContext.publish(payload).then((response: IOperationResponse) => {
 console.log(response)
 }).catch(() => { })
}
```

Here's a JavaScript example:

```
const updateSidePaneContext = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('UpdateSidePane');
 payload.setSectionId('newSectionId');
 sidePaneContext.publish(payload).then((response) => {
 console.log(response);
 }).catch(() => { })
}
```

Side Pane Events

SidePane Events

UEF supports two sidePane events.

- SidePaneOpen Event
- SidePaneClose Event

SidePaneOpen Event

This event is used to listen to sidePane open events of a particular sidePane. You must create the SidePaneContext object on top of which they call the `sidePaneOpen Event subscribe` API, by passing their interested sidePane's ID.

Here's a TypeScript example:

```
const listenSidePaneOpenEvent = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload: IEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusSidePaneOpenEvent');
 sidePaneContext.subscribe(payload, (res: IEventResponse) => {
 const response = res as ISidePaneOpenEventResponse;
 const responseData: ISidePaneData = response.getResponseData();
 responseData.getActiveSectionId();
 });
}
```

Here's a JavaScript example:

```
const listenSidePaneOpenEvent = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusSidePaneOpenEvent');
 sidePaneContext.subscribe(payload, (response) => {
 const responseData = response.getResponseData();
 console.log(responseData.getActiveSectionId());
 });
}
```

SidePaneClose Event

This event is used to listen to sidePane close events of a particular sidePane. You must create the SidePaneContext object on top of which they call the `sidePaneClose Event subscribe` API, by passing their interested sidePane's ID.

Here's a TypeScript example:

```
const listenSidePaneCloseEvent = async () => {
```

```
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const sidePaneContext: ISidePaneContext = await
 uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload: IEventRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusSidePaneCloseEvent');
 sidePaneContext.subscribe(payload, (res: IEventResponse) => {
 const response:ISidePaneCloseEventResponse = res as ISidePaneCloseEventResponse;
 const sidePaneCloseData: ISidePaneCloseData = response.getResponseData();
 const id: string = sidePaneCloseData.getId();
 })
}
```

Here's a JavaScript example:

```
const listenSidePaneCloseEvent = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const sidePaneContext = await uiEventsFrameworkInstance.getSidePaneContext('sidePaneId');
 const payload =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusSidePaneCloseEvent');
 sidePaneContext.subscribe(payload, (res) => {
 const sidePaneCloseData = response.getResponseData();
 const id = sidePaneCloseData.getId();
 });
}
```

Modal Window Operations

Modal Window Operations

Using an external application you can open a modal window or pop up window. Modal and pop up operations work on `ModalWindowContext` and also in the Tab context.

The differences between Modal and pop up are as follows:

- You can create only one modal. You can, however, create any number of pop ups.
- Modal is non interactable and pop up is interactable.

> **Note:** You can perform the modal window operation on a particular tab by providing the browser tab ID to get the tab context. If browser tab ID isn't given for getting tab context, the opener page context of MCA floating tool bar window will be the default tab context. Modal window operations supported only in browser Tab's tabContext only, not supported in MSI Tabs tab context.

You can get the `ModalWindowContext` by calling `getModalWindowContext` API provided in UEF provider object as shown here:

```
getModalWindowContext(): Promise<IModalWindowContext>;
```

And here's how you can access ModelWindowContext:

```
const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
const modalWindowContext:IModalWindowContext = await getModalWindowContext();
```

UEF supports four ModalWindow actions:

- OpenModal
- CloseModal
- OpenPopup
- ClosePopup

Note: All the model window operations work with global context and tab context.

OpenModal Action in Modal Window Context

Here's a TypeScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response: IModalWindowOperationResponse = await modalWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

CloseModal Action in Modal Window Context

Here's a TypeScript example:

```
const closeModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject: ICloseModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CloseModal') as ICloseModalWindowRequest;
 requestObject.setId('modal1');
 const response: IModalWindowOperationResponse = await modalWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
```

```
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const closeModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CloseModal');
 requestObject.setId('modal1');
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

OpenPopup Action in Model Window Context

Here's a TypeScript example:

```
const openPopup = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject: IOpenPopupWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenPopup') as IOpenPopupWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 requestObject.setTitle('Test title');
 requestObject.setClosable(true);
 requestObject.setStyle({width:'1000px', height:'1000px'});
 const response: IModalWindowOperationResponse = await modalWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const openPopup = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenPopup');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 requestObject.setTitle('Test title');
 requestObject.setClosable(true);
 requestObject.setStyle({width:'1000px', height:'1000px'});
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

ClosePopup Modal Window Context

Here's a TypeScript example:

```
const closePopup = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname', 'v1');
 const modalWindowContext: IModalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
```

```
 const requestObject: IClosePopupWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('ClosePopup') as IClosePopupWindowRequest;
 requestObject.setId('popup1');
 const response: IModalWindowOperationResponse = await modalWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const closePopup = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('appname',
 'v1');
 const modalWindowContext = await uiEventsFrameworkInstance.getModalWindowContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('ClosePopup');
 requestObject.setId('popup1');
 const response = await modalWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

Modal Window Operations in TabContext

Modal Window Operations in TabContext

You can perform Modal Window operations on Global Context, and browser Tab context. If you want to perform window operation in another tab, use `getTabContext` with browser ID.

> **Note:** You can perform the window operation on a particular tab by getting the tab context of a tab by providing the browser tab ID. Window operations are supported in the browser tab's tab context only, they aren't supported in MSI Tab. Use the `getCurrentBrowserContext` API to get the tabContext for the MCA floating toolbar window.

The following example shows the syntax for the `getTabContext` method:

```
getModalWindowContext(): Promise<IModalWindowContext>;
```

OpenModal Action in Tab Context

Here's a TypeScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const openerWindowContext: ITabContext = await uiEventsFrameworkInstance.getTabContext();
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response: IModalWindowOperationResponse = await openerWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
```

```
}
```

Here's a JavaScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app',
 'v1');
 const openerWindowContext = await uiEventsFrameworkInstance.getTabContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response = await openerWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

OpenPopup Action the Tab Context of a given TabId

Here's a TypeScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('tabId');
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 const response: IModalWindowOperationResponse = await tabContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app',
 'v1');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('tabId');
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 const response = await tabContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

Modal Window Operations in Current Browser Context

Modal Window Operations in Current Browser Tab Context

You can perform the Modal Window operation on Global Context, and browser Tab context. For the MCA floating toolbar window, to get the opener tabs Tab context, use getCurrentBrowserTabContext method. It returns opener tab context of MCA floating toolbar window.

Note: You can perform the window operation by getting the Tab context of the opener window of MCA floating tool bar window. If browser tab ID is not given, the opener page context of MCA floating tool bar window will be the default tab context. If the opener browser tab is closed, the tab context is retrieved from another opened browser tab.

The following example shows the syntax for the `getTabContext` method:

```
getCurrentBrowserTabContext(tabId?:string): Promise<ITabContext>;
```

OpenModal Action in Tab Context

Here's a TypeScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const openerWindowContext: ITabContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext();
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response: IModalWindowOperationResponse = await openerWindowContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app',
 'v1');
 const openerWindowContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext();
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('modal1');
 const response = await openerWindowContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

OpenPopup Action in Tab Context of a Given TabId

Here's a TypeScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app', 'v1');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext('tabId');
 const requestObject: IOpenModalWindowRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal') as IOpenModalWindowRequest;
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 const response: IModalWindowOperationResponse = await tabContext.publish(requestObject) as
 IModalWindowOperationResponse;
 const id:string = response.getResponseData().getId();
}
```

Here's a JavaScript example:

```
const openModal = async () => {
 const uiEventsFrameworkInstance = await CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('app',
 'v1');
 const tabContext = await uiEventsFrameworkInstance.getCurrentBrowserTabContext('tabId');
 const requestObject = uiEventsFrameworkInstance.requestHelper.createPublishRequest('OpenModal');
 requestObject.setURL('https://www.wikipedia.org/');
 requestObject.setId('popup1');
 const response = await tabContext.publish(requestObject);
 const id = response.getResponseData().getId();
}
```

Agent Info Operation in Global Context

Agent Info Operation in Global Context

This action can be performed on the Global context and the Browser Tab context.

Get First Name of Logged-in Agent

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
globalContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getFirstName()); // usage of getFirstName
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
globalContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Get Last Name of Logged-in Agent

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>

const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
```

```
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
globalContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getLastName()); // usage of getLastName
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
globalContext.publish(requestObject).then((response) => {
 console.log(response.getLastName());
});
```

Get Email Address of Logged-in Agent

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>

const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
globalContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getEmailAddress()); // usage of getEmailAddress
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
globalContext.publish(requestObject).then((response) => {
 console.log(response.getEmailAddress());
});
```

Get User Name of Logged-in Agent

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>

const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
globalContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getUserName()); // usage of getUserName
```

```
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
globalContext.publish(requestObject).then((response) => {
 console.log(response.getUserName());
});
```

Get Party ID of Logged-in Agent

Here's a TypeScript example:

```
/// <reference path="uiEventsFramework.d.ts"/>

const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
globalContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getPartyId()); // usage of getPartyId
});
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const globalContext = await frameworkProvider.getGlobalContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
globalContext.publish(requestObject).then((response) => {
 console.log(response.getPartyId());
});
```

Agent Info Operation in Tab Context

Agent Info Operation in Tab Context

This action can be performed on a particular tab by getting the tab context of a tab by providing the browser tab ID.

All the previously detailed functions work with Tab Context. The following block shows the syntax for getTabContext method:

The following example shows the syntax for the `getTabContext` method:

```
getTabContext(tabId?:string): Promise<ITabContext>;
```

Example Function with TabContext

The following code sample shows an example in TypeScript for getting agent's first name using `getFirstName` method using opener windows tab context.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerContext: ITabContext = await frameworkProvider.getTabContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
openerContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getFirstName()); // usage of getFirstName
});
```

The following code sample shows an example in JavaScript for getting agent's first name using `getFirstName` method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerContext = await frameworkProvider.getTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 openerContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example function with TabContext with given Tab Id

All the Agent info functions will work with Tab Context.

The following code sample shows an example in TypeScript for getting agent's first name using `getFirstName` method using tab context of given tab Id.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext: ITabContext = await frameworkProvider.getTabContext('tabId');
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
tabContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getFirstName()); // usage of getFirstName
});
```

The following code sample shows an example in JavaScript for getting agent's first name using the `getFirstName` method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext = await frameworkProvider.getTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Agent Info Operation in Current Browser Tab Context

Agent Info Operation in Current Browser Tab Context

This action can be performed on a particular tab by getting the tab context of a tab by providing the browser tab ID. For MCA floating toolbar window, to get the opener tabs tab context, use the `getCurrentBrowserTabContext` method which returns opener tab context of MCA floating toolbar window.

All the functions in global context previously detailed work with Tab Context.

The following example shows the syntax for the `getTabContext` method:

```
getCurrentBrowserTabContext(tabId?:string): Promise<ITabContext>;
```

Example Function with Current Browser TabContext

The following code sample shows an example in TypeScript for getting agent's first name using `getFirstName` method using opener windows tab context.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
openerContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getFirstName()); // usage of getFirstName
});
```

The following code sample shows an example in JavaScript for getting agent's first name using `getFirstName` method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerContext = await frameworkProvider.getCurrentBrowserTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 openerContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Example function with TabContext with given Tab Id

All the Agent info functions will work with Tab Context.

The following code sample shows an example in TypeScript for getting agent's first name using `getFirstName` method using tab context of given tab Id.

```
/// <reference path="uiEventsFramework.d.ts"/>
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
```

```
const requestObject: IOperationRequest =
 frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo') as
 CX_SVC_UI_EVENTS_FRAMEWORK.IOperationRequest;
tabContext.publish(requestObject).then((message: IOperationResponse) => {
 const response: IGetAgentInfoResponse = message as IGetAgentInfoResponse;
 console.log(response.getFirstName()); // usage of getFirstName
});
```

The following code sample shows an example in JavaScript for getting agent's first name using the `getFirstName` method.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
const requestObject = frameworkProvider.requestHelper.createPublishRequest('GetAgentInfo');
 tabContext.publish(requestObject).then((response) => {
 console.log(response.getFirstName());
});
```

Pop Operation in TabContext

Pop ServiceRequest Create Page using Current Browser Tab Context

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext('tabId');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop ServiceRequest Create Page using Current Browser Tab Context of Given Tab ID

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext('tabId');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop ServiceRequest Edit Page

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open an edit Service request page with view as details view.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 requestObject.setRecordId('SR0000282245');
 requestObject.setInputParameters({view:'detail'});
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used toopen an edit Service request page with view as details view.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
```

```
requestObject.setRecordId('SR0000282245');
requestObject.setInputParameters({view:'detail'});
requestObject.setOpenPageInNewBrowserTab(true);
const response = await openerTabContext.publish(requestObject);
const tabContext = response.getResponseData();
```

Pop Case Create Page

Here's a TypeScript example of IPopFlowInAppRequest used to open and Create a Case page:

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext: IGlobalContext = await frameworkProvider.getGlobalContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Case');
 const response: IPopFlowResponse = await globalContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

Here's a JavaScript example:

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const globalContext = await frameworkProvider.getGlobalContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 const response = await globalContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop Case Edit Page

The following code sample shows an example in TypeScript for the `IPopFlowInAppRequest` to edit a Case in a new browser tab.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Case');
 requestObject.setRecordId('CDRM2245');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to to edit a Case in a new browser tab.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Case');
 requestObject.setRecordId('CDRM2245');
```

```
requestObject.setOpenPageInNewBrowserTab(true);
const response = await openerTabContext.publish(requestObject);
const tabContext = response.getResponseData();
```

Pop Contact Create Page

The following code sample shows an example in TypeScript for the `IPopFlowInAppRequest` to open a Create Contact page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Contact');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to open a Create Contact page. .

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await openerTabContext.publish(requestObject);
```

Pop Contact Edit Page

The following code sample shows an example in TypeScript for the `IPopFlowInAppRequest` to open an Edit Contact page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Contact');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used toopen an Edit Contact page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Contact');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
```

```
const response = await openerTabContext.publish(requestObject);
```

Pop Account Create Page

The following code sample shows an example in TypeScript for the `IPopFlowInAppRequest` to open a Create Account page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used toopen a Create Account page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await openerTabContext.publish(requestObject);
```

Pop Account Edit Page

The following code sample shows an example in TypeScript for the `IPopFlowInAppRequest` to open an Edit Account page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used toopen an Edit Account page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('Account');
 requestObject.setRecordType('CDRM_123456');
```

```
requestObject.setOpenPageInNewBrowserTab(true);
const response = await openerTabContext.publish(requestObject);
```

Pop Open AppUI Page

The following code sample shows an example in TypeScript for the `IPopFlowAppUIRequest` to open a Customer 360 page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowAppUIRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowAppUIRequest;
 requestObject.setApplicationUIName('advanced-customer-care');
 requestObject.setFlow('main');
 requestObject.setPage('main-start/main-dashboard');
 requestObject.setOpenPageInNewBrowserTab(true);
 requestObject.setInputParameters({ contactPartyNumber: "CDRM_943646", selectedAccountId: "9466225076" });
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript for the `IPopFlowAppUIRequest` to open a Customer 360 page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setApplicationUIName('advanced-customer-care');
 requestObject.setFlow('main');
 requestObject.setPage('main-start/main-dashboard');
 requestObject.setOpenPageInNewBrowserTab(true);
 requestObject.setInputParameters({ contactPartyNumber: "CDRM_943646", selectedAccountId: "9466225076" });
 const response = await openerTabContext.publish(requestObject);
```

Pop Open Generic Page

The following code sample shows an example in TypeScript for the `IPopFlowGenericRequest` to open an article page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowGenericRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowGenericRequest;
 requestObject.setFlow('service/ec/container/sr');
 requestObject.setPage('view-article');
 requestObject.setInputParameters({answerId: "10006003"});
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript for the `IPopFlowGenericRequest` to open an article page

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
```

```
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setFlow('service/ec/container/sr');
requestObject.setPage('view-article');
requestObject.setInputParameters({answerId: "10006003"});
requestObject.setOpenPageInNewBrowserTab(true);
const response = await openerTabContext.publish(requestObject);
```

Pop Open Generic Page with Full URL

The following code sample shows an example in TypeScript for the `IPopFlowGenericRequest` to open an article page (full URL including https.)

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowUrlRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowUrlRequest;
 requestObject.setUrl('https://test.oracle.com/view-article?answerId=10006003');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
```

The following code sample shows an example in JavaScript for the `IPopFlowGenericRequest` to open an article page (full URL including https.)

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setUrl('https://test.oracle.com/view-article?answerId=10006003');
 requestObject.setOpenPageInNewBrowserTab(true);
 const response = await openerTabContext.publish(requestObject);
```

Pop New Object with Set Operation

The following code sample shows an example in TypeScript for Pop with the set operation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 const setFieldRequestObject: ISetFieldValueOperationRequest =
 (frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
 ISetFieldValueOperationRequest);
 setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
 setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
 recordContext.publish(setFieldRequestObject).then((message) => {
 const setFieldresponse = message as ISetFieldValueResponse;
```

```
//custom code
}).catch((error: IErrorData) => {
// custom code
});
```

The following code sample shows an example in JavaScript forPop with the set operation.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerTabContext = await frameworkProvider.getTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setRecordType('ServiceRequest');
const response = await openerTabContext.publish(requestObject);
const tabContext = response.getResponseData();
const recordContext = await tabContext.getActiveRecord();
const setFieldRequestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
recordContext.publish(setFieldRequestObject).then((message) => {
// custom code
}).catch((error) => {
// custom code
});
```

Pop Existing Object with Get Operation

The following code sample shows an example in TypeScript for Pop with the get operation.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
const requestObject: IPopFlowInAppRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
requestObject.setRecordType('ServiceRequest');
requestObject.setRecordId('SR0000282245');
const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
const tabContext: ITabContext = response.getResponseData();
const recordContext: IRecordContext = await tabContext.getActiveRecord();
const getFieldRequestObject: IGetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation') as
IGetFieldValueOperationRequest);
getFieldRequestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);

recordContext.publish(getFieldRequestObject).then((message) => {
const response = message as IGetFieldValueResponse;
const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
const problemDescriptionValue =
response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
// custom code
}).catch((error: IErrorData) => {
// custom code
});
```

The following code sample shows an example in JavaScript forPop with the get operation.

```
const frameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
```

```
const openerTabContext = await frameworkProvider.getTabContext();
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setRecordType('ServiceRequest');
requestObject.setRecordId('SR0000282245');
const response = await openerTabContext.publish(requestObject);
const tabContext = response.getResponseData();
const recordContext = await tabContext.getActiveRecord();
const getFieldRequestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusGetFieldValueOperation');
getFieldRequestObject.setFields(['ServiceRequest.Title', 'ServiceRequest.ProblemDescription']);
recordContext.publish(getFieldRequestObject).then((response) => {
// custom code
const titleFieldValue = response.getResponseData().getField('ServiceRequest.Title').getValue();
const problemDescriptionValue =
response.getResponseData().getField('ServiceRequest.ProblemDescription').getValue();
}).catch((error) => {
// custom code
});
```

Pop New Object with Set Mandatory Field and Save Operation

The following code sample shows an example in TypeScript for Pop with the set operation and save.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const openerTabContext: ITabContext = await frameworkProvider.getTabContext();
const requestObject: IPopFlowInAppRequest =
frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
requestObject.setRecordType('ServiceRequest');
const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
const tabContext: ITabContext = response.getResponseData();
const recordContext: IRecordContext = tabContext.getActiveRecord();
//set field value
const setFieldRequestObject: ISetFieldValueOperationRequest =
(frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation') as
ISetFieldValueOperationRequest);
setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
recordContext.publish(setFieldRequestObject).then((message) => {
const setFieldresponse = message as ISetFieldValueResponse;
//save operation
const saveRequestObject: IOperationRequest =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
recordContext.publish(saveRequestObject).then((message) => {
const response = message as ISaveRecordResponse;
const oldObjectId = response.getResponseData().getOldObjectId();
const newObjectId = response.getResponseData().getObjectId();
const objectType = response.getResponseData().getObjectType();
}).catch((error: IErrorData) => {
// custom code
});
}).catch((error: IErrorData) => {
// custom code
});
```

The following code sample shows an example in JavaScript forPop with the set operation and save.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext();
```

```
const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
requestObject.setRecordType('ServiceRequest');
const response = await openerTabContext.publish(requestObject);
const tabContext = response.getResponseData();
const recordContext = await tabContext.getActiveRecord();
//set field value
const setFieldRequestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSetFieldValueOperation');
setFieldRequestObject.field().setValue('ServiceRequest.Title', 'New Title');
setFieldRequestObject.field().setValue('ServiceRequest.ProblemDescription', 'New Problem Description');
recordContext.publish(setFieldRequestObject).then((message) => {
//save operation
const saveRequestObject =
frameworkProvider.requestHelper.createPublishRequest('cxEventBusSaveRecordOperation');
recordContext.publish(saveRequestObject).then((response) => {
const oldObjectId = response.getResponseData().getOldObjectId();
const newObjectId = response.getResponseData().getObjectId();
const objectType = response.getResponseData().getObjectType();
}).catch((error) => {
// custom code
});
}).catch((error) => {
// custom code
});
```

Pop Operation in Current Browser Tab Context

Pop Operation in Current Browser Tab Context

You can perform the pop operation can perform on Global Context and Browser Tab context. For the MCA floating toolbar window, to get the opener tabs Tab context, use the `getCurrentBrowserTabContext` method. This method returns the opener tab context of the MCA floating toolbar window.

> **Note:** Pop operations in the Global context won't work in MCA floating tool bar window. In that particular case, you can perform the pop operation by getting the tab context of the opener window of MCA floating tool bar window. If the browser tab ID is not given, the opener page context of MCA floating tool bar window will be the default tab context. If opener browser tab is closed, it gets tab context of another opened browser tab.

```
getCurrentBrowserTabContext(tabId?:string): Promise<ITabContext>;
```

Pop ServiceRequest Create Page using Current Browser Tab Context

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext('tabId');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

Pop ServiceRequest Create Page using Current Browser Tab Context of Given Tab ID

The following code sample shows an example in TypeScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext('tabId');
 const requestObject: IPopFlowInAppRequest =
 frameworkProvider.requestHelper.createPublishRequest('PopOperation') as IPopFlowInAppRequest;
 requestObject.setRecordType('ServiceRequest');
 const response: IPopFlowResponse = await openerTabContext.publish(requestObject) as IPopFlowResponse;
 const tabContext: ITabContext = response.getResponseData();
```

The following code sample shows an example in JavaScript where the `IPopFlowInAppRequest` is used to open a Create a Service Request page.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
 const openerTabContext = await frameworkProvider.getTabContext('tabId');
 const requestObject = frameworkProvider.requestHelper.createPublishRequest('PopOperation');
 requestObject.setRecordType('ServiceRequest');
 const response = await openerTabContext.publish(requestObject);
 const tabContext = response.getResponseData();
```

getCurrentBrowserTabContext

This method returns the current browser tabContext in the application if no optional parameter is passed. This method is exposed over `UefFrameworkProvider` object.

> **Note:** This API returns the browser tabContext from which the MCA floating toolbar window is opened, if its called from MCA toolbar window.

Here's the syntax:

```
getCurrentBrowserTabContext(browserTabId?: string): Promise<ITabContext>;
```

> **Note:** The current browser TabContext list will be returned if the browserTabId is not passed.

Parameter	Required	Description
browserTabId	No	TabId for the browser where the application is loaded.

Here's a TypeScript example of the `getCurrentBrowserTabContext`.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
```

Here's a JavaScript example the `getCurrentBrowserTabContext`.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const tabContext = await frameworkProvider.getCurrentBrowserTabContext();
```

getDependentTabs

This method returns the list of child tabs or dependent tabs of a tab. Use this API to track all the child tabs opened from a parent tab.

This API is an exposed over tabContext object.

Here's the syntax:

```
getDependentTabs(): Promise<ITabContext[]>;
```

Here's a TypeScript example of the `getDependentTabs`.

```
/// <reference path="uiEventsFramework.d.ts"/>
 const frameworkProvider: IUiEventsFrameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const currentTabContext: ITabContext = await frameworkProvider.getCurrentBrowserTabContext();
const childTabContexts: ITabContext[] = await currentTabContext.getDependentTabs();
```

Here's a JavaScript example the `getCurrentBrowserTabContext`.

```
const frameworkProvider = await
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID');
const currentTabContext = await frameworkProvider.getCurrentBrowserTabContext();
const childTabContexts = await currentTabContext.getDependentTabs();
```

Example

Here's a code sample which recursively finds out a tabContext's child tabs and inner child tabs and closes them:

```
async function getChildTabs(selectedTab, grandChildren) {
 return new Promise(async (resolve, reject) => {
 if (!grandChildren) {
 grandChildren = [];
 childTabsInitiatorPromiseResolve = resolve;
 }
 const newChildTabs = await selectedTab.getDependentTabs();
 grandChildren = grandChildren.concat(newChildTabs);
 newChildTabs.forEach(async (child) => {
 await getChildTabs(child, grandChildren);
 });
 if (newChildTabs.length === 0) {
 childTabsInitiatorPromiseResolve(grandChildren);
 }
 })
}

// Function to close the child tabs
function closeChildTabs(extAction) {
 getChildTabs(selectedTabContext).then((childTabs) => {
 if (childTabs) {
 const payload =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('cxEventBusCloseTabOperation');
 setCallbackCounter(extAction, 'call');
 childTabs.forEach((selectedChildTabContext, i) => {
 selectedChildTabContext.publish(payload).then((message) => {
 childTabsList.push = message.responseDetails.data.payload.tabId;
 if (i == childTabs.length - 1) {
 document.getElementById('allActionsResults').innerHTML = `<div><span id="tabId"> Closed Tabs:
 ${childTabsList}</span></div>`;
 }
 }).catch((error) => {
 console.log(error.message);
 });
 })
 }
 })
}
```

Subscription and Publish Custom Events

Subscribe Custom Events

You can subscribe to any custom event by invoking the UEF subscribe API on GlobalContext, RecordContext and Browser-TabContext.

The event to which subscription is to be added to is CustomEvent. If the event request payload user can pass the customEventName and add a subscription by this manner you'll be notified whenever an event gets generated on top of the context to which he has added the subscription to. You can get any data passed in the event and the custom event for which the event is fired by calling `getData()` and `getCustomEventName()` on the response object of the event subscription.

The examples which follow show adding a custom event subscription on different contexts:

Subscribe Custom Event on GlobalContext

Here's a TypeScript example for adding custom event subscription on `GlobalContext`.

```
const subscribeCustomEvent = async () => {
 const payload: ICustomEventSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
 payload.setCustomEventName('customEventName');
 const globalContext: IGlobalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.subscribe(payload, (message: IEventResponse) => {
 const response: ICustomEventSubscriptionResponse = message as ICustomEventSubscriptionResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Here's a JavaScript example for adding custom event subscription on `GlobalContext`.

```
const subscribeCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('customEventName');
 const globalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.subscribe(payload, (response) => {
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Subscribe Custom Event on TabContext (Browser Tab)

Here's a TypeScript example for adding custom event subscription on `TabContext`.

```
const subscribeCustomEvent = async () => {
 const payload: ICustomEventSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
```

```
 payload.setCustomEventName('customEventName');
 const tabContext:ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.subscribe(payload, (message: IEventResponse) => {
 const response: ICustomEventSubscriptionResponse = message as ICustomEventSubscriptionResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Here's a JavaScript example for adding custom event subscription on `TabContext`.

```
const subscribeCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('customEventName');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.subscribe(payload, (response) => {
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Subscribe Custom Event on RecordContext

Here's a TypeScript example for adding custom event subscription on `RecordContext`.

```
const subscribeCustomEvent = async () => {
 const payload: ICustomEventSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
 payload.setCustomEventName('customEventName');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 recordContext.subscribe(payload, (message: IEventResponse) => {
 const response: ICustomEventSubscriptionResponse = message as ICustomEventSubscriptionResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Here's a JavaScript example for adding custom event subscription on `RecordContext`.

```
const subscribeCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('customEventName');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 const recordContext = await tabContext.getActiveRecord();
 recordContext.subscribe(payload, (response) => {
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Publish Custom Events

You can publish any custom event by invoking UEF publish API on GlobalContext, RecordContext and BrowserTabContext.

The action name to which publish API is to be invoked is CustomEvent. In the publish request payload you can pass the customEventName and also any data along with that request by calling, setCustomEventName() and setEventPayload() APIs respectively.

By publishing a customEvent in this manner, will invoke a notification to all receivers who have previously added a subscription for this same event. Also, those receivers will get the data with which this publish request is invoked.

You can get any data passed in the publish request feedback by calling getData() and getCustomEventName() on the response object of the publish request.

The examples which follow show adding a custom event publish request in different contexts:

Publish Custom Event on GlobalContext

Here's a TypeScript example for adding a custom event publish request on `GlobalContext`.

```
const publishCustomEvent = async () => {
 const payload: ICustomEventRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent') as ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext: IGlobalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.publish(payload).then((message: IOperationResponse) => {
 const response: ICustomEventResponse = message as ICustomEventResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

Here's a JavaScript example for adding a custom event publish request on `GlobalContext`.

```
const publishCustomEvent = async () => {
 const payload: ICustomEventRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent') as ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const globalContext: IGlobalContext = await uiEventsFrameworkInstance.getGlobalContext();
 globalContext.publish(payload).then((message: IOperationResponse) => {
 const response: ICustomEventResponse = message as ICustomEventResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

Publish Custom Event on TabContext

Here's a TypeScript example for adding a custom event publish request on `TabContext`.

```
const publishCustomEvent = async () => {
 const payload: ICustomEventRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent') as ICustomEventRequest;
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 const response: ICustomEventResponse = message as ICustomEventResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

Here's a JavaScript example for adding a custom event publish request on `TabContext`.

```
const publishCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent');
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.publish(payload).then((message) => {
 console.log(message.getResponseData());
 console.log(message.getResponseData().getData());
 console.log(message.getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

Publish Custom Event on RecordContext

Here's a TypeScript example for adding a custom event publish request on `RecordContext`.

```
const subscribeCustomEvent = async () => {
 const payload: ICustomEventSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
 payload.setCustomEventName('customEventName');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 recordContext.subscribe(payload, (message: IEventResponse) => {
 const response: ICustomEventSubscriptionResponse = message as ICustomEventSubscriptionResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData());
 console.log(response.getResponseData().getCustomEventName())
 });
};
```

Here's a JavaScript example for adding a custom event publish request on `RecordContext`.

```
const publishCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent');
```

```
 payload.setCustomEventName('customEventName');
 payload.setEventPayload({ message: 'any data' });
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 const recordContext: IRecordContext = await tabContext.getActiveRecord();
 recordContext.publish(payload).then((message) => {
 console.log(message.getResponseData());
 console.log(message.getResponseData().getData());
 console.log(message.getResponseData().getCustomEventName());
 }).catch((err) => {
 console.log(err);
 });
};
```

Note: CustomEvent publish is not supported on MSI.

These event subscription and publish events enable you to define any new lifecycle event on a record which is not supported by UEF. A custom event subscription on a particular Record context will generate a notification only when a custom Event publish or a custom Event trigger from VB happens on that particular record context. No other record context subscriptions are notified. Similarly, a custom event subscription on a particular Browser tab context generates a notification only when a custom Event publish or a custom Event trigger from VB happens on that particular tab context. No other tab context subscriptions will be notified.

This capability of custom event also enables External Application to External Application communication inside VB through UEF.

Communication between External in VB through UEF

You can use custom events to enable iFrame to iFrame communication in VB. To enable this communication channel, you must define any custom event.

To do this the external application must listen to the event and add a subscription for the custom event. This event can then be published from another external application. The event subscription and event publish actions should happen over the same context with same event name (customEventName) in order to facilitate this communication.

Note: The custom event subscription on Global Context will only be listened to if the same custom event (same customEventName) publish happens on globalContext. The event notification won't be received if the publish request of the same event is happening on top of a different context like browser tabContext or recordContext.

Example of External Applications Communication through CustomEvent for customEventName (myEvent1)

Here's a TypeScript example for adding a custom event subscription in External Application 1 and publishing the same event from External Application 2 on top of same `TabContext`.

```
//Call the below method from External Application - 1
const subscribeCustomEvent = async () => {
 const payload: ICustomEventSubscriptionRequest =
 uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent') as
 ICustomEventSubscriptionRequest;
```

```
 payload.setCustomEventName('myEvent1');
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.subscribe(payload, (message: IEventResponse) => {
 const response: ICustomEventSubscriptionResponse = message as ICustomEventSubscriptionResponse;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData()); // { message: 'any data' }
 console.log(response.getResponseData().getCustomEventName()) // 'myEvent1'
 });
};

// Call the below method from External Application - 2
const publishCustomEvent = async () => {
 const payload: ICustomEventRequest =
 uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent') as ICustomEventRequest;
 payload.setCustomEventName('myEvent1');
 payload.setEventPayload({ message: 'any data' });
 const tabContext: ITabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.publish(payload).then((message: IOperationResponse) => {
 }).catch((err) => {
 console.log(err);
 });
};
```

Here's a JavaScript example for adding a custom event subscription in External Application 1 and publishing the same event from External Application 2 on top of same `TabContext`.

```
//Call the below method from External Application - 1
const subscribeCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createSubscriptionRequest('cxEventBusCustomEvent');
 payload.setCustomEventName('myEvent1');
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.subscribe(payload, (message) => {
 const response = message;
 console.log(response.getResponseData());
 console.log(response.getResponseData().getData()); // { message: 'any data' }
 console.log(response.getResponseData().getCustomEventName()) // 'myEvent1'
 });
};

// Call the below method from External Application - 2
const publishCustomEvent = async () => {
 const payload = uiEventsFrameworkInstance.requestHelper.createPublishRequest('CustomEvent');
 payload.setCustomEventName('myEvent1');
 payload.setEventPayload({ message: 'any data' });
 const tabContext = await uiEventsFrameworkInstance.getTabContext('browserTabId');
 tabContext.publish(payload).then((message) => {
 }).catch((err) => {
 console.log(err);
 });
};
```

Custom Event Subscription and Publish from VB

Custom event publish request can be listened inside VB and from VB the publish request can be acted upon.

You can either resolve or reject the request to send positive or negative feedback back to the requestor. Similarly Custom event can be triggered from VB so any receiver who has added a subscription to it will get a notification with the event data send from VB.

In the Service Center VB application there are two events in the container called uefCustomEvent and triggerUefCustomEvent which are used to to add a listener to a customEvent publish request or to trigger a customEvent from VB respectively:

uefCustomEvent Usage: Listening to and acting upon a Custom Event publish request inside VB

A custom event publish request on top of GlobalContext or on top of TabContext from the same browser tab can be listened ti inside VB and you can write your own logic on that request. You can either resolve or reject this request which will in turn give a positive or negative feed back to the publisher who is publishing the customEvent request.

Here's an example of the structure of the `event.event` object raised by `uefCustomEvent`.

```
{
markFailure: () => {},
markSuccess: (data: any) => {},
requestDetails: {
customEventName: string;
payload: any;
},
objectContext: {
objectId: null,
objectType: null,
tabId: string,
msiTabId: null,
msiSubTabId: null
}
}
```

Here are the steps:

1. Add a listener to uefCustomEvent.
2. In the event object, check the customEventName.
3. Call the markSuccess or markFailure callbacks which are available in the event object.
4. MarkSuccess callback execution will give a positive feedback to the customEvent publisher. Any data passed in the markSuccess callback is received at the publisher.
5. MarkFailure callback execution gives negative feedback to the customEvent publisher.

If you need to invoke the `uefCustomEvent` listener in browserTab-1 from another browserTab-2, you can call the publish api on top of browserTab-1's tabContext from browserTab-2 and perform the actions detailed in this step.

triggerUefCustomEvent Usage: Triggering a Custom Event from VB

You can trigger any customEvent from a VB application to allow any application suscribed to the same event to get a notification of it. You can pass any data with this custom event which is received at the receiver where the subscription has been added.

An event named **triggerUefCustomEvent** in the ServiceCenter container page accepts eventName and payload properties in its event payload object..

Here are the steps:

1. From an action chain in container-page, call Fire Event block (for example, on a button click, add an action chain that calls Fire Event block in the action chain for the button click).
2. Select the eventName as triggerUefCustomEvent in page level (container-page)
3. Click on the assign block for the payload object of the event
4. Give any customEventName in the eventName attribute of the event object.
5. Give any data as the object in the payload attribute of the event object.

Any application which has added a subscription for the same customEventName will get a notification whenever this action chain block is executed. It will also receive the data passed in the payload attribute. Thus VB is able to trigger any customEvent and pass data along with it so that any application which is interested in the same customEvent will get the notification and the event data passed from VB.

> **Note:** The event getting triggered from VB is triggered on top of GlobalContext and hence the same can be listened from any browser tabs, which has added a subscription for it on top of globalContext..

```
UIEventsAPPFramework.UefClient.getUEFProvider().then((CX_SVC_UI_EVENTS_FRAMEWORK) => {
 CX_SVC_UI_EVENTS_FRAMEWORK.uiEventsFramework.initialize('MyFirstExtensionID').then((uefProvider) => {
 uefProvider.getGlobalContext().then((globalContext) => {
 const requestObject = uefProvider.requestHelper.createSubscriptionRequest('cxEventBusTabOpenEvent');
 globalContext.subscribe(requestObject, (response) => {
 console.log('Response from UEF API used in APP Window___', response);
 });
 });
 });
});
```

Example: Subscribe Field Value Change in VB application window using UEF:

```
async listenFieldValueChange() {
 const uefProvider = await UIEventsAPPFramework.UefClient.getUEFProvider();
 const frameworkProvider = await uefProvider.uiEventsFramework.initialize('FVC');
 const tabContext = await frameworkProvider.getTabContext();
 const recordContext = await tabContext.getActiveRecord();
 const requestObject =
 frameworkProvider.requestHelper.createSubscriptionRequest('cxEventBusFieldValueChangeEvent');
 requestObject.setFields(['ServiceRequest.Title','ServiceRequest.ProblemDescription']);

 recordContext.subscribe(requestObject, (response) => {
 const fieldName = response.getResponseData().getFieldName();
 const newValue = response.getResponseData().getNewValue();
 const oldValue = response.getResponseData().getOldValue();
 console.log(`Field Name: ${fieldName}, oldValue: ${oldValue}, newValue: ${newValue}`);
 });
}
```